Universiteit Leiden

Master Computer Science

Computer Based Generation Of Optimal Body Structures

Name: Andrei Mestani
Student ID: s3649253

Date: [29/07/2025]

Specialisation: Data Science

1st supervisor: Niki van Stein
2nd supervisor: Thomas Bäck

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Modern vehicle designs must deliver lighter yet robust structures under tight development schedules. Early body-in-white (BIW) layout is still drafted largely by hand, a practice that limits the portion of design space engineers can explore before costly finite-element verification. This thesis introduces a *2-D genetic-algorithm demonstrator* that automatically generates feasible BIW skeletons from geometry. Structural members are represented as parametric beam primitives on a mirrored 60×30 grid and fitness is the weighted sum of five purely geometric penalties. Three GA (Genetic Algorithm) variants are evaluated under an identical budget of 24 000 objective evaluations. The **baseline** variant with static weights attains the best median final fitness, of proving that the algorithm can satisfy all hard constraints without physics coupling. The **adaptive** schedule, which re-weights objectives during the run, converges twice as fast in the first evaluations, but plateaus at a higher median fitness. The **repair** variant applies a local shift heuristic that mends broken joints after mutation. It lowers worst-case penalty values by 25 % relative to the adaptive schedule while maintaining similar mean performance. A subsequent Bayesian optimization search over crossover rate, mutation rate and penalty weights further improves the median fitness by 14 % without increasing the evaluation budget. These results demonstrate that disciplined constraint handling and diverse layout exploration are already achievable at the geometric stage. Future work will lift the encoding to shell-element 3-D models and integrate stiffness and crash simulations to form a full physics-coupled optimisation pipeline.

Contents

Chapter 1

Introduction

Automobiles represent a great segment of our urban infrastructure and are among the fastest-growing industries in the world today [1]. The evolution of design and engineering of vehicles serves as a great example of the pace this technology is shifting. This can be addressed from different perspectives and objectives. This thesis covers and discusses the design perspective and how this fragment affects the overall objectives of automobile construction. Traditionally, design methodologies predominantly depend on human knowledge. These approaches have proven to be successful, although they are inherently laborious and resource-heavy 2. This can slow down adjustments to new materials, advancing technologies, and evolving regulatory requirements. As a growing industry, the need for automobiles that are robust and aerodynamically optimized increases. Relying solely on traditional methods may prevent this industry from reaching a higher level of workflow.

This thesis investigates a genetic algorithm (*GA*) framework for body-in-white *(BIW)* optimization. Inspired by the ideas of evolution and natural selection, genetic algorithms can provide a framework for investigating large design spaces and developing solutions that satisfy intricate restrictions. One of the core contributions of this thesis is the integration of optimization through evolutionary algorithms *(EA)* 2.1.2. In particular, this study investigates how evolutionary computation might be used to improve structural elements inside a simplified representation of the design space. The purpose of this thesis is to create a foundational demonstrator or a first step towards an optimization framework. It focuses on using simple components as building blocks for the vehicle body. By utilizing genetic operators, this framework seeks on iteratively improving solutions by generating and evolving configurations using a genetic algorithm. It pays attention to integrating simplified domain-specific restrictions 2.1.1. More complicated areas of this field are yet to be considered in continuation. A complete design system involves a greater and precise analysis of numerous elements. This is a *Proof Of Concept* delivered by implementing fundamentals mentioned in Chapter 4.

Consequently, the project aims to answer the following research questions:

1. Can a conventionalized representation of a 2-D design space and components generate design layouts that satisfy geometric feasibility constraints inside a fixed evaluation budget? 5.2.1
2. Does shifting priorities during the optimization improve its fitness and convergence compared to static objectives?5.2.2

3. Can injecting repair steps after genetic operators deliver better fitness and designs?5.2.3
4. Can hyper-parameter search discover optimal setups compared to those predefined and hand-tuned to achieve optimal body arrangements?6.1

The following chapters develop a detailed description of the problem statement 2 and a review of the related work 3.

Chapter 2

Background

Designing and optimizing vehicle body structures in automotive construction has long been a challenging task [2]. Setting up a proper integration of various mechanical, electrical, and aesthetic components is crucial to these structures. Ensuring passenger safety, structural integrity, and efficient energy use is essential to a vehicle's overall performance. Historically, *Computer Aided Design (CAD)*, *Finite Element Method (FEM)* for virtual prototypes of structures, and expertise involvement were used to realize such ideas [3]. However, the repetitive nature of these approaches often limits their scalability and adaptability. The development of vehicle body design through computer optimization techniques and their effects on the automotive sector are all covered in this part.

2.1 Problem Statement

Traditionally, the design development relied heavily on trial and error. The process involved engineers utilizing *FEM* for design simulations based on an input from *CAD*. This designed input is still done by humans, and therefore, it presents a gap which we want to explore. Through a lengthy pipeline of steps, they would identify weaknesses or opportunities for improvement. Although effective, this approach for generating robust designs is labor-intensive and costly. In most cases, it fails to fully explore the design space due to time constraints. Modern approaches increasingly leverage computational methods to streamline and enhance the design process and deal with target balancing. [4][5].

One straightforward example of this trade-off is weight reduction, which has emerged as a major focus in vehicle design [6][7]. Its direct influence on fuel or electrical consumption and emissions has resulted in changes throughout the evolution of *BIW*. Regulatory standards, such as the *Corporate Average Fuel Economy (CAFE)* requirements [8] and the European Union's emissions targets [9], further emphasize the importance of lightweight construction. Meanwhile, vehicle safety regulations demand enhanced security in the case of an accident, requiring structural designs capable of handling energy effectively during collisions [10].

Computer-aided simulation tools such as *Finite Element Analysis (FEA)*[11] and *Computational Fluid Dynamics (CFD)* [12] have altered vehicle body design. These tools have made it possible for engineers to simulate variations of scenarios. Such cases start from crash tests to aerodynamic evaluations, without the need for physical prototypes. These simulations help optimize designs early in the development process by distinguishing stress centers, deformation

patterns, and failure points. Additionally, computer-based techniques make it easier to explore materials and geometries that would be difficult to test with conventional techniques.

2.1.1 Vehicle Body Structure Design

Vehicle body structures must fulfill a variety of requirements to reach the standards of modern automotive engineering [13]. These specifications include minimal weight for fuel efficiency and lower emissions, aerodynamic and driving performance, as well as structural strength.

The design of a body-in-white can be framed as the search for a vector of design variables

$$\boldsymbol{x} = \left[x_1, x_2, \ldots, x_n\right]^\top \in \mathcal{X}$$

that satisfies a set of requirements. This set of combinations should offer the best possible compromise between several engineering goals. Formally, we seek the feasible set

$$\mathcal{F} \;=\; \left\{ \boldsymbol{x} \in \mathcal{X} \;\middle|\; c_k(\boldsymbol{x}) = 0 \;(k = 1, \ldots, m),\; g_\ell(\boldsymbol{x}) \leq 0 \;(\ell = 1, \ldots, r) \right\}$$

, where c_k are equality constraints (symmetry, package envelopes, joint locations) and g_l are inequality constraints (minimum weld length, maximum panel deflection, material limits, etc.). Based on this set, the objective vector is evaluated

$$\boldsymbol{f}(\boldsymbol{x}) \;=\; \left[f_1(\boldsymbol{x}),\; f_2(\boldsymbol{x}),\; \ldots,\; f_p(\boldsymbol{x})\right]$$

. Its components typically represent conflicting targets such as mass, stiffness, crash absorption, noise vibration harshness, manufacturing costs, etc. No single design can minimize all objectives simultaneously. Therefore, the framework aims to identify Pareto-optimal solutions:

$\boldsymbol{x}^\star \in \mathcal{F}$ is Pareto-optimal if $\nexists\, \boldsymbol{y} \in \mathcal{F}$ such that $f_i(\boldsymbol{y}) \leq f_i(\boldsymbol{x}^\star)\ \forall i$ and $f_j(\boldsymbol{y}) < f_j(\boldsymbol{x}^\star)$ for some j.

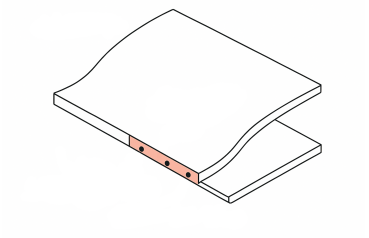
In practice, engineers usually transform the vector f into a scalar score such as a *weighted sum*. This way, the single-objective optimizers can be employed.

Connectivity, Isolation and Symmetry

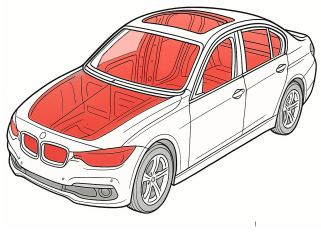
Vehicle bodies are not simply a collection of parts. They must behave as a single structure that shields the passenger region from external forces. In optimization terms, these three geometric constraints are particularly important:

- **Connectivity** All structural components must be linked to form one continuous component. This facilitates the transmission of crash loads without stress concentrations. In order to achieve this, engineers pay attention to how metallic parts are connected. This small-scale perspective for each connected component ensures a rigid overall structure at the end of the process. Formally, we can think of the structure as a contact graph $G(x) = (V, E)$. Nodes V are parts, and edges E represent weldable flange contacts. An optimal body design requires that all flange attachments are connected. This requirement is illustrated in Figure 2.1a.

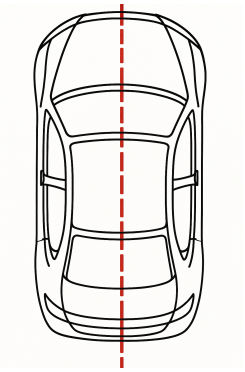
- **Isolation** Certain inner regions such as *battery box, interior or engine compartments* must be hermetically sealed from the exterior as shown in figure 2.1b. This is essential for reasons of safety, proper functioning, as well as a better driving experience. Let $\Omega_{\text{in}} \subseteq \mathcal{X}$ denote the set of grid cells that define those regions. A *flood-fill* which can represent any external flow of force that starts on the outer boundary may not reach any cell in Ω_{in}.
- **Symmetry** By principle, body manufacturing procedures usually prescribe bilateral symmetry. In addition of several components which change from the driver's side. It is in the vehicle's nature to have a symmetrical design. Every part placed at (x, y) must have a mirrored counterpart at $(W-1-x, y)$ as observed in figure 2.1c. Symmetry is modeled as a hard equality constraint on the geometry before any other target is estimated.



(a) **Connectivity** Local connectivity between an outer body panel and an inner reinforcement. The orange strip represents the flange area available for spot-welding and dots indicate individual weld nuggets.



(b) **Isolation** The red transparent regions mark the sealed zones that must remain isolated from external flow paths.



(c) **Symmetry** The dashed red line marks the symmetry axis. Every structural component on the left-hand side must have a mirrored counterpart on the right-hand side to ensure balanced load paths.

Figure 2.1: Illustration of the three geometric constraints

2.1.2 Evolutionary Methods

Evolutionary solutions offer a natural fit for the body optimization problem outlined in the previous section. They can explore large design spaces, including a considerable number of variables. An evolutionary search maintains an entire population of alternatives and therefore discovers trade-off solutions for a specific problem. The following sections will review *GA*s and a representation of our optimization in terms of *MOO*.

Genetic Algorithms and Multi-Objective Optimization

Modern vehicle body design can be defined as a multi-objective problem, requiring engineers to balance conflicting goals within one structure. This approach provides a method to deal with

these issues by taking balancing into account. *Multi-Objective Optimization* produces a set of optimal solutions, each of which represents a distinct balance between conflicting criteria. For example, one solution might prioritize structural robustness at the expense of slightly increased weight. Differently, one might favor weight reduction with moderate reductions in crash performance [14].

Genetic algorithms (GAs) are well structured to handle such multiobjective optimization problems [15][16]. These algorithms iteratively develop a population of potential solutions by imitating the laws of natural selection. *GA*s examine large design spaces and find solutions that satisfy predetermined constraints and goals by utilizing genetic operations that include *mutation*, *crossover*, and *selection*. The algorithm starts by creating an initial population, often at random or by seeding known individuals. Each individual is then evaluated by taking into consideration several objectives of the specific problem. Based on the scores for each objective, the *GA* performs four operations every generation:

- **Selection**: Based on its strategy, this step favors the better individuals and chooses them as parents.
- **Crossover**: Exchanges traits between two parents to create offspring
- **Mutation**: Applies small or random changes to an offspring in order to obtain variation.
- **Replacement**: This step forms the next generation by merging a subset of the parent population with the new offspring. This new generation is then re-evaluated and inserted again into the pipeline.

This cycle of steps continues until a stopping criterion is set. This last element can be a fitness threshold, a maximum number of generations, or a wall-time limit. This algorithm architecture maintains an evolving population and relies on objective rankings. Therefore it supports both continuous and discrete variables.

The main advantage of *GAs* is their flexibility. This trait gives them an advantage in vehicle body design, where the relationships between design variables such as material properties, geometric setups, and load distributions are often nonlinear and interdependent [17], [18]. In contrast to traditional optimization methods, which may become grounded by local optima, *GAs* work best in discovering diverse solutions across the design space. Their architecture can facilitate the optimization of various perspectives, such as the placement and orientation of structural elements, material distribution, and connectivity. Their flexibility has also ensured the inclusion of advanced manufacturing constraints, ensuring that designs are not only effective in theory but also practical to produce.

Search-Space Representation

Genetic Algorithms rely on the encoding that maps a physical design to the genome. This is an essential element that defines environmental variables on which genetic operators act. In structural optimization, the following representations are common:

- **Parameter Encoding**: Every *CAD* OR *FEM* variable such as dimension, control points, material IDs are directly encoded into the gene. This ensures an accurate definition of components. However, it can lead to a slow evaluation due to a large chromosome.

- **Voxel Grids**: Design space is discretized into cells that are either solid or empty. This implementation can be done either in 2D or 3D definitions. Transformation operators rely simply on cell manipulation, and finalized individuals require geometrical smoothing before manufacturing. [19]
- **L-System Encoding**: In this approach, the genome is a short string of production rules. During decoding, those rules are applied recursively to create a connection graph that is then mapped geometrically into a design grid. This technique facilitates repeated modules and symmetry but obstructs the transformation operators from manipulating chromosomes without breaking geometric constraints. [20]

Chapter 3

Related Work

The past decades have produced great work when it comes to computational optimization of vehicle structures. This work includes methods and evolutionary techniques for complex engineering layouts. To position the present thesis within that landscape, this chapter reviews prior work along two main axes. Firstly, we will discuss applications that target *BIW* or similar structural problems. Subsequently, we will review advances in genetic and other evolutionary algorithms that make those applications possible.

