



Instituto Politécnico Nacional

Centro de Investigación en Computación

Laboratorios de Inteligencia Artificial, y Simulación y
Modelado



**An Evolutionary Strategy Supported by Domain Knowledge for
Aircraft Automatic Conceptual Design**

Tesis

Que para obtener el grado de:
Maestría en Ciencias de la Computación

Presenta:

Ing. Jesús Pérez Romero

Directores de tesis:

Dr. Salvador Godoy Calderón

Dr. Juan Carlos Chimal Eguía

Ciudad de México, México

Junio, 2017



INSTITUTO POLITÉCNICO NACIONAL SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México siendo las 15:30 horas del día 14 del mes de junio de 2017 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

“An Evolutionary Strategy Supported by Domain Knowledge for Aircraft Automatic Conceptual Design”

Presentada por el alumno:

PÉREZ	ROMERO	JESÚS							
Apellido paterno	Apellido materno	Nombre(s)							
		Con registro:	A	1	5	0	1	8	6

aspirante de: **MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron **APROBAR LA TESIS**, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA Directores de Tesis



Dr. Salvador Godoy Calderón



Dr. Juan Carlos Chimal Eguía



Dr. Adolfo Guzmán Arenas



Dr. Ricardo Barrón Fernández



Dr. Rogelio Gerardo Hernández García



Dr. Francisco Hiram Calvo Castro

PRESIDENTE DEL COLEGIO DE PROFESORES



Dr. Marco Antonio Ramírez Salinas

ESTADO MEXICANO
INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACION
EN COMPUTACION



INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 14 del mes Junio del año 2017, el (la) que suscribe Jesús Pérez Romero alumno (a) del Programa de Maestría en Ciencias de la Computación con número de registro A150186, adscrito a Centro de Investigación en Computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Salvador Godoy Calderón y Dr. Juan Carlos Chimal Eguía y cede los derechos del trabajo intitulado An Evolutionary Strategy Supported by Domain Knowledge for Aircraft Automatic Conceptual Design, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección vgprom@vgprom.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Jesús Pérez Romero

Nombre y firma

Resumen

En este trabajo se propone una estrategia evolutiva dirigida por simulación y soportada por conocimiento previo del dominio para la automatización del diseño conceptual de una aeronave, junto con un marco de trabajo para modelar y definir el proceso de diseño y la aeronave vista como un sistema de ingeniería.

La solución propuesta se desarrolla a través de la realización del diseño conceptual de una aeronave de ala fija que desempeña una misión de planeo (al interior de un simulador de vuelo) y es implementada a través de cuatro variantes de la estrategia diseñada.

Este trabajo esencialmente incorpora piezas de conocimiento del dominio en el proceso automático de diseño conceptual de una aeronave para mejorar la búsqueda de un diseño óptimo y para ajustarse de mejor manera a la naturaleza basada en toma de decisiones a través de selección de alternativas y a la relevancia que la experiencia y el conocimiento del campo de aplicación tienen en la fase de diseño conceptual. Esta sinergia es lograda a través de la combinación de operadores basados en conocimiento previo del dominio del sistema (dos operadores son propuestos: operador de variación preconfigurada y operador de variación correctiva) con operadores evolutivos clásicos (recombinación y mutación) para producir diseños conceptuales (soluciones).

Aunque diseñar una aeronave es usualmente un problema de optimización de diseño multiobjetivo y multidisciplinario, el aspecto multidisciplinario es puesto fuera del alcance de este trabajo, pospuesto para trabajo futuro, sin embargo, la naturaleza multiobjetivo es atendida a través de técnicas de escalarización por lo que una sola solución es obtenida como salida.

Palabras clave: Algoritmo evolutivo, diseño conceptual de aeronaves, conocimiento del dominio, estrategia evolutiva

Abstract

This thesis introduces a simulation-driven evolutionary strategy, supported by domain knowledge, for aircraft conceptual design automation along with a framework to model and define the design process and the aircraft as an engineering system.

The proposed solution is developed through the realization of a conceptual design of a fixed-wing aircraft to perform a gliding mission (inside a flight simulator), and it is implemented through four variants of the designed strategy.

This work essentially incorporates pieces of domain knowledge into the automatic conceptual design process of an aircraft to improve the search for an optimal design, and to better match the —selection decision-making nature— and —experience and field knowledge relevance— in the conceptual design phase. This synergy is achieved by combining operators based on prior knowledge of the system domain (two operators are proposed: preconfigured variation operator and corrective variation operator), with classical evolutionary algorithms operators (recombination and mutation) to produce design concepts (solutions).

Although designing an aircraft is usually a multi-objective, multi-disciplinary design optimization problem, the multi-disciplinary aspect is taken out of the scope in this work, postponed for future work; however, multi-objective nature is tackled through scalarization techniques, therefore, a single solution is obtained as output.

Keywords: Evolutionary algorithm, aircraft conceptual design, domain knowledge, evolutionary strategy

Contents

Contents	I
List of figures	IV
List of tables	IX
List of algorithms	X
Glossary	XI
Abbreviations	XIII
Units of measurement	XV
Nomenclature	XVI
Aircraft design nomenclature	XIX
1 Introduction	1
1.1 Background	1
1.1.1 Aircraft design	2
1.2 Motivation	3
1.3 Objectives	5
1.4 Disclaimer	6
1.5 Document outline	6
2 Theoretical framework	8
2.1 Problem domain	8
2.2 Problem type	8
2.2.1 Optimization	9
2.2.1.1 Optimization problem characterization	9
2.2.2 Design optimization	10
2.2.2.1 Design optimization problem formulation	10
2.3 Solution approach	11
2.3.1 Simulation-driven design	11
2.3.2 Hybrid metaheuristic approach	12
2.3.3 Optimization problem scalarization	13
2.3.4 Evolutionary design	14
2.3.4.1 Evolutionary approach suitability	14
2.3.4.2 Evolutionary approach implementation	15

2.3.5	Domain knowledge incorporation	15
2.3.5.1	Knowledge	15
2.3.5.2	Knowledge based system	16
2.3.5.3	Rule based system	17
2.4	Evolutionary computing framework	18
2.4.1	General evolutionary algorithm model	18
2.4.2	Evolutionary algorithm components and properties	20
2.4.3	Evolutionary algorithm generalized algorithm	21
2.4.4	Evolutionary algorithm instantiation	21
2.4.5	Mainstream evolutionary algorithms instances	22
2.4.6	Evolutionary operators	22
2.4.6.1	Variation operators	23
2.4.6.1.1	Recombination variation operator (recombinator)	23
2.4.6.1.2	Mutation variation operator (mutator)	25
2.4.6.1.3	Combined variation operators	26
2.4.6.1.4	Mutation and recombination effects	26
2.4.6.2	Selection operators (selectors)	27
2.4.7	Extension points	29
2.4.7.1	Parameters as functions	29
2.4.7.2	Variation rates sampling frequency	29
2.4.7