3.1 Evolutionary Design

Evolutionary computation was introduced to *body-in-white (BIW)* optimization in the 1990s [21][22]. Since this period, it has evolved into a collection of techniques able to search design spaces. Unlike *gradient-based* optimizers, genetic and other population-based algorithms do not require the objective to be smooth or even continuous[18]. This trait makes them useful whenever elements such as crashworthiness, large deformations, weld layout, or discrete part catalogs are involved. Throughout this period, the literature has evolved from *GA*s that drive a *FEM* model directly, to surrogate and constraint-aware variants which reduce evaluation cost and improve feasibility [23].

3.1.1 Early GA Studies For Body-In-White

Throughout time, evolutionary design has helped solve optimization problems with complex and conflicting objectives. *Genetic algorithms (GAs)* have the proper structure and can serve as a consistent approach to apply this evolutionary methodology.

Foundational work as the *Non-Dominated Sorting Genetic Algorithm (NSGA-II)* algorithm has made it possible for *GA* optimization to be applied in a wide range of domains related to structural design and engineering [24]. The NSGA-II method is an example of multi-objective optimization. It includes mechanisms to balance competing objectives such as weight reduction and structural integrity, which enables the simultaneous optimization of several criteria [21].

3.1.2 Surrogate-Assisted Approaches

In general, structural optimization for vehicle bodies focuses on objectives such as weight reduction, aerodynamic efficiency, and crash safety [25][26]. These are principal objectives

when it comes to vehicles meeting modern performance and regulatory standards. Consumer expectations for safety and sustainability increase as technology advances and innovations occur. *Finite element analysis (FEA)* and *Computational Fluid Dynamics (CFD)*, which offer information on stress distribution, impact resistance, and other factors, are used in traditional methods [3][12]. These techniques serve to assess the performance of designs. Nevertheless, these techniques tend to be more evaluative or informative rather than providing creative solutions. In addition, they require manual labor to produce preliminary designs. There exists a gap of unusual combinations that might perform better than typical designs, and this space is constrained by the dependence on human skill.

Recent studies extend well beyond simple response-surface fitting. Neural networks are now trained on small archives of crash or *NVH (Noise, Vibration, Harshness)* simulations and integrated thousands of times inside the genetic loop [27][28]. This step facilitates and speeds up the process by cutting evaluation time. Some authors employ active learning strategies [29], selecting new simulations where prediction uncertainty is highest. Others utilize a combination of sparse sets of detailed solid models in a multi-fidelity framework [30].

3.2 Constraint Handling

Evolutionary algorithms cope poorly with hard geometric rules, unless those rules are explicitly embedded. Three main strategies dominate the background for structural optimization. Penalty methods attach a cost term to any constraint violation. This brings the challenge of scaling the penalty so that feasible offspring are favored while still allowing the population to traverse narrow infeasible corridors [31]. Decoder approaches build feasibility into the chromosome itself, for example, by encoding only flange points and letting a constructive pipeline trace a valid attachment. Finally, repair operators leave the encoding untouched but apply a quick snap to the components or help by trimming overlaps [20]. Studies show that repair is especially effective when violations are sparse and local. This operator saves valuable building elements without inflating the objective with large penalties [32].

3.3 Symmetry And Modularity

In structural design, symmetry is a fundamental concept that comes due to the nature of the car body. In the meantime, that connectivity ensures that every part of a structure functions as a single component, symmetry promotes appropriate stress distribution, apart from aesthetic appeal. Although these ideas are frequently taken into account separately when optimizing designs, combining them at the same time presents considerable difficulties. For instance, if components don't align or interact as planned, preserving symmetry through mirroring or rotational transformations may unintentionally break connectivity. On the other hand, designs that prioritize connectivity over symmetry may be structurally wasteful or visually imbalanced, although they have strong functionality. Existing research has pushed towards addressing these principles separately, with symmetry-preserving transformations being widely used in architecture and mechanical design [33][20], and connectivity algorithms being applied in network and structural optimization [34]. However, few studies have successfully integrated these principles into a unified framework, particularly within the context of automated design.

3.4 Isolation

Incorporating restrictions such as isolated structures has been difficult to achieve, even if computational optimization has progressed. Generally, specific areas need to maintain their

functional independence, and in designs that include safety compartments, isolation is very important [35].

One of the key components of structural optimization is isolation, which is the process of creating enclosed or functionally different areas within a design. Isolation is essential for maintaining functionality and safety in domains including biotechnology, architecture, and vehicle design. Vehicle compartments, for example, need to be sealed off to safeguard occupants in the case of an accident, while bio-engineered tissues need separate areas for certain cellular processes. Isolation is rarely included as a primary requirement in automated design frameworks, despite its significance [36]. In the majority of cases, current methods either address isolation manually or place it lower in the scale of importance by establishing it as a secondary evaluation criterion. The ability of computational techniques to completely automate and revolutionize the design process is limited by this challenge.

Chapter 4

Methodology

This research presents a genetic algorithm *(GA)*-based framework for a **2D Proof-Of-Concept**. It focuses on adhering to domain-specific constraints and operates through a systematic workflow. This pipeline includes initialization, evaluation, genetic operations, and selection. During initialization, structural elements are randomly placed within a defined design space, with symmetry achieved through mirroring to ensure balanced configurations. The evaluation phase involves a multi-criteria fitness function, incorporating metrics such as isolation, connectivity, and efficiency. These metrics guide the algorithm toward generating designs that not only meet predefined objectives but also optimize geometrical parameters, such as part usage and structural connectivity.

The framework employs genetic operations, such as crossover and mutation, to iteratively refine design candidates by exploring a wide search space of potential solutions. Crossover strategies merge isolated regions from parent structures, fostering offspring with enhanced isolation and connectivity, while mutation introduces diversity by altering attributes like shape placement and orientation. The selection mechanisms prioritize solutions with optimal fitness, allowing the algorithm to evolve over generations. The following sections describe in detail the techniques and methods utilized for the framework.

4.1 Design Space And Representation

The design space for this optimization problem was created based on a structured 2D grid with dimensions of *60x30* cells. The restriction to two dimensions deliberately isolated geometric feasibility. Future research will lift this to shell-element 3-D models. Although simple, it provided an effective representation for generating and optimizing structural designs. This grid was divided into symmetrical halves which were separated by a central axis that serves as a mirroring boundary. Each cell in the grid was denoted by a number that assigns its specific role in the design space. This enforces and represents constraints during the design process. Boundary cells were represented by the value 0, and they defined the contour of the design space. This perimeter confined the free area and restricted the placement of structural elements beyond these limits. The inner area of the design space was marked with value 1, and it represents the available region that can be utilized for parts to be positioned and manipulated. This was the proper design space where the optimization takes part and was made use of. Additionally, in the center of this space was placed a rectangular area marked with the value 8. This square indicates the interior of the vehicle and plays a great role in guiding the optimization forward

and enforcing rules in the design space. The full design space can be seen in 4.1.

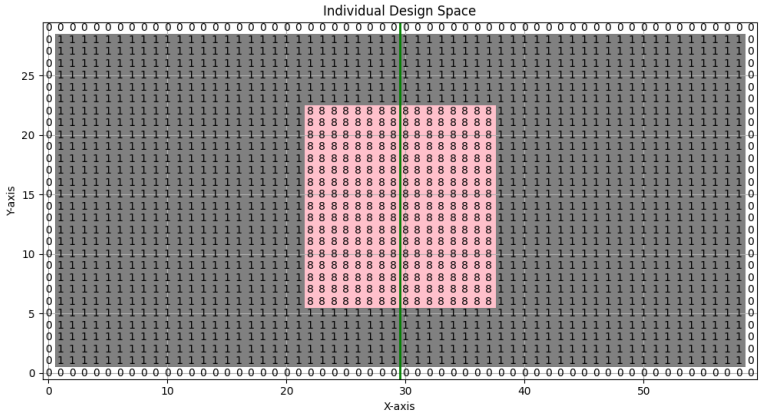


Figure 4.1: Design Space Visualization

Table 4.1: Cell–value legend for the design grid. Each cell stores one integer that drives both visualisation and objective evaluation: **0** marks the fixed outer boundary, **1** is empty, the range **2–98** encodes catalog parts (each part has a specific number assigned) *except for the single reserved value* ***8***, which marks the cabin interior that must remain sealed. The flood-fill routine converts any cell that becomes cut off from the exterior to **99**.

Value	Representation
0	Fixed boundary of the design window (non–traversable)
1	Free/unused cell inside the usable half–grid
2-98	Occupied by component
8	Pre-defined *interior* region that must be sealed
99	*Isolated* cell after component placement (label added during evaluation)

4.1.1 Simulation Of Parts

The data utilized to handle this optimization was created in a straightforward way that represents vehicle parts. The part catalog contains a set of structured delineations for various geometric shapes used in the design space. These shapes were parametrically defined, and based on the specific requirements, these parameters can vary. Each part was described by its *type*, *flexible parameters*, *starting* and *ending* points, *flanges*, and *normal vectors*. All shape parameters were interpreted on the 2-D grid. Thickness and out-of-plane effects were deliberately ignored at this stage. This systematic representation allowed these parts to integrate into the evolutionary algorithm's workflow and ensured effective manipulation and evaluation of each property.

One of the key aspects of this representation is the flanges and their norm vectors as seen in figure 4.2. This definition served as the essential component upon which one of the project objectives, connectivity, was composed. These flange definitions identify specific sections of the shape that can be connected to other shapes. Each flange has a defined start and end point, which marked the area and side of the designated connection for a specific shape. This interpretation was supported by a normal vector that indicates the direction of a potential connection. This workflow ensures that shapes can be placed and rotated within the design

Table 4.2: Parametric part catalog. *Free pars.* lists the adjustable length parameters (integer cell units) for each part. *Fl.* gives the number of weldable flange segments. *Norm vectors* indicate the outward directions of those flanges in the part's local frame. *Mirror-rot* encodes how the part's rotation code $r \in \{0, 90, 180, 270\}^\circ$ must be updated when the left half of the grid is reflected to the right.

Part	**Free pars.**	**# Fl.**	**Norm vectors** $\vec{n}$	**Mirror-rot.** $(^\circ)$
Straight line	ℓ	2	$\{\pm x\}$	$\{0\mapsto 180, 90\mapsto 90, 180\mapsto 0, 270\mapsto 270\}$
L-shape	ℓ_1, ℓ_2	4	$\{\pm x, \pm y\}$	$\{0\mapsto 270, 90\mapsto 180, 180\mapsto 90, 270\mapsto 0\}$
Angular left	ℓ_1, ℓ_2	4	$\{\pm x, \pm y\}$	$\{0\mapsto 180, 90\mapsto 270, 180\mapsto 0, 270\mapsto 90\}$
U-shape	ℓ_1, ℓ_2	6	$\{\pm x, +y\}$	$\{0\mapsto 0, 90\mapsto 270, 180\mapsto 180, 270\mapsto 90\}$
Angular right	ℓ_1, ℓ_2	4	$\{\pm x, \pm y\}$	$\{0\mapsto 90, 90\mapsto 0, 180\mapsto 270, 270\mapsto 180\}$

space while maintaining structural and functional relationships. One last addition to each part's properties was the *mirror-rotation* feature, which defines the change of orientation when a part is mirrored. This detail was added to preserve symmetry in the design space.

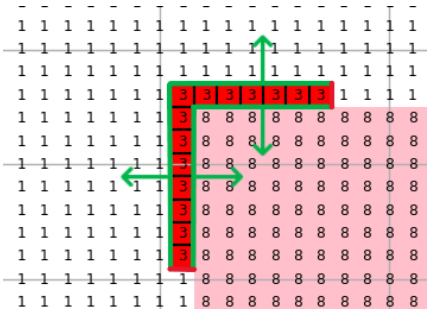


Figure 4.2: Visualization of an L-Rib with denoted flanges and norm vectors. The green areas denote the flanges of legal connections and the respective norm vectors.

Every individual layout was stored in a fixed-length chromosome. The chromosome was composed of successive six-gene blocks. As seen in table 4.3, each block corresponds to a single part from the catalog and contains respectively: the catalog index, its parametric lengths, the anchor-cell coordinates inside the design grid, and a discrete rotation code. The maximum number of parts allowed in the body design is $B = 96$, and every chromosome has exactly $Bx6 = 576$ integer genes. In the case that a design uses fewer than B parts, the remaining blocks were simply padded with a sentinel value. In this way, the genome length stays constant while the effective part count varies. In the decoding stage, the algorithm scans the chromosome and ignores any padded blocks. It instantiates the specified shape with its stored parameters, applies the encoded rotation, and deposits the part with its flange and normal vectors attached.

In order to establish a common framework for generating continuous approximations of parts, an added functionality was included. Once a block in the chromosome is decoded into the tuple, the engine calls that description into an explicit list of boundary points and flange normal vectors into the global grid coordinates. This functionality works by sampling points along the geometric outlines of the shapes and allows for a representation of their physical dimensions and connections. Through this method, it was made possible to support dynamic generation and placement of parts in the grid-based design space. This standard parametrized approach ensured consistency throughout the optimization process.

```
  0, 19, 17,  2, 13,   0,  1, 18,  0,  1, 14,   0,
  0,  8, 12,  3, 14,   0,  1, 14,  0,  4, 15, 270,
  1, 12,  0,  7, 16, 270,  1, 13,  0,  4, 16,   0,
  1, 10,  0,  1, 13, 180,  0, 16,  9,  2, 12, 270,
```

Listing 4.1: Complete 48-gene example of the chromosome

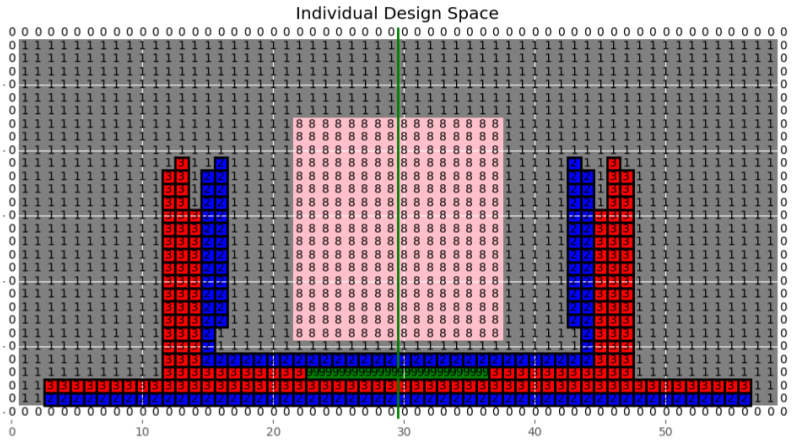


Figure 4.3: Example of the **chromosome–to–layout** decoding. The six active 6-gene blocks in the chromosome array 4.1 (see Table 4.3) are rendered on the left half of the grid and mirrored to the right.

4.2 Initialization

The evolutionary search began by synthesising an initial set of layouts that already respect the most restrictive design rules. The initialization started by retrieving the 60x30 rectangular grid and proceeded to fill the space with layouts generated at random but under certain feasibility rules. The static zones in the design space guaranteed that any initial design produced already satisfies essential constraints. One important detail that needs to be emphasized is that during the creation phase, the algorithm utilized only the left half of the design space for the placement of parts. This was done to simplify the overall optimization. One of the key elements of vehicle structures is symmetry. Although several components might vary for the driver's side, in general, cars have a symmetric construct. For this reason, in this task, it was generalized to simplify the process by creating individuals that contain parts only on the left of the design space. Afterwards, it was proceeded with analyzing how this fraction affects the whole structure by mirroring the components. The algorithm 1 delineates the entire creation process.

For each new individual, a random target count was drawn *(5-15 parts)*. Parts were selected one after another from the shape catalog, and a new part was attached to the structure at random. The function selects a part that was already placed and tries to establish a minimal connection with the new shape by retrieving its flanges and normal vectors. For each selection, the algorithm sampled geometric parameters, rotation, and anchor position. A candidate part was accepted only if it fulfills these minimal conditions:

- Lies entirely inside the free area designated for part placement and does not exceed design space boundaries.

Table 4.3: Six-gene block encoding used in the fixed-length chromosome. Each active block stores *one* catalog part. The six integers encode: the shape index s; one or two length parameters ℓ_1, ℓ_2; the anchor position (x, y) of the part's local origin on the left half-grid; and a discrete rotation code $r \in \{0, 1, 2, 3\}$ that maps to $\{0°, 90°, 180°, 270°\}$.

Gene	Symbol	Meaning / range
0	s	Shape-catalog index $\in \{0, \ldots, 4\}$
1	ℓ_1	Primary length parameter (4–19)
2	ℓ_2	Secondary length (0–19; ℓ_2=0 for straight line)
3	x	Anchor-cell x coordinate (0–29)
4	y	Anchor-cell y coordinate (0–29)
5	r	Rotation code $0{:}3 \mapsto \{0°, 90°, 180°, 270°\}$

- Shares at least one flange cell (valid or invalid 4.3.1) with an existing neighbor so that minimal connectivity is preserved.
- Does not overlap with an existing neighbor.

If no admissible position was found after a small number of tries, the algorithm simply skips that part and continues. This ensures quick termination in the case of crowded regions.

Algorithm 1 CREATE INDIVIDUAL

Require: shape catalog $\mathcal{C}$; empty grid $\mathcal{G} \in \{0, 1, 8\}^{30\times60}$; block capacity B=96; trial cap $T_{\max}$=20
$\mathbf{g} \leftarrow [-1]^{6B}$ ▷ padded chromosome
$n \leftarrow$ randint$(5, 15)$ ▷ target #parts
for $k \leftarrow 0$ **to** $n - 1$ **do**
  for $t \leftarrow 1$ **to** $T_{\max}$ **do**
    $(s, \ell_1, \ell_2) \leftarrow$ SAMPLE SHAPE($\mathcal{C}$)
    $(x, y, r) \leftarrow$ SAMPLE POSE(Left Half($\mathcal{G}$))
    $\mathcal{S} \leftarrow$ INSTANTIATE(s, ℓ_1, ℓ_2, r)
    if INSIDE FREE AREA$(\mathcal{S}, \mathcal{G})$ **and** TOUCHES NEIGHBOR$(\mathcal{S}, \mathcal{G})$ **and** $\neg$OVERLAPS$(\mathcal{S}, \mathcal{G})$ **then**
      encode $(s, \ell_1, \ell_2, x, y, r)$ into block k of $\mathbf{g}$
      draw $\mathcal{S}$ onto $\mathcal{G}$ ▷ reserve cells
      break ▷ placement accepted
    end if
  end for ▷ skip part if no feasible pose after $T_{\max}$ tries
end for
MIRROR LEFT TO RIGHT($\mathcal{G}$) ▷ enforce symmetry
return $\mathbf{g}$

4.2.1 Mirroring

Once the left half is complete, the entire configuration was mirrored based on the vertical center line. The initial attempt was to do this by having each part of the left side of the design space positioned twice by calculating its mirrored anchor position on the right side. In the catalog section, the last property of a part was the *mirror-rotation*, which was also used to define the orientation after establishing the mirrored anchor position. Simply re-positioning the part by utilizing the coordinates was not enough because it would be identical on the other side of the grid. In our case, we needed it to be mirrored. Each part has its own specific translation when it comes to this property and it depends on its orientation. However, this would increase the complexity of the pipeline as the number of parts for an individual can be high, and the number of individuals itself can increase based on the final configuration budget.

Consequently, mirroring was done by applying a reflection operation to the rectangular grid. Instead of computing an explicit mirroring for every catalog part, the framework enforced bilateral symmetry by reflecting the left-hand grid cells to the right-hand side in a single pass:

$$G_{x,\,W-1-y} \;:=\; G_{x,y}, \qquad \forall\, x \in \{0,\ldots,H-1\},\ \forall\, y \in \{0,\ldots,\lfloor \tfrac{W}{2} \rfloor - 1\}. \tag{4.1}$$

Here $G \in \mathbb{Z}^{H\times W}$ is the design grid, H its height and W its width (even). Each assignment copied the value of a cell on the left half $(y < W/2)$ to its mirrored column index $(W-1-y)$. The operation touches every cell only once and removes the need to track a part-specific *mirror-rotation* attribute.

This functionality copies every occupied cell from the left half of the grid to its symmetrical position on the right. Concretely, the algorithm walks through all rows and columns from the left boundary to the center line. For each cell in column y, it writes the same value in column $(cols-1)-y$. The grid already stores part identifiers on a per-cell basis, hence this one-pass reflection instantly produces a perfectly mirrored layout. The minimal constraints that we initially aimed for when creating an individual are duplicated on the right side without having to create a second copy of every part object. The operation runs in $O(HXW/2)$ time and allocates no additional memory. This made it less costly than recomputing a new anchor position and applying the *mirror-rotation* as previously mentioned.

The flaw of this operation stands exactly where its advantage is, because by simply mirroring the structure, we lose information about the parts reflected on the right side. In other words, the algorithm knows that a mirrored cell of a part is occupied, but it no longer has an explicit record of which catalog part produced that occupancy. This was a trade-off that could be accepted because we can apply evolutionary operators on one side and concurrently continue to evaluate the entire structure based on the information that we hold from this side. If, at a later stage, explicit information is required for the entire structure *(CAD Modeling)* , reconstruction can be performed by reapplying the original part list with the stored *mirror-rotation* rules. However, for our current optimization, the lighter cell-reflection approach was both sufficient and efficient.

The resulting layout was finally flattened into a six-gene block chromosome with unused blocks padded by an inactive marker and inserted into the initial population. The outcome was diverse, yet always feasible starting pool on which the evolutionary search can begin to act.

4.3 Evaluation

At the start of the evaluation phase, a fresh design grid was utilized for the algorithm to redraw the shapes encoded in the individual chromosome. It started by scanning the genome block by block, skipping any inactive markers, and converts the integer genes into a concrete part instance. After decoding, this information was used to drop that instance at the requested anchor position and rotation. Before the placement succeeds, a small check for collision with the design boundary or with already positioned components was done. In the case that such events occur, the attempt was aborted, and the shape was ignored. Each successful placement lays an identifier into its respective cell and appends a record to an in-memory list that contains: *position, rotation* and *type*. This list was later used for pairwise analyses, which serve to formulate our objectives. By the end of this cycle shown in algorithm 2, the grid holds an

explicit map of occupied and empty cells, while the list preserves the geometric parameters of every component standing on the left of the design space.

These two tools served as the basis for the evaluator to continue with the computation of five raw metrics that together will describe the structural quality of the candidate:

1. **Local Connectivity Penalty**: This first metric makes use of flange and normal vector information to estimate the quality of connections from one part to another.
2. **Global Connectivity Penalty**: This metric counts the number of connected components the structure contains. This is done by building a graph whose nodes are parts and whose edges are valid joints. Each extra component beyond the first adds one penalty unit.
3. **Shape Count Penalty**: This penalty serves as an estimate of the number of parts utilized in the individual. This metric aims to minimize the total number of parts, which increases efficiency.
4. **Isolation Penalty**: This metric allows monitoring the quality of interior isolation. Estimates the gaps that allow penetration to flow from the interior area to the usable region of the design space.
5. **Distance Penalty**: This last objective component compares the center of every part with the closest interior border. This metric aims to bring the idea of a tighter design to the algorithm.

Algorithm 2 EVALUATE INDIVIDUAL — decode genome and score penalties

Require: chromosome $\mathbf{g}$; weight vector $\boldsymbol{\omega} = (\omega_{\mathrm{LC}}, \omega_{\mathrm{GC}}, \omega_{\mathrm{SC}}, \omega_{\mathrm{IS}}, \omega_{\mathrm{DS}})$
$\mathcal{G} \leftarrow$ CLEAR GRID
$\mathcal{L} \leftarrow$ DECODE AND DRAW$(\mathbf{g}, \mathcal{G})$ ▷ §3.x

$p_{\mathrm{LC}} \leftarrow$ LOCAL CONNECTIVITY$(\mathcal{L})$
$p_{\mathrm{GC}} \leftarrow$ GLOBAL CONNECTIVITY$(\mathcal{L})$
$p_{\mathrm{SC}} \leftarrow$ SHAPE COUNT$(\mathcal{L})$
$p_{\mathrm{IS}} \leftarrow$ ISOLATION$(\mathcal{G})$
$p_{\mathrm{DS}} \leftarrow$ DISTANCE COMPACTNESS$(\mathcal{L})$

$f \leftarrow \omega_{\mathrm{LC}}\, o_{\mathrm{LC}} + \omega_{\mathrm{GC}}\, o_{\mathrm{GC}} + \omega_{\mathrm{SC}}\, o_{\mathrm{SC}} + \omega_{\mathrm{IS}}\, o_{\mathrm{IS}} + \omega_{\mathrm{DS}}\, o_{\mathrm{DS}}$
return $(f,\ [o_{\mathrm{LC}}, o_{\mathrm{GC}}, o_{\mathrm{SC}}, o_{\mathrm{IS}}, o_{\mathrm{DS}}])$

4.3.1 Local Connectivity

In car manufacturing, body stiffness is one of the most important conditions, whether we consider safety or quality. Although stiffness itself was not evaluated, local connectivity was used only as a geometric proxy. For an overall structure to be solid, it means that every metallic component needs to establish a proper connection to the others. In the modern automobile industry, such tasks are conducted by *spot-welding*. This is a dominant technique that allows for small or large metallic components to be attached. After a considerable number of parts, it produces the outer structure that we see as the body of the vehicle. 2.1.1 Additional methods are involved, but mostly for other materials.

To ensure sound weld lines in this geometric model, two geometric facts are considered:

- *Flange area*: The two metallic components need to share a certain surface size so that a full pattern of weld nuggets can be formed. This shared surface is essential to establish a solid connection between components.

- *Orientation*: The designated flanges mentioned above have a specific direction in which they can establish a connection. Physical aspects such as shape and size are considered when specifying the direction of a connection. This ensures stiffness and prevents detachment when the shared connection area is not sufficient to maintain a solid connection between two components.

To capture this requirement, the optimizer looked at every unordered pair of parts after they have been laid on the grid. For each pair, it scans the list of flange cells that were stored during placement. If it finds at least one grid node where a flange cell from part A coincides with a flange cell from part B and the two outward normals point in opposite directions, it increments one valid local connection. This procedure continued until all components have been checked and tested to add up the total number of such contacts.

Finally, the algorithm judged whether that total was "good enough". After calculating the total observed connections, it proceeds to estimate the theoretical maximum number of contacts. This estimation was done by looking at the length of the flanges involved in the individual and adding each flange cell. However, this theoretical maximum does not represent what we want when it comes to an overall optimal stiff structure. For components to be fully connected and every flange cell or normal vector to be exploited, it indicates that parts are overly connected. Therefore, the raw total of connections of an observed individual is first turned into a ratio between [0,1] depending on the theoretical maximum of this same individual. After this step, a soft global target band is defined. Ratios that fall within this target were considered acceptable and incur no penalty. Ratios below the band indicate that there is too little weldable overlap, and conversely, the ones that fall above indicate too much. Those layouts received an increasing penalty the further they deviate. In this way, the local connectivity metric steers the search towards designs whose parts can be welded properly and cheaply. It provides reliability while still giving the algorithm freedom to experiment with different arrangements.

Mathematically, the metric was computed in two steps. First, the observed–to–maximum contact ratio is:

$$r_{\mathsf{LC}} \;=\; \frac{C_{\mathsf{obs}}}{C_{\mathsf{max}}}, \qquad 0 \le r_{\mathsf{LC}} \le 1, \tag{4.2}$$

where C_{obs} is the number of valid flange overlaps found in the grid and C_{max} is the theoretical maximum for the same set of parts (sum of all flange lengths). Given an acceptable band $[r_{\mathsf{low}},\, r_{\mathsf{high}}]$. The raw penalty returned is:

$$p_{\mathsf{LC}} = \begin{cases} 0, & r_{\mathsf{low}} \le r_{\mathsf{LC}} \le r_{\mathsf{high}}, \\ (r_{\mathsf{low}} - r_{\mathsf{LC}}), & r_{\mathsf{LC}} < r_{\mathsf{low}}, \\ (r_{\mathsf{LC}} - r_{\mathsf{high}}), & r_{\mathsf{LC}} > r_{\mathsf{high}}. \end{cases} \tag{4.3}$$

4.3.2 Global Connectivity

Beyond the quality of individual small-scale connections, a body structure must act as a single component frame. Logically, a car is one single body composed of components, and it has no outer extensions separate from the structure. Crash forces should flow uninterrupted from

one end of the passenger to the other. This metric counts disconnected components as a geometry proxy for crash performance. In the case that a layout was separated into two or more components, this disconnection causes the independent deformation and an obvious drop in connectivity. Such an event calls for the welding process to be reinforced, hence we aim to avoid it.

In the initialization section, when creating individuals, it was mentioned that a set of minimal rules was defined for the candidate to be valid. One of them was that every component in creation needs to share at least one flange cell with an existing neighbor. Therefore, the question may arise: Why establish a global connection penalty when the overall connection of a single component is forced in creation? The reason why we introduce this logic lies in the impact that is brought by the transformation functions 4.4. These operators act and tweak the individual only by modifying the chromosome, regardless of what changes happen in the design space. This can cause not only multiple components but also overlapping parts. Overlapping is also prohibited in creation, but can occur because of the same above-mentioned reason. Moreover, it could be mentioned that these two events were linked with each other as the occurrence of one can cause the other. After the individual is transformed, the chromosome re-enters the evaluation pipeline. During the decoding step, if the transformation function has caused one shape to overlap, this shape is skipped in the grid reconstruction because it violates an essential rule in creation. Concurrently, if there is a missing part in the individual, this would cause disconnection in the original single structure and will result in two or more components. Therefore, when this penalty was applied it both covered and monitored cases where parts were overlapping and disconnection of an overall structure.

In order to enforce this requirement, the evaluator turned the set of parts into a graph. Every part is a node, and an edge is drawn whenever the local connectivity check identifies at least one valid flange attachment between two parts. A *Breadth First Search (BFS)* was conducted, which marks all parts reachable through successive joints. If unvisited nodes remain, we deduced that another component had been detected, and the algorithm repeats searching until all nodes have been covered. The count of these found components was then converted into a penalty. Zero means that the structure is solid and forms one continuous component and it increments one unit for every extra fragment found. This metric was constructed in a certain manner that it blends local flange contact into its real-world constraint value.

Let N_{comp} denote the number of connected components identified by the *BFS*. If the structure is continuous $N_{\text{comp}} = 1$ and no cost is applied. Contrarily an each extra fragment adds a linear penalty:

$$p_{\text{GC}} \;=\; \big(N_{\text{comp}} - 1\big), \qquad N_{\text{comp}} \geq 1. \tag{4.4}$$

4.3.3 Shape Count

In a body structure, each additional component brings extra complexity. Therefore, we penalized large part counts in the geometry. Added components lead to more inspection steps, potential sources of variation on the assembly line and essentially an increase in manufacturing costs. Automotive design therefore favors layouts that achieve mobility and crash performance with as few components as possible. Every rib that can be eliminated without harming the main overall targets translates directly into savings.

In order to cover this perspective, the evaluator simply counted how many blocks were successfully placed in the design grid after its completion. A global optimum of 15 parts was set with the aim of reflecting a trivial required count for the design. The observed number of components was then normalized through this global optimum and beyond this threshold, the metric increases linearly. This rule was comparatively soft in comparison to the others and does not forbid larger assemblies when they are necessary to satisfy other metrics. It encouraged the search toward leaner and simpler solutions whenever alternative layouts of equal fitness exist.

Let N_{parts} be the number of components successfully placed in the grid and N_{opt} the preferred upper bound. The shape–count penalty is:

$$p_{\text{SC}} \;=\; \max\bigl(0,\; N_{\text{parts}} - N_{\text{opt}}\bigr), \tag{4.5}$$

4.3.4 Isolation

One of the most basic and essential aspects that needs to be considered in modern body structures is isolation. Vehicle bodies consist of mechanical sections that are corrosion sensitive and need to be hermetically sealed 2.1.1. Along with these spaces, the interior cabin of the car needs to allow for a certain separation from the outer area. Any unintended gap that allows moisture or pressure waves to infiltrate into these protected sections can compromise durability and safety.

To monitor this condition, the evaluator utilized the flood-fill search from the outer boundary and marks every empty reachable cell without crossing components. This process was done three times to cover the entire interior region. Three zones were defined in the left half of the design space, which reflect the side, upper, and lower parts of the cabin, denoted in 4.4.

- **Pass 1** - Starts from the side boundary of the interior and marks every cell it can reach without crossing a part.
- **Pass 2** - Starts from the upper boundary of the cabin and goes through all cells in the direction of the design boundary.
- **Pass 3** - Repeats the same process as the second one for the hood boundary until the lower frame of the design space.

This examination was divided into three sections to obtain a quantitative overall isolation score. Initially, this target was measured by checking the interior perimeter cells and retrieving the fraction of how many were occupied by components. However, this imposed the idea that the interior should be tightly sealed exactly at the perimeter. The second approach consisted of applying one *Breadth First Search* from the start of the design boundary and checking the entire left half of the design space for flow infiltration into the interior cabin. A practical and simple solution, but we lost measurement by getting a categorical answer. If the *BFS* finds at least one way to the interior, it would simply report that there was no isolation, although some sections of the interior were sealed. The current technique aimed to answer the question of how well the perimeter is isolated by also allowing partial isolation at a larger frame than the interior region. The closer the structure comes to the contours, the better the isolation. Concurrently, it allowed the algorithm to understand which movement brings it closer to the target and does not confine it to placing components exactly at the perimeter cells.

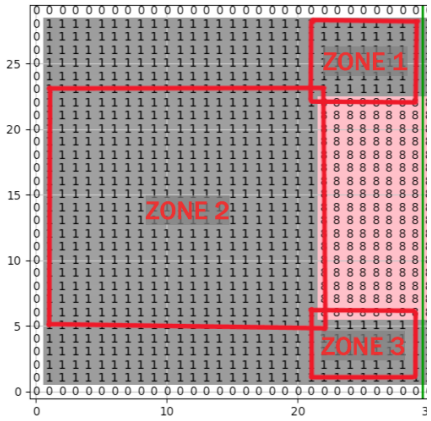


Figure 4.4: Isolation assessment zones used by the three flood-fill passes. Zone 1 spans the outer side corridor; Zones 2 and 3 test the upper and lower gaps around the cabin perimeter. The evaluator measures the fraction of cells in each zone that remain unreachable from the exterior.

As previously mentioned, we proceeded by working with the left half of the design space when aiming for this target. Monitoring the left half up until the mirroring axis was enough for us to reflect the isolation quality of the second half. After all three flood fill processes are finished, the algorithm reports for each zone the fraction of cells that remain unmarked or successfully isolated by the observed structure. These three fractions were then combined into a single isolation score by retrieving the average value. Designs that seal all three sections incur no penalty, whereas any leak in even one zone can reflect a sharp increase in penalty. The entire check runs in linear time because each flood-fill operation touches every grid node at most once.

Let each zone $i \in \{1, 2, 3\}$ contain Z_i grid cells and let L_i be the subset that the flood-fill marks as *leaky* (i.e. reachable from the exterior). The sealed fraction per zone is

$$s_i \;=\; 1 \;-\; \frac{L_i}{Z_i}, \qquad 0 \le s_i \le 1. \tag{4.6}$$

The three fractions are averaged and converted into the isolation penalty:

$$p_{\mathrm{IS}} \;=\; \Big(1 \;-\; \tfrac{1}{3}\sum_{i=1}^{3} s_i\Big), \tag{4.7}$$

so a perfectly sealed layout $(s_1 = s_2 = s_3 = 1)$ yields $o_{\mathrm{IS}} = 0$, while any gap in even a single zone raises the penalty in proportion to the unsealed area.

4.3.5 Distance

Besides stiffness and sealing, a body arrangement must also consider the tightness around the interior. Parts that are positioned too far from the interior frame either force costly metal components or conflict with other assemblies and essentially increase manufacturing complexity and costs. However, the distance penalty is purely positional and no cost model is included.

The overall structure should be compact and concentrated around the interior area within a specific range, which allows for other targets to be reached.

As we mentioned in the previous section, we allowed the isolated area to be larger than the intended interior region and we measure its quality. This current technique aims to support tightness by pushing the components in the center of the design grid. The current objective aims to encourage parts that are positioned closer to the cabin and discourage those that burden the arrangement by being attached further from the interior. However, in the real world, a structure that frees up as much interior as possible is preferred.

To measure compactness, this evaluation step measured for every placed part the *Euclidean Distance* between the part's centroid and the nearest cell on the interior border that surrounds the cabin zone. As long as this clearance remained within a specific range, the band was considered close and incurred no cost. In the case that the centroid drifted further outward, this excess distance was added to a cumulative penalty that grows linearly. The overall penalty is formed by summing this distance over all parts and rises whenever the layout begins to spread away from the cabin. The combination of this technique with the *Shape Count Penalty* 4.3.3 pushes the optimizer toward a more compact and efficient structure.

Formally, let d_k be the Euclidean distance from the centroid of the part k to the nearest cell on the interior border and let d_{band} denote the maximum clearance considered acceptable. The distance objective is

$$p_{\text{DS}} \;=\; \sum_{k=1}^{N_{\text{parts}}} \max\bigl(0,\; d_k - d_{\text{band}}\bigr), \tag{4.8}$$

Parts whose centroids lie within the band ($d_k \le d_{\text{band}}$) add no cost; any excess distance contributes linearly to the penalty.

4.4 Transformation

Once the initial population has been evaluated, the evolutionary cycle continues by applying transformation operators. The pipeline relies on this procedure to generate variation and explore new regions of the design space to gradually refine advantageous solutions. These operators try to introduce randomness into the process and lightly perturb the generated candidates in order to unblock design arrangements which are possibly not accessible simply by raw initialization. In classical genetic programming these operators are crossover and mutation.

- **Crossover** recombines genetic material from two parents to create offspring that inherit features of both.
- **Mutation** injects small random changes that keep the population from converging prematurely.

However, in the context of gene block encoding used in this work, the transformation operates in the following ways:

- A chromosome is modified strictly by its genetic information. The operators utilize only the gene blocks to randomize or tweak the arrangement and are blind to the changes that happen in the design space. Invalid parts that do not comply with the minimal rules in creation are thrown out of the gene. This can cause disconnections or disturbances in

the entire structure. However, after the decoding process, these gaps or irregularities are translated into penalties and handled during evaluation.

- Every part is represented by exactly six genes. Therefore, an operator should never split a block in the middle. This would cause parameters from different parts to be mixed and confuse the process of decoding individuals.

Crossover

Crossover is the main engine for mixing useful sub-structures that are created in different candidates. In our encoding, each part is represented by a six-gene block, so this transformation operator is designed to exchange the whole blocks rather than individual genes. This workflow was designed to preserve the local integrity of every part while allowing sets of components to be combined between individuals. The entire process observed in algorithm 3 implemented the following steps:

1. **Parent Selection** Two parents are drawn from the mating pool with the same selection strategy performed for the GA. Their chromosomes are therefore two lists of length 96, each list partitioned into 16 blocks.
2. **Single Cut-Point** A single cut index is chosen uniformly at random on a block boundary. E.g., between block k and block k+1, where 1¡k¡15. Every gene information standing before the cut is copied from Parent A to *Offspring 1* and from Parent B to *Offspring 2*. All genes after the cut point are exchanged. This technique allows for the offspring to inherit only whole components, and no shape is ever split.
3. **Mirroring Consistency** Because only the left half of the design is encoded, no special action is required for the right half. After the evaluation, the offspring will be mirrored the same way as its parents were.

Algorithm 3 BLOCK CROSSOVER($\mathbf{g}^A, \mathbf{g}^B$)

Require: parent chromosomes $\mathbf{g}^A, \mathbf{g}^B \in \mathbb{Z}^{6B}$, block count B=96
$c \leftarrow \text{randint}(1, B-1)$ ▷ cut index on block boundary
$p \leftarrow 6c$ ▷ gene position of the cut
$\mathbf{o}^1 \leftarrow \mathbf{g}^A[0{:}p] \parallel \mathbf{g}^B[p{:}]$
$\mathbf{o}^2 \leftarrow \mathbf{g}^B[0{:}p] \parallel \mathbf{g}^A[p{:}]$
return $(\mathbf{o}^1, \mathbf{o}^2)$

Mutation

While crossover shuffles existing component blocks, mutation is the mechanism that injects variation into the chromosome. This operator aims to tweak the genes of a candidate to explore the possibility of pushing promising ones to reach their optimum. In each generation, every offspring chromosome is analyzed block by block to retrieve gene material, which will be utilized for modification. After obtaining the necessary information, the operator proceeds to vary the selected component by applying the following changes:

- **Remove** ((33% Probability) The block is discarded, imitating the engineer's decision to drop an unnecessary rib. After the removal, a small check on the count of components is done. In situations where the shape count drops lower than 8, the original chromosome is restored, so the design never degrades to an empty frame.

- **Move** ((33% Probability) The part's anchor coordinates are nudged by 1 cell. This variation tries to click joints or flanges together when a better repositioning might be present, but not found by the original chromosome. A quick boundary check clamps the new position inside the design grid and monitors the anchors in order to avoid sliding into the protected cabin. This tiny step size keeps the change local and allows the GA to fine-tune joint geometry discovered by crossover.
- **Resize** ((34% Probability) This variation extends or cuts the flange lengths of an L-rib or the single length of the straight component. The maximum variation lies between [-2:2] in order to offer small steps of exploration and avoid early convergence by applying large changes. Lengths are clipped to the catalog limits, meaning that if the maximum or minimum length of a flange its reached the operator can not extend any further.

After these steps were applied, the genome blocks were flattened back into the individual and are ready for re-evaluation. This three-step strategy, delineated in algorithm 4, reflects the current procedure of body construction practice by considering these three perspectives:

- Deleting the rib reduces cost.
- Sliding a flange by a few millimeters helps it meet a mating component.
- Stretching a leg adjusts crash absorption and minimizes additional part attachments.

Every mutation operation touches at most one block. Therefore, feasibility checks are lightweight, however, the cumulative effect over many generations granted the optimizer the freedom of exploring optimal arrangements that the crossover alone or initialization could not reach.

Algorithm 4 MUTATE BLOCK($\mathbf{g}$)

Require: chromosome $\mathbf{g} \in \mathbb{Z}^{6B}$, active-block set $\mathcal{A}$, minimum part count $N_{\min}$, probabilities $P_{\text{rm}}, P_{\text{mv}}, P_{\text{rs}}$
if $\mathcal{A} = \varnothing$ **then return** $\mathbf{g}$ ▷ no active parts
end if
$k \leftarrow$ RANDOM CHOICE($\mathcal{A}$) ▷ block index
$(s, \ell_1, \ell_2, x, y, r) \leftarrow$ block k of $\mathbf{g}$
draw $u \sim \mathcal{U}(0, 1)$

if $u < P_{\text{rm}}$ **then** ▷ **Remove**
  if $|\mathcal{A}| > N_{\min}$ **then**
    set block k to $[-1, \ldots, -1]$ ▷ pad with sentinel
  end if
else if $u < P_{\text{rm}} + P_{\text{mv}}$ **then** ▷ **Move**
  $(x', y') \leftarrow (x + \text{randint}(-1, 1),\ y + \text{randint}(-1, 1))$
  $(x, y) \leftarrow$ CLAMP TO GRID(x', y')
else if $u < 1$ **then** ▷ **Resize**
  $\delta \leftarrow \text{randint}(-2, 2)$
  $(\ell_1, \ell_2) \leftarrow$ CLAMP TO LIMITS$(\ell_1 + \delta, \ell_2 + \delta)$
end if
write $(s, \ell_1, \ell_2, x, y, r)$ back into block k
return $\mathbf{g}$

4.5 Implementation

The preceding sections introduced the ingredients of the optimization framework from chromosome encoding, population initialization, evaluation, to the genetic operators. In this implementation section, we will proceed by explaining how these components are integrated into one concrete data flow that is executed at every generation. The evolutionary engine is implemented with the DEAP Framework [37]. This library supplies the population container,

selection utilities, and a "toolbox" for registering genetic operators. Every step of the following pipeline is executed through a DEAP mechanism that has been customized to respect the frame of the gene block encoding and the design constraints discussed earlier. The data flow is visualized in Figure 4.5 and is described in the steps below:

- **Population Creation** A fixed-size population of chromosomes was initialized based on the evaluation budget through the first step of the creation mentioned earlier. Each individual was constructed and quality checked for irregularities before being encoded and prepared for further evaluation.
- **Evaluation** (Initial) After the batch of chromosomes was generated, every individual was immediately decoded, mirrored, positioned, and undergoes the evaluation process. Every candidate gets assigned a full vector of objective scores and the derived scalar fitness to each genome before any selective pressure is applied.
- **Selection** Using those fitness values, the algorithm uses *Tournament Selection* to rank the best individuals of the population. This strategy implies individuals competing with each other, and based on the initial evaluation score, the fittest advance to the mating pool. This process repeats until enough parents have been chosen to create the offspring of the next generation. In parallel, a copy of the current best individual is carried forward unchanged. This preserves elitism and ensures that the global progress is never lost.
- **Transformation** The selected candidates are processed by the transformation functions. Crossover was utilized to exchange gene blocks between two parents and provide the offspring population that will be inherited by the next generation. Mutation applies geometric variations to the proper chromosome blocks. The operators together supply a mix of exploitation and exploration required for effective search.
- **Variant Hook** Immediately after the transformation step, the offspring enter a lightweight pre-processing step, which depends on the optimization variant that is selected. These three approaches are explained in the next section 4.5.1.
- **Evaluation** (Offspring) Each newborn chromosome is decoded, mirrored, and placed on a fresh design grid. Every new candidate re-enters the evaluation phase and gets assigned a fitness score by the 5 objective functions. Parents and offspring now have comparable scores.
- **Population Update** Parents and offspring are merged, which implies the best individuals, ranked by scalar fitness, are retained until the population size is restored. This combines diversity and the reliability of strong candidates.
- **Termination** After every generation, the current incumbent (i.e. the best-so-far chromosome with the lowest fitness/penalty vector at this moment of the generation) is stored in the run log.. The generative loop in this pipeline starts from the third step, which is selection, and ends at the population update. Hence, evaluation precedes the first selection and then follows every transformation step. A fixed evaluation budget is decided before running the optimization, which is denoted by population size x number of generations. These two parameters define the duration of this entire circuit, which is repeated until the evaluation budget is exhausted.

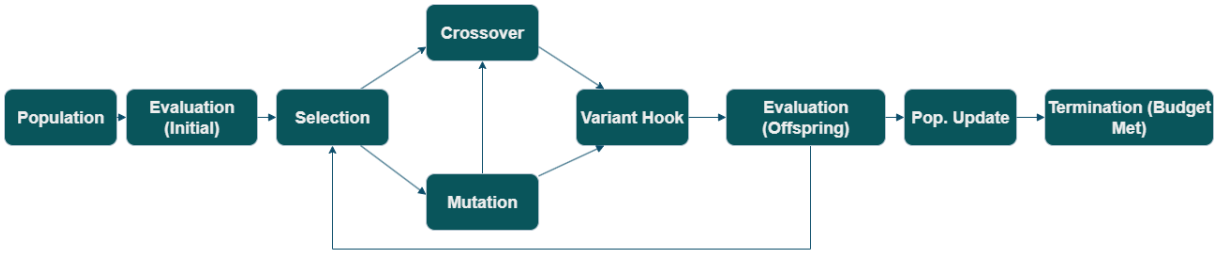


Figure 4.5: Generation-level data flow of the GA implementation.

4.5.1 Variants

In the previous section, we described the entire workflow of our optimization. As mentioned in the pipeline, the 5th step of the process was the variant hook. After selection and genetic transformation, the offspring pass through a small step that customizes the core GA without altering its structure. Every other step of the pipeline remains the same in terms of its role in the optimization. However, this small stage described in algorithm 5 simply defines the way the following procedures will function.

Algorithm 5 VARIANT HOOK$(\mathcal{P}, \boldsymbol{\omega}, g)$

Require: offspring set $\mathcal{P}$, weight vector $\boldsymbol{\omega}$, current generation g, chosen *variant* $\in$ {BASELINE, ADAPTIVE, REPAIR} variant
BASELINE
return $(\mathcal{P},\ \boldsymbol{\omega})$
ADAPTIVE ▷ dynamic weight schedule
$\boldsymbol{\omega} \leftarrow$ ADAPT WEIGHTS$(\boldsymbol{\omega}, g)$
return $(\mathcal{P},\ \boldsymbol{\omega})$
REPAIR ▷ local shift–repair loop
for all $p \in \mathcal{P}$ **do**
    $p \leftarrow$ REPAIR SHIFT(p)
end for
return $(\mathcal{P},\ \boldsymbol{\omega})$

Baseline

The baseline variant is a "vanilla" setup which does not introduce any changes to the workflow. It is a pure GA optimization with static settings. Penalty weights, operator probabilities and evaluation logic remain fixed throughout the entire run. This variant therefore represents a conventional steady-state algorithm.

$$f_{\mathrm{B}}(p) \;=\; \sum_{i=1}^{5} \omega_i\, o_i(p), \tag{4.9}$$

Where $o_i(p)$ are the five objective terms and the weights ω_i remain constant for the entire run.

Adaptive

In contrast to how the simple genetic optimization workflow functions, this approach introduces dynamic variations of the priorities during the optimization. In the baseline variant, we mentioned that the penalty weights remain fixed. Each fitness constructor has a given weight at the beginning of the GA run, which defines the priorities for every penalty and it is carried throughout the entire optimization. On the contrary, adaptive weights allow for the penalty weights to

change during the optimization. This adaptive variant achieves this by rescheduling the objective weights over time. At the beginning of each generation, the hook computes new weight values as a smooth function of the generation counter, which can be a linear ramp or a logistic curve and passes them to the evaluator. This was one way of exploiting this approach, with the generation count defining how the weights will fluctuate during the optimization. However, this priority shift can also be done based on the fitness values for a penalty, e.g, when one penalty value reaches a certain satisfactory value, the weight on this penalty gets lowered and vice versa. Hence, the adaptive variant can be utilized in many ways to redefine the importance arrangement based on the specific goal of the optimization.

$$f_{\mathsf{A}}\big(p, g\big) \;=\; \sum_{i=1}^{5} \omega_i(g)\, o_i(p), \qquad \omega_i : \{0, \ldots, G_{\max}\} \longrightarrow \mathbb{R}_{\geq 0}, \tag{4.10}$$

where g is the current generation index, $o_i(p)$ are the five objective terms, and each weight $\omega_i(g)$ is updated by a time-dependent schedule (e.g. linear ramp, logistic curve, or feedback rule) before the evaluation step begins.

Repair

In the transformation step, operators occasionally can leave a part detached or slightly out of place. As previously mentioned in the respective section 4.4, genetic operators work strictly through the chromosome information without taking into account changes in the design space. This can lead to offspring that violate connectivity rules and other anomalies in the overall structure. The repair variant adds a local shift search before evaluation, which tries small positional tweaks to fix this issue. This approach was based on the grid space positioning and reasons on the design space arrangement to fix gaps that transformation operators can not cover. This increases computational complexity because to make this work, several penalty constructor methods are pre-applied before entering evaluation. By iterating over any part that lies in a separate component and trying a small set of one-cell moves, the repair operator checks if a move reduces the total number of components or stabilizes any other irregularity in the offspring. In case these operations do not improve the candidate, the original position was kept. The search is capped at a few iterations to keep the overhead negligible. After this repair phase, the offspring enters evaluation with the standard penalty weights as defined in the baseline variant.

$$f_{\mathsf{R}}(p) \;=\; \sum_{i=1}^{5} \omega_i\, o_i\big(\,\mathcal{R}(p)\big), \tag{4.11}$$

where $\mathcal{R}(p)$ denotes the layout obtained after the local shift–repair heuristic was applied to the offspring chromosome, and ω_i are the same fixed weights used in the baseline evaluation.

Chapter 5

Experiments

The previous chapter specified the *GA* framework and its three configurable variants *(Baseline, Adaptive, Repair)*. In the present chapter, we will formalize the experimental setup and describe how these variants are applied in a controlled study. Firstly, the motivation of the underlying design trade-off that led to the creation of these methods is explained.

5.1 Design Trade–off

In this optimization, we introduced 5 penalties that guide the optimizer towards optimal designs. Early pilot runs of the *GA* showed that two of them are the most influential: **Global Connectivity** and **Isolation**.

It was observed that these two penalties pull the optimizer in opposite directions. Prioritizing isolation by giving it a higher weight on the fitness formation encourages the algorithm to isolate the interior region successfully. Although this isolation convergence is reached through the transformation functions. It was seen that relying solely on creation before transformation functions come into play would converge the isolation only up to a certain point, leaving one side of the interior traversable. Concurrently, transformation functions add a certain variation to the initial designs that pushes the isolation penalty forward to full convergence. However, as we mentioned in the transformation section 4.4, these operators act only on a chromosome level and are blind to changes in the design space. When transformation operators come into play, they propose solutions that fully isolate the interior, but in most cases, they end up combining blocks of structures that cause overlapping parts. This leads to the decoder skipping them and results in loose parts that fragment the design. Although this is not allowed theoretically, the selection strategy allows it because of the large weight set to the isolation penalty, which dominates the fitness composition.

In the opposite case, a high weight on global connectivity penalizes any fragmentation of the structure. Therefore, when transformations suggest isolated structures, they are most likely filtered out of optimization because the global connectivity weight does not tolerate disconnections. Only the rare offspring that happen to isolate the cabin and avoid overlapping or skipping parts can survive the selection.

This trade-off characterizes the central obstacle of this optimization. The three introduced variants were designed to navigate this trade-off.

5.2 Variant Purpose

5.2.1 Baseline

The baseline optimizer is the single approach that serves as ground control. It implies a steady-state genetic algorithm that keeps every hyperparameter and penalty weight fixed from the start of the optimization to the end. This variant aims to answer the following two questions:

- **Reference Performance** How far can a conventional *GA* balance the **isolation** vs **global connectivity** trade-off when the objective landscape is completely static?
- **Calibration Point** Which scale for each penalty term is reasonable before we introduce further specialized variants?

Operating with constant weights means the optimizer cannot "escape" the diagonal frontier mentioned in section 5.1. Instead, it must discover a compromise layout that simultaneously satisfies both penalties.

5.2.2 Adaptive Weights

The **Adaptive** variant keeps the *GA* kernel identical to the baseline, but it differentiates by replacing the static weight vector with a one-dimensional schedule

$$\boldsymbol{\omega}(g) \;=\; \big(\omega_{\mathrm{LC}},\, \omega_{\mathrm{GC}}(g),\, \omega_{\mathrm{SC}},\, \omega_{\mathrm{IS}}(g),\, \omega_{\mathrm{DS}}\big), \qquad g = 0,\, \ldots,\, G-1.$$

where the **global connectivity** ω_{GC} weight and the isolation weight ω_{IS} evolve smoothly with the generation counter. This schedule is learned together with the usual *GA* parameters and can take the form of a linear ramp. logistic curve, or any differentiable function implemented in the wrapper. The rationale was to exploit the temporal leverage of a *GA* run:

- **Early Generations** Place a higher priority on **isolation** so the optimizer explores structures that can seal the interior in the early phase, even if they have disconnected components.
- **Late Generations** Gradually shift priority toward **global connectivity**, guiding the search to reconnect detached parts or discard designs that remain fragmented.

By allowing the objective frame to shift, this variant seeks solutions that lie beyond the static trade-off frontier.

5.2.3 Repair Mechanism

The last approach is the **Repair** mechanism, which keeps the same static weight vector and genetic operator probabilities as the *Baseline*, but inserts a lightweight *local shift* operator immediately after the transformation functions and before evaluation. This variant scans the left half of the design grid for any disconnected component and tries several *one-cell* moves. The first move that lowers the component count or at least improves at least one target without worsening the rest is accepted. This procedure was capped at a small number of iterations and aims to lower the **global connectivity** penalty. Because the shift is local and takes into consideration the topology of the design, it tries to restore connectivity with minimal geometric change. This variation aimed to retain the isolation quality that might have been improved by

genetic operators and additionally improve overall connection. The strong feature of this variant is being conscious of the changes that happen in the design space. However, this increases complexity during the optimization.

5.3 Experimental Setup

All three variants operate under one strictly uniform protocol and only their internal logic differs 4.5.1. This section describes every element that remains identical across variants during the experimentation phase. This is done with the aim of later comparisons being unambiguous and reproducible.

5.3.1 Protocol

Parameter Values

The chromosome encoding, mirroring step, evaluation pipeline and $k{=}3$ tournament selection strategy are *exactly* those described in chapter 4. None of the variants alters these core mechanics. The rest of the *GA* parameter values, such as transformation probabilities and weights, were established based on *hyperparameter optimization*. The single best configuration returned by the first study 5.4.1 was retrieved and replayed across 20 runs for these two variants. Similarly, the best incumbent retrieved by the second study 5.4.2 was utilized to benchmark it across 20 separate runs with a lower budget.

Multi-seed

In order to control randomness in the algorithm, deterministic seeds were implemented. Each incumbent was re-evaluated on the fixed seed list. Using the *same* seeds for every variant ensures that paired statistical tests are valid.

Logging

Every run was recorded by *IOHProfiler* [38]. This heuristic tool reports for each generation the fitness values in every evaluation. These logs were later processed through *IOHAnalyzer* [39] to produce convergence curves, plots and statistical summaries. These elements are described and compared along variants in chapter 6.

5.4 Hyper-parameter Optimization

Hyperparameter tuning for all three variants was conducted in two independent studies. In order to execute this optimization ,the *SMAC* library [40] was used.

5.4.1 Baseline & Adaptive

The first study is a joint optimization that allows the optimizer to explore both *Baseline* and *Adaptive* variants simultaneously. Table 5.1 presents the parameters exposed for the optimization.

As observed in the table, the variant parameter was set as a categorical variable. This joint search gives the optimization platform the freedom to explore between the two variants and establish based on the cost which one performs best.

The optimization budget was 40 trials in total. Each trial of the genetic algorithm was initialized with $N_{\text{pop}} = 500$ individuals and $G = 50$ generations which yields a **fixed budget** of $25{,}000 = 500 \times 50$ fitness evaluations. The run was performed with a fixed *SMAC* seed, and reports the best scalar fitness at generation 50 as the cost. The final best incumbent configuration was extracted from this single run history.

Table 5.1: Hyper-parameters and their search ranges for *SMAC*. The categorical variable `variant` chooses the GA algorithm to be tuned (*baseline* or *adaptive*). For each suggested configuration, SMAC samples continuous crossover and mutation probabilities $\texttt{cxpb}, \texttt{mutpb} \in [0.10, 0.90]$ and integer penalty weights for global connectivity and interior isolation in the inclusive range $[100, 3000]$. All other GA settings (population, evaluation budget, repair heuristic) remain fixed.

Parameter	**Search domain**
`variant`	{baseline, adaptive} (categorical)
– Crossover prob. (`cxpb`)	$[0.10,\ 0.90]$ (continuous)
– Mutation prob. (`mutpb`)	$[0.10,\ 0.90]$ (continuous)
– Global-connectivity weight (`conn_weight`)	$[100,\ 3000]$ (integer)
– Isolation weight (`iso_weight`)	$[100,\ 3000]$ (integer)

5.4.2 Repair

The second study was conducted only for the third *Repair* variant with the exposed parameters in table 5.2. The *Repair* evaluator performs additional collision checks and partial penalty pre-computations. This operator increases the wall-clock time and the fitness call is higher. To keep total optimization effort reasonable, **20 trials** were allotted and searched only within that variant.

A constant overall evaluation budget of $\mathbf{4000}$ fitness calls *per trial* was enforced:

$$N_{\text{pop}} \times n_{\text{gen}} = 4000.$$

SMAC was therefore allowed to choose `pop_size` *and* derive $n_{\text{gen}} = 4000/\texttt{pop\_size}$ during the run (rounded to the nearest integer). This allowed the optimizer to balance *exploration* and *exploitation* as well.

5.5 Generalization

To assess the robustness of the framework beyond the default setup, two small variations were introduced and observed:

- **Additional Parts** In order to observe how the optimizer behaves with slightly more complex geometries, new parts were included in the creation of individuals. In the catalog 4.2, the three shapes *Angular left, U-shape, Angular right* were used to generalize the performance of the program after the *HPO* optimization. This variation was created to observe how well the optimizer can utilize the new parts introduced.
- **Interior Region** The second test consisted of a geometrical change of the interior area in the design space. Different from the original interior shape (*rectangular*), the new

Table 5.2: Hyper-parameters and their search ranges for the *Repair-only SMAC* study. The five parameters match Table 5.1, but here the population size `pop_size` is included as an integer variable in $[100, 400]$.

Parameter	**Search domain**
`pop_size` (integer)[†]	$[100,\ 400]$
– Crossover prob. (`cxpb`)	$[0.10,\ 0.90]$ (continuous)
– Mutation prob. (`mutpb`)	$[0.10,\ 0.90]$ (continuous)
– Global-connectivity weight (`conn_weight`)	$[100,\ 3000]$ (integer)
– Isolation weight (`iso_weight`)	$[100,\ 3000]$ (integer)

[†]The number of generations per trial is computed as $n_{\text{gen}} = 4000/\texttt{pop\_size}$ (rounded), ensuring a constant evaluation budget of 4000 per trial.

interior consists of two external boxes on the top and bottom of the original rectangular cabin. This also aimed for a minimal reflection of the vehicle shape seen from above. Through this variation, we aimed to observe how well the optimizer can adapt to the new interior shape and evaluate isolation.

Chapter 6

Results

This chapter evaluates the three genetic-algorithm variants under three complementary sections. Section 6.1 describes the *hyperparameter* optimization of the three variants. Concurrently, 6.2 quantify how well the variant configurations mentioned in Chapter 5) minimize the scalar fitness on the 2D design domain. All results reported here stem from the 2-D demonstrator. Numbers should be interpreted as qualitative indicators for a future 3-D study. It reports convergence trajectories, distribution statistics, and qualitative design snapshots. Section 6.3 shows the **robustness** of those tuned settings when the problem is altered. Together, the analyses expose both the strengths and limitations.

6.1 Hyper-Parameter Optimization

6.1.1 Baseline & Adaptive *HPO*

Incumbent configuration

SMAC settled on returning a **exploratory** setting. Both the crossover rate *(cxpb* $= 0.77$*)* and especially the mutation rate *mutpb* $= 0.85$ lie close to the upper bound of their respective search spaces. The scalar fitness formulation rewards more heavily global connectivity. The optimizer returned a weight value 3 times larger for the first penalty function. Out of these variants, the best selected by *SMAC* is the baseline one as shown in the table 6.1.

Table 6.1: Incumbent returned by *SMAC* from the **Baseline & Adaptive** study

Parameter	**Value**
Variant (`variant`)	`baseline`
Crossover probability (`cxpb`)	0.774
Mutation probability (`mutpb`)	0.846
Global–connectivity weight (`conn_weight`)	2155
Isolation weight (`iso_weight`)	800

Convergence

The trajectory in figure 6.1 reveals a **fast** descent of the fitness. Within the very first trials, the incumbent cost falls from ≈ 690 to ~ 540. A final improvement drives the fitness down to

500, after which the curve remains stable for the rest of the evaluations.

This pattern indicates that *SMAC* locates a near-optimal configuration early in the optimization. Afterwards, lacking evidence for any better region, the optimizer switches to local exploitation around that incumbent. This result suggests that the search space is benign enough for the optimizer to converge quickly. Additional trials beyond the first dozen provide diminishing returns, resulting in the final incumbent reported in 6.1.

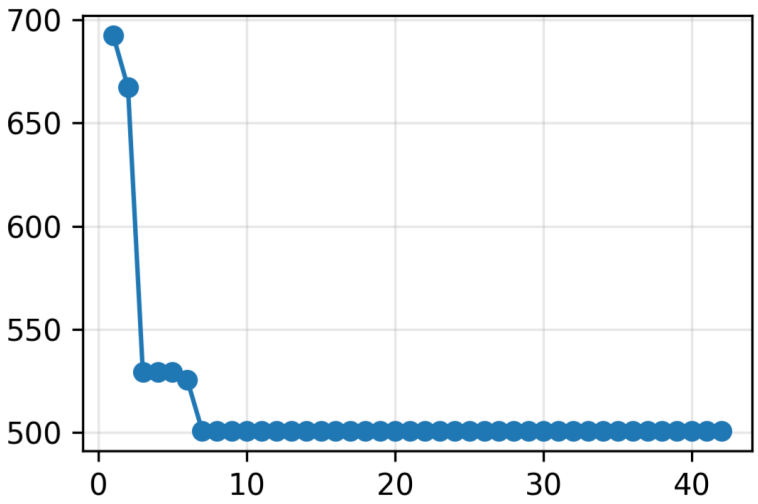
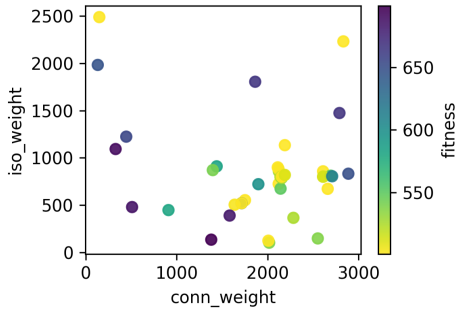


Figure 6.1: Incumbent cost vs. *SMAC* trial (**Baseline & Adaptive** Study)

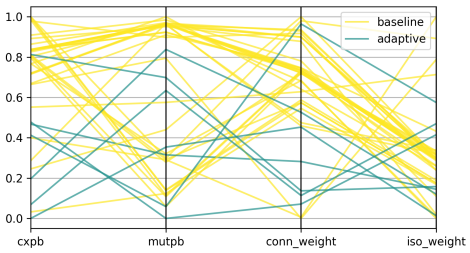
Penalty Trade-Off

In Figure 6.2, the relation between several parameters during the optimization is depicted. Panel 6.2a shows that the optimizer quickly discovers a sweet spot for the two penalty weights around $conn\_weight \approx 1800$–2300 and $iso\_weight \approx 600$–1000. Trials outside this rectangle are systematically darker, confirming that *under* and *over-penalizing* lead to worse overall fitness. Panel 6.2b places the same top 40 trials in a parallel coordinate layout. This visualization brings to light two observations:

- The four axes reveal distinct behavior. While *cxpb* and *mutpb* span most of their ranges, the penalty weights collapse to the tight segments already seen in panel 6.2a.
- The *adaptive* traces tend to sit higher on the mutation axis. This observation hints that the *adaptive* variant profits from a slightly larger mutation value. Meanwhile, the *baseline* variant compensates with stronger connectivity weights.



(a) Trade-off between penalty weights.



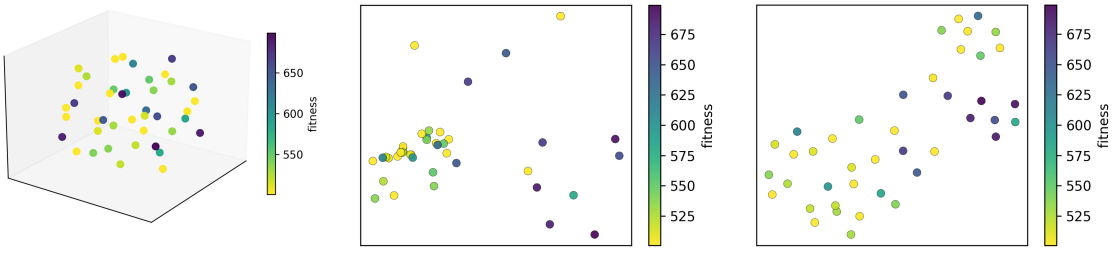
(b) Parallel–coordinates of the best 40 trials.

Figure 6.2: Exploration of the hyper-parameter space. In (a) each point is a trial and the color encodes the achieved fitness. In (b) lines are colored by the GA variant chosen in that trial.

Dimensionality Reduction

Figure 6.3 depicts the optimization results in a lower-dimensional space. Three techniques are used to represent the data in a basic visualization while preserving its substance.

- **t-SNE** a In the panel, a non-linear t-SNE mapping of the optimizer's search trajectory is observed. In this *3D* latent space, two observations emerge:
  - The best trials (yellow points) are concentrated towards the center of the *3D* plot. This indicates that *SMAC* converged to a single mode.
  - Concurrently, darker outliers surround that island. This shows that *SMAC* sampled and discarded incumbents of a substantially higher cost.
- **PCA** b The second panel projects the *z-standardized* parameters onto the first two principal components. This plot confirms that the optimal region occupies only a fragment of the parameter space.
- **UMAP** c The third panel complements the dimensionality reduction with a UMAP mapping that balances local and global structure. The embedding reproduces a similar dense cluster, but also reveals a line of average solutions (green/blue) that **t-SNE** had compressed.



(a) t-SNE (3-D) (b) PCA (2-D) (c) UMAP (2-D)

Figure 6.3: Complementary low-dimensional views of the 40 *SMAC* trials. Color encodes fitness.

6.1.2 Repair *HPO*

Incumbent configuration

Table 6.2 lists the best setting returned by the *repair-only SMAC* study. Compared with the *Baseline & Adaptive* optimization, the mutation probability remains at the upper end of the search space. Meanwhile, crossover is noticeably lower. Different from the first study, the selected penalty weights are reversed in priority. The *isolation* weight in this study is higher than the *global connectivity*. This can be reasoned by the need to compensate a smaller evaluation budget of 4000 evaluations.

Table 6.2: Incumbent returned by *SMAC* for the **repair study**.

Parameter	**Value**
Crossover probability (`cxpb`)	0.374
Mutation probability (`mutpb`)	0.793
Global–connectivity weight (`conn_weight`)	588
Isolation weight (`iso_weight`)	1440
Population size (`pop`)	263
Number of generations (`ngen`)	15

Convergence

Figure 6.4 shows the incumbent trajectory. Similarly to the first study, *SMAC* again discovers a steep fitness drop in the very first evaluations. The cost degrades from ≈ 700 to the ≈ 510 level by trial 4 and then stagnates. The early drop indicates that even with fewer samples, the optimizer is able to converge on a single promising region. Subsequently, spending the remaining trials refining variations around it.

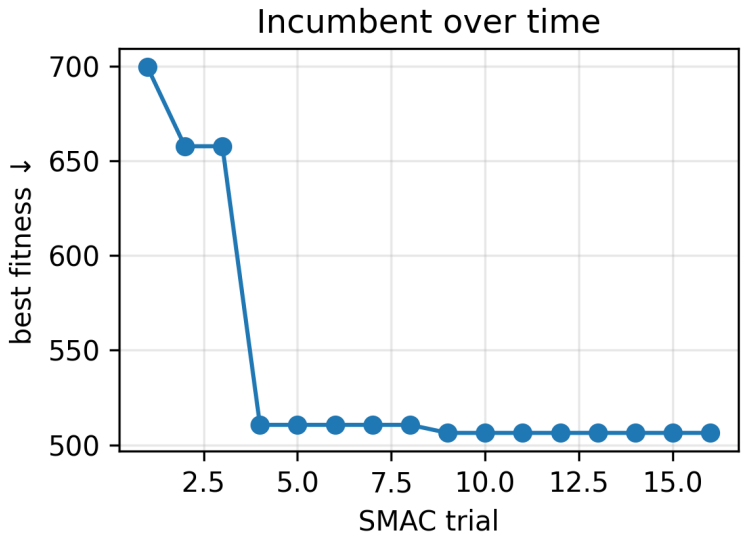
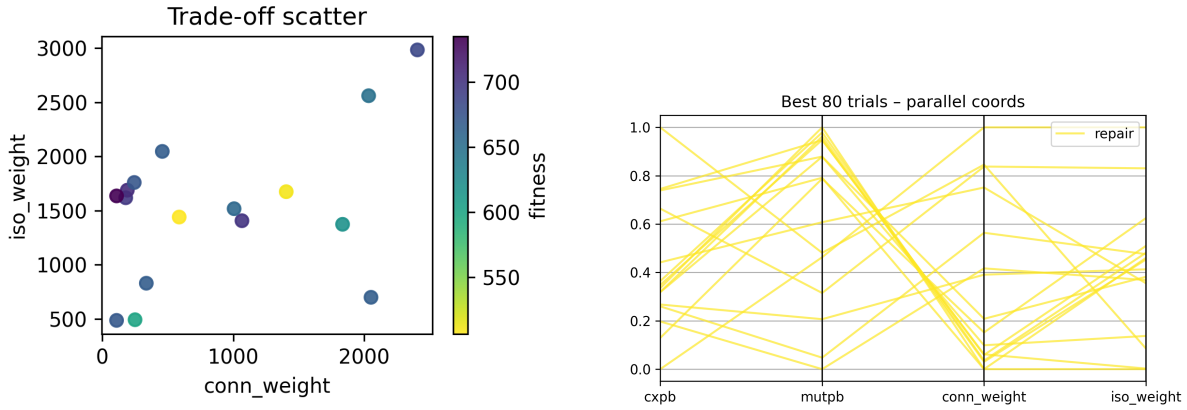


Figure 6.4: Incumbent cost vs. *SMAC* trial (repair study)

Penalty trade-off

Figure 6.5 illustrates the two views used for the weight balance applied to the 20 *repair* trials. In the scatter plot (a) the best solutions cluster in one diagonal strip *conn_weight*≈ 500–1800 and *iso_weight* $\approx$ 800–1700. Points outside this band (upper left or lower right corners) are consistently darker. This assures that an *imbalanced* emphasis on only one penalty is unfavorable. In the parallel-coordinate view (b), the observation that the most successful trials adopt mutation rates above 0.7 is retrieved. Meanwhile, crossover spreads from 0.2 to 0.8. This fact suggests that the *repair* variant relies more heavily on mutation to explore the search space.



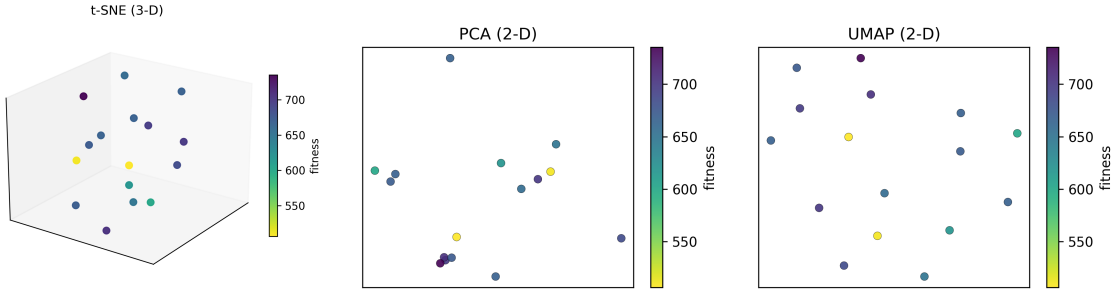
(a) Trade-off between penalty weights.

(b) Parallel–coordinates of repair trials.

Figure 6.5: Exploration of the hyper-parameter space for the *repair* variant. Colour encodes fitness.

Dimensionality reduction

Figure 6.6 summarizes the 20 *repair* trials in three reduced spaces:



(a) t-SNE (3-D)

(b) PCA (2-D)

(c) UMAP (2-D)

Figure 6.6: Low-dimensional embeddings of the *repair* trials (colour = fitness).

- **t-SNE** (a) separates the trial set into sorts. The lighter points lay in the center part. This represents a single optimal space of the incumbents.
- **PCA** (b) shows a similar split along the first component, which again is governed mainly by the two penalty weights.

- **UMAP** (c) preserves both local neighborhoods and global distances. This embedding confirms that the top-performing configurations sit tightly together.

6.2 Benchmark Runs

After selecting the two best performing *hyperparameter* configurations for each variant. A total of 20 independent *GA* runs per variant were executed. These runs were done to obtain statistically sound performance curves with a budget capped at 4000 evaluations. The reason behind this limit is that the *repair* variant has a lower budget. Although the *baseline* and *adaptive* variants were tuned at a higher budget, each variant will be compared fairly at the same expense. Each run starts from a different random seed and uses the best incumbent parameters reported in section 6.1.1. All runs were logged with *IOHProfiler* and analyzed with *IOHAnalyzer*. The following subsections present these summaries and compare the three variants.

6.2.1 Fitness Trajectory

The following figure illustrates how quickly each variant lowers the *best-so-far* fitness throughout 20 runs. The thick colored line is the mean trajectory, while the shaded ribbon captures the interquartile ranges. The *x axis* is scaled by log_{10} in every plot.

Baseline Variant - In Figure 6.7, the **(olive dashed)** line tracks the mean *best-so-far* fitness of the 20 *baseline* runs. During the first evaluations, the average curve drops quickly from the initial $f_{\text{mean}} \approx 2900$ to below $f = 1000$. It eliminates almost 60% of the objectives within the first evaluations. This demonstrates that the plain GA can exploit a good portion of the search space very quickly. After this early dive, the slope flattens. Beyond roughly 2000 evaluations, the curve plateaus at $f_{\text{mean}} \approx 280$. A long declining tail brings the curve down to its final fitness value of ≈ 273. The absence of sharp drops after the early phase suggests that the remaining search spaces contain only marginally better configurations. Exploration continues but has minimal returns. Overall, the *baseline* variant delivers fast initial progress and a reliable late-stage refinement.

Adaptive Variant - The **(orange solid)** line in Figure 6.7 displays the mean *best-so-far* fitness for the *adaptive* runs. The mean *best so far* fitness starts around 2000 and descends steadily as the evaluations proceed. In comparison with the *baseline* variant, the *adaptive* scheduler shows more gradual improvement in the early stage. By the first evaluations, the mean falls to $f_{mean} \approx 1100$, which is roughly halfway to its initial value. Between 10^2 and 10^3 evaluations, the descent accelerates again, reaching ≈ 700 and finally leveling off just around 600. The interquantile band reveals a wider range throughout the first evaluations, which reflects additional stochasticity by the *adaptive* weights. To summarize, the *adaptive GA* displays a smoother but longer learning curve, although it needs more evaluations to match the *baseline*'s late-stage performance.

Repair Variant - The **(green dotted)** line represents the mean fitness trajectory of the 20 *repair* runs. The *best so far* fitness starts at a value of roughly 2000 and follows two stages of improvement. Initially, the fitness decreased steadily but at a gentler slope than the *baseline* curve. By the first evaluations, it reaches $f_{mean} \approx 1300$ and not until the following evaluations

does it break the 1000 barrier. The wide interquantile band in this region highlights a larger spread, which reveals that some executions profit markedly from early repairs, although not consistently. At a second stage, the decline accelerates and pushes the mean down to ≈ 650 by 2000 evaluations. Beyond that point, the curve flattens, which suggests that the remaining budget is spent on local refinements. Overall, the *repair* variant eventually attains a fitness level comparable to the *adaptive GA*. However, it does so more slowly and with a greater variance.

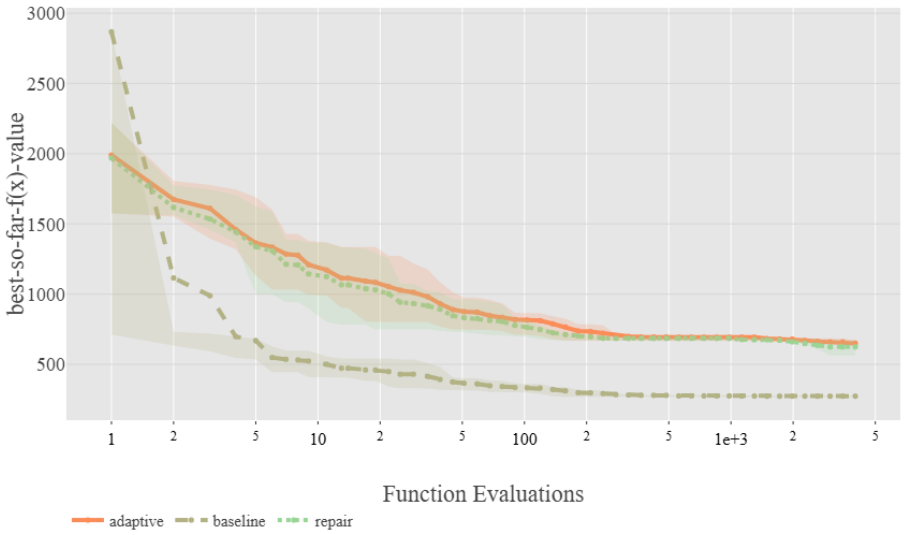


Figure 6.7: Mean convergence curves for the three GA variants. Shaded regions denote the 25–75 % percentile range. (20 runs, $4\,000$ evaluations each).

Table 6.3 summarizes the distribution of the best fitness obtained at the end of each benchmark run. Looking at these statistics, the following observations are noticed:

- **Baseline** exhibits a remarkably tight spread. The mean and the median are virtually identical, and the standard deviation is ≈ 8. Every recorded percentile between the 2^{nd} and the 98^{th} lies in a narrow band. This confirms the visualization of the convergence curves 6.7. Once the variant is tuned, the optimizer reliably steers most runs to the same high-quality individuals.
- **Adaptive** shows a different pattern. Its center of mass is higher, and the distribution is wide $\sigma \approx 56$. The 2^{nd} and 10^{th} percentiles dip below 560, indicating that the variant does find competitive solutions. However, most of the runs stagnate around $660 - 700$. Together with the ribbon visualized in the convergence plot 6.7, the table emphasizes that the variant's performance is run-dependent.
- **Repair** stands in an intermediate position. The observed median is comparable with the *adaptive* variant, but the lower tail is substantially better. The large standard deviation ($\sigma \approx 76$) and the broad upper percentiles indicate that this variant either converges well or gets stuck away from the optimum. While computationally costly, this variant offers the potential for strong solutions.

Table 6.3: Distribution of final best-found fitness over 20 runs per variant (lower is better). Columns list the arithmetic mean, median, sample standard deviation, and the empirical percentiles 2 %, 5 %, 10 %, 25 %, 50 %, 75 %, 90 %, 95 % and 98 %. The baseline achieves the lowest central tendency (*mean = 273*), while the repair variant shows a wider dispersion (*SD = 76*) but still improves the worst-case tails compared with the adaptive schedule.

Variant	**Mean**	**Median**	**SD**	2%	5%	10%	25%	50%	75%	90%	95%	98%
Baseline	273	268	7.5	268	268	268	268	268	277	282	285	290
Adaptive	650	668	55.8	512	528	561	666	668	672	692	698	703
Repair	624	652	76.2	501	501	510	563	652	672	688	708	736

6.2.2 Empirical Cumulative Distribution

Figure 6.8 summarizes the benchmark into a single "cumulative target" view. For every target fitness shown on the *x axis*, the ordinate reports the fraction of runs that achieved a lower value than that threshold.

- **Baseline (olive dashed)** - The curve remains at 1.0 until it reaches ≈ 620, then drops. Every *baseline* run eventually attains 620, however, only a handful manage to push below ≈ 480. The very steep descent implies low variability once this threshold is crossed.
- **Adaptive (orange solid)** - The line begins to fall a decade earlier than the *baseline* variant. It reaches the 50 % mark at ≈ 560, and tails off near 480. The broader "stairs" section indicates a wider dispersion of fitness compared with the *baseline* variant.
- **Repair (green dotted)** - For this variant, first declines appear above 700 and the curve drops gradually. It does not reach zero until ≈ 580, which reflects that the variant's slower progress. This is seen as well in the convergence plots 6.7.

The plot confirms the ranking, which was already visible in the convergence trajectories. The *baseline* variant leads as the best approach, and then is followed by the *adaptive* and *repair* variants.

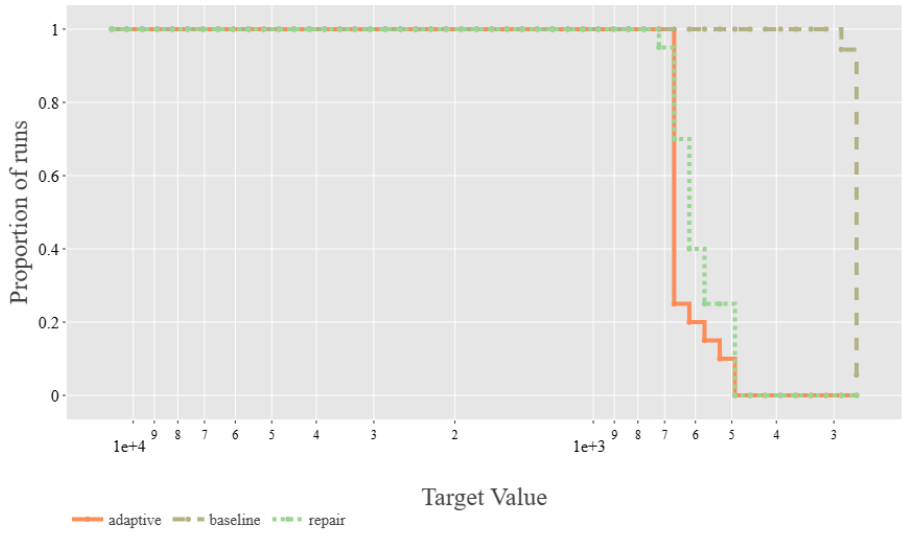


Figure 6.8: Empirical cumulative distribution of best-of-run fitness for the three GA variants. Curves farther left indicate a higher probability of reaching lower (better) fitness.

6.2.3 Probability Of Success

The following figures 6.9 compare the spread of the fitness reached at the end of every independent run. Each violin merges a density estimate with an internal *box and whisker* summary.

- **Baseline** - For the *baseline* variant the violin is compact and the interquartile range is barely 5 points. The full span covers 25 points $(268 - 294)$. This assures a strong consistency already visible in the convergence curves. After tuning, this variant almost always converges to the same quality region. Its fitness of 273 is by far the lowest (best) of the three algorithms.
- **Adaptive** - In this panel, a bimodal density of the fitness is observed. One tight area is centered at $\approx 570 - 590$ and a second one is observed around 530. Although the fitness is clearly above the *baseline* variant, the lower section shows that a minority of runs can approach *baseline*'s performance.
- **Repair** - This violin plot is the widest of all three. Stretching from ≈ 500 up to 740 with a mean near 660. This observation indicates that the computationally heavier operator does not translate into more reliable outcomes. Instead, its stochastic nature yields both occasional optical runs ≈ 510 and several poor ones above 700.

Taking everything into account, these plots reinforce the ranking suggested by the earlier analyses. The *baseline* variant delivers the best and most stable performance and the *adaptive* approach attains a middle ground with moderate variance. Meanwhile, the *repair* variant delivers less than the other approaches, it is noticeably less predictable.

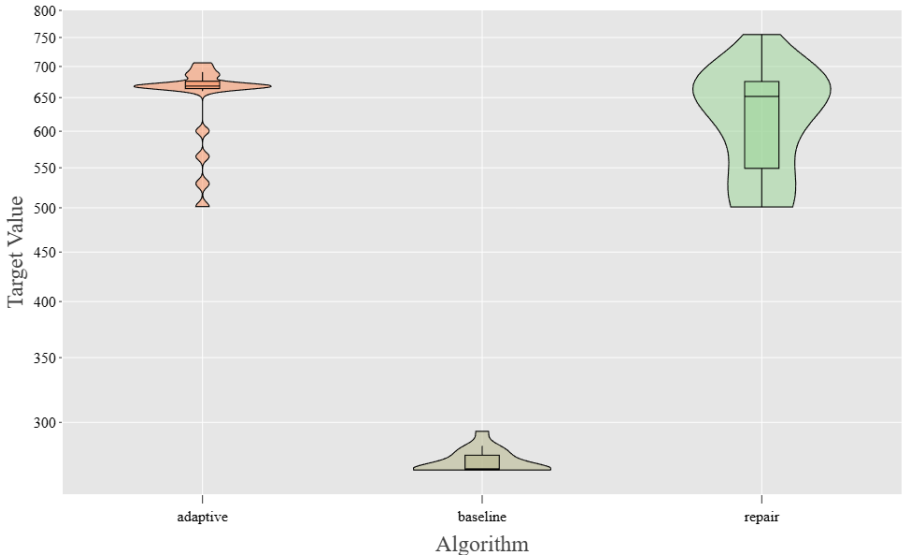
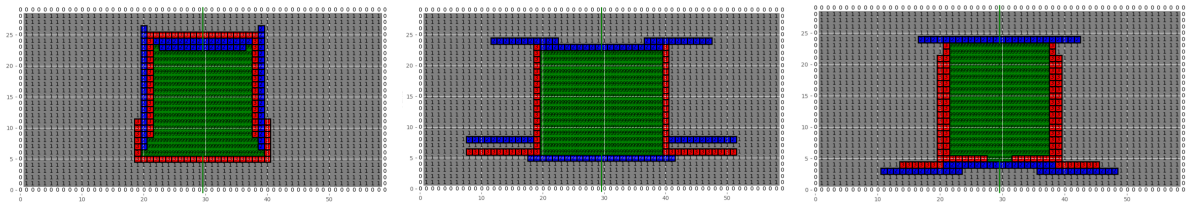


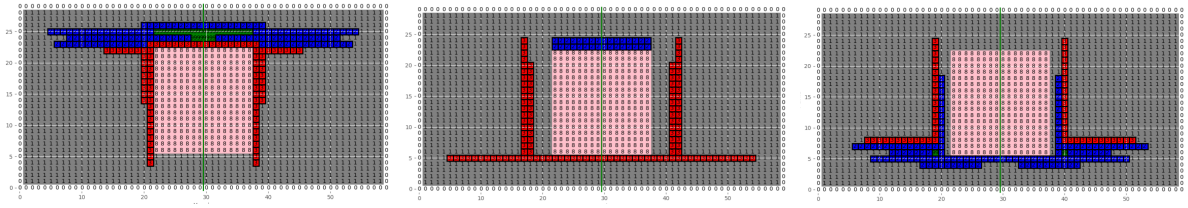
Figure 6.9: Empirical–probability density of each variant. Each curve integrates to 1 and is related to the corresponding *ECDF* by $p(x) = \frac{d}{dx}F(x)$. Peaks highlight where each variant most often converges.

6.2.4 Illustrative Designs



(a) Baseline (b) Adaptive (c) Repair

Figure 6.10: Highest-quality layouts discovered by each GA variant. Green cells mark the isolated interior region. Blue and red tiles are `Line` and `L` shapes respectively.



(a) Baseline (b) Adaptive (c) Repair

Figure 6.11: Representative low quality outcomes. Gaps, alignments or excessive material lead to higher isolation and connectivity penalties, explaining the tails observed in Fig. 6.9.

6.3 Generalization

To check whether the tuned GA settings remain valuable beyond the exact setup on which they were optimized, the best incumbent is run under the two variations already introduced in Section 5.3.

6.3.1 Additional Parts

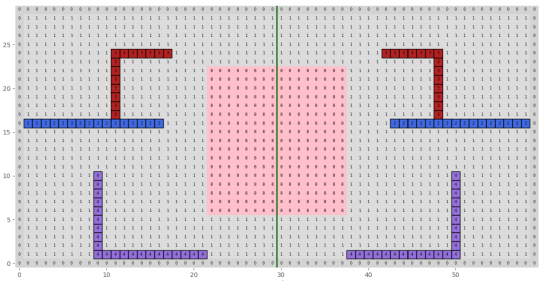


Figure 6.12: Best individual after 25 000 evaluations when the catalog is extended with additional parts.

(a) Additional-shapes run (Fig. 6.12).

Metric	Value
Function evaluations	25 000
Local connectivity	0.10
Global connectivity	0
Shape count	3.65
Isolation	0.10
Distance	0.12

(b) Altered-interior run (Fig. 6.13).

Metric	Value
Function evaluations	25 500
Local connectivity	0.30
Global connectivity	0
Shape count	0.59
Isolation	0.54
Distance	0.63

Table 6.4: Performance indicators for two robustness tests.

The additional geometric parts increase the search space combinations. Fig. 6.12 shows that the GA now constructs a sparse pattern rather than a solid enclosure. Although it tries to keep the structure connected and maintains a low border penalty, two signals reveal poor adaptation:

- **Connectivity** drops only one-third of what the tuned baseline achieved on its native task, reflecting fewer flange contacts between heterogeneous parts. Global connectivity does not reflect the actual number of components in this test.
- **Shape Count** The shape count explodes to 3.65: many placement attempts succeed (more parts on the grid), but they do not cooperate to block the interior ports. An edge case of ratios exceeding 1 can also be observed because the reference values were calibrated for the original shape limit.

Introducing new shapes without re-tuning leads the *GA* to explore attractive but not optimal layouts. The incumbent weights fail to encourage the positive alignments required for the parts to attach. Encouraging the need for either adaptive penalties or a fresh optimization pass when the design catalog changes.

6.3.2 Interior Region

With the interior replaced by the top and bottom ports, the untuned *baseline* GA reaches a less optimal result than the original *baseline*. Table 6.4b presents the causes:

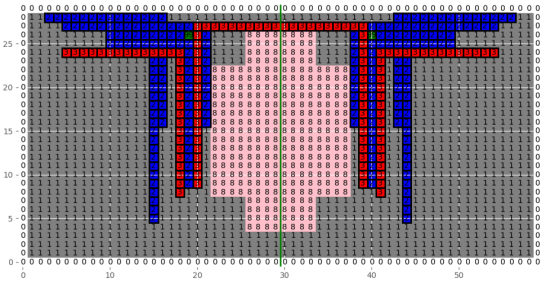


Figure 6.13: Best individual found after 25,000 evaluations when the interior is replaced by the new interior geometry.

- **Isolation** shoots up to 0.54 because the two side ports open large corridors that the original penalty weights underestimate.
- **Connectivity** is modest but acceptable, showing that the algorithm still joins most parts. Nevertheless, the border distance term doubles, which indicates repeated placements too close to the outer wall.
- **Shape Count** A higher shape ratio signals that many candidate parts are rejected for collisions, wasting evaluations.

Chapter 7

Conclusion

The study confirms that the proposed 2-D GA framework can reach geometrically feasible *BIW* layouts for a single connected component. The default *GA* settings reach more than 94% isolation of the interior. Introducing a smooth weight schedule accelerates early isolation but yields no significant fitness gain at termination. This suggests only a partial benefit over static priorities. Meanwhile, a lightweight repair heuristic reliably removes residual fragments in difficult seeds, improving final penalties without prohibitive runtime. Finally, *SMAC* tuning discovers parameter sets that outperform the best hand-tuned recipe by roughly 14 % in median fitness. Together, these results answer the four research questions affirmatively. They show that disciplined constraint handling based on a simplified **2-D** geometry is achievable in the context of *BIW*. Hence, the framework is a *reference implementation* that future work can evolve into a full 3-D, physics-coupled digital body-design system.

Limitations and Future Work

- **2-D proof-of-concept** - The study operates on a single planar slice; extending to full 3-D shell geometries will require richer primitives, manufacturability constraints and an *FEM* stiffness objective.
- **No structural fitness** - Weight and stiffness are mentioned as motivation, yet the optimization balances only geometric representations of these constraints. Coupling the GA with rapid structural surrogates or multi-fidelity *FE* evaluations is a natural next step.
- **Computational cost** - Even in 2-D, the Repair GA consumed 4$\times$ more CPU per evaluation. Parallel island models or GPU-accelerated evaluations could further speed up this for a *3D* work.
- **Limited catalog and generalization** - Future research should investigate online penalty scheduling that adapts to observed patterns and meta-learning across problem instances.

This thesis delivers a foundational demonstrator for a reproducible GA pipeline that negotiates isolation versus connectivity in a simplified BIW setting. With targeted improvements, focused on 3-D physics, adaptive penalties and surrogate-assisted evaluation, this approach can grow into a practical assistant for early-stage automotive body design.

References

[1] International Organization of Motor Vehicle Manufacturers, *2024 world motor vehicle production and sales*, Available at: https://www.oica.net/wp-content/uploads/Passenger-Cars-2024.pdf (accessed 2025-05-26), 2024.

[2] P. Oosterhuis and J. Jin, "A review of body-in-white structural design challenges," *International Journal of Automotive Technology*, vol. 21, no. 5, pp. 873–887, 2020.

[3] R. D. Cook, *Finite Element Modeling for Stress Analysis*. John Wiley & Sons, 2001.

[4] J. Cagan and M. Campbell, "Computational design synthesis: A survey," *AI EDAM*, vol. 33, no. 2, pp. 217–234, 2019.

[5] P. Y. Papalambros and D. J. Wilde, *Principles of Optimal Design: Modeling and Computation*. Cambridge, UK: Cambridge University Press, 2000.

[6] M. C. Moynihan and J. M. Allwood, "The lightweighting challenge for passenger vehicles," *Energy Policy*, vol. 73, pp. 426–432, 2014.

[7] W. Zhang and J. Xu, "Advanced lightweight materials for automobiles: A review," *Materials Design*, vol. 221, p. 110 994, 2022, ISSN: 0264-1275. DOI: https://doi.org/10.1016/j.matdes.2022.110994.

[8] National Highway Traffic Safety Administration, *Corporate average fuel economy (cafe) standards: Model years 2024–2026, final rule*, Federal Register Vol. 87, No. 100, 2022.

[9] European Commission, *Regulation (eu) 2019/631: Co_2 emission performance standards for passenger cars and vans*, Official Journal of the European Union L111/13, 2019.

[10] S. Carlucci, V. Bianco, and O. Pierini, "Structural design optimization for lightweight vehicles: State of the art and future perspectives," *International Journal of Automotive Technology*, vol. 20, no. 2, pp. 195–204, 2019. DOI: 10.1007/s12239-019-0018-6.

[11] S. Liu and Y. Hu, "Application of finite-element analysis in body-in-white development," *Automotive Engineer*, vol. 42, no. 3, pp. 28–33, 2017.

[12] H. K. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson, 2007.

[13] J. Reimpell, H. Stoll, and J. Betzler, *The Automotive Chassis: Engineering Principles*, 2nd ed. SAE International, 2001.

[14] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Chichester, UK: John Wiley & Sons, 2002.

[15] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog, 1973.

[16] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[17] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: University of Michigan Press, 1975.

[18] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison–Wesley, 1989.

[19] O. Sigmund, "Numerical instabilities in topology optimization," *Structural Optimization*, vol. 16, no. 1, pp. 68–75, 1998.

[20] V. K. Valsalam and R. Miikkulainen, "Evolving symmetry for modular system design," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 3, pp. 368–386, 2011. DOI: 10.1109/TEVC.2011.2112663.

[21] F. Xiong, X. Zou, Z. Zhang, and X. Shi, "A systematic approach for multi-objective lightweight and stiffness optimization of a car body," *Structural and Multidisciplinary Optimization*, vol. 62, Dec. 2020. DOI: 10.1007/s00158-020-02674-5.

[22] Y. He, W. Xu, and F. Gu, "Multi-objective optimization of automotive front rail based on surrogate model and nsga-ii," in Mar. 2021, pp. 251–260, ISBN: 978-3-030-68302-3. DOI: 10.1007/978-3-030-68303-0_20.

[23] C. Soto, "Structural topology optimization for crashworthiness," *International journal of crashworthiness*, vol. 9, pp. 277–283, Jun. 2004. DOI: 10.1533/ijcr.2004.0288.

[24] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[25] H. Zhou, D.-Q. Jiao, Q.-Y. Wang, Q.-Y. Chen, L. Xin, and G.-L. Wang, "Aerodynamic shape optimization of passenger car fender based on the ffd method," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 239, Dec. 2023. DOI: 10.1177/09544070231210037.

[26] I. Park and D. Papadimitriou, "Efficient surrogate-based nvh optimization of a full vehicle using frf based substructuring," vol. 2, Apr. 2020. DOI: 10.4271/2020-01-0629.

[27] N. K. Brown, A. P. Garland, G. M. Fadel, and G. Li, ""deep reinforcement learning for engineering design through topology optimization of elementally discretized design domains"," *Materials Design*, vol. 218, p. 110 672, 2022, ISSN: 0264-1275. DOI: https://doi.org/10.1016/j.matdes.2022.110672.

[28] Li, Haoran, Zhou, Haosu, and Li, Nan, "An integrated convolutional neural network-based surrogate model for crashworthiness performance prediction of hot-stamped vehicle panel components," *MATEC Web Conf.*, vol. 401, p. 03 013, 2024. DOI: 10.1051/matecconf/202440103013.

[29] J. Yu and B. Jafarpour, "Active learning for well control optimization with surrogate models," *SPE Journal*, vol. 27, no. 05, pp. 2668–2688, Oct. 2022, ISSN: 1086-055X. DOI: 10.2118/209191-PA. eprint: https://onepetro.org/SJ/article-pdf/27/05/2668/3015263/spe-209191-pa.pdf.

[30] S. van Rijn, S. Schmitt, M. Olhofer, M. van Leeuwen, and T. Bäck, "Multi-fidelity surrogate model approach to optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '18, Kyoto, Japan: Association for Computing Machinery, 2018, pp. 225–226, ISBN: 9781450357647. DOI: 10.1145/3205651.3205757.

[31] C. A. Coello Coello, "A comprehensive survey of evolutionary-based multiobjective optimization techniques," *Knowledge and Information Systems*, vol. 1, pp. 269–308, 2000.

[32] F. Ahmed, K. Deb, and B. Bhattacharya, "Structural topology optimization using multi-objective genetic algorithm with constructive solid geometry representation," *Applied Soft Computing*, vol. 39, pp. 240–250, 2016, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2015.10.063.

[33] W. Li, Y. Shi, C. Wang, Y. Tang, Y. Yu, and Z. Liu, "Structural topology optimization of reflective mirror based on objective of wavefront aberration," *Machines*, vol. 10, p. 1043, Nov. 2022. DOI: 10.3390/machines10111043.

[34] O. Sigmund, "A 99 line topology optimization code written in matlab," *Structural and Multidisciplinary Optimization*, vol. 21, pp. 120–127, 2002.

[35] X. Cui, B. Panda, C. Chin, N. Sakundarini, C.-T. Wang, and K. Pareek, "An application of evolutionary computation algorithm in multidisciplinary design optimization of battery packs for electric vehicle," *Energy Storage*, vol. 2, Apr. 2020. DOI: 10.1002/est2.158.

[36] G. CHENG and Z. J. and, "Study on topology optimization with stress constraints," *Engineering Optimization*, vol. 20, no. 2, pp. 129–148, 1992. DOI: 10.1080/03052159208941276. eprint: https://doi.org/10.1080/03052159208941276.

[37] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, Jul. 2012.

[38] C. Doerr, H. Wang, F. Ye, S. van Rijn, and T. Bäck, "Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics," *arXiv e-prints:1810.05281*, Oct. 2018. arXiv: 1810.05281.

[39] H. Wang, D. Vermetten, F. Ye, C. Doerr, and T. Bäck, "Iohanalyzer: Detailed performance analyses for iterative optimization heuristics," *ACM Transactions on Evolutionary Learning and Optimization*, vol. 2, no. 1, Apr. 2022.

[40] M. Lindauer, K. Eggensperger, M. Feurer, *et al.*, "Smac3: A versatile bayesian optimization package for hyperparameter optimization," *Journal of Machine Learning Research*, vol. 23, no. 54, pp. 1–9, 2022.

Code availability

All source code, configuration files and example logbooks needed to reproduce the experiments are publicly available at github.com/andreim77/biwga (tag v1.0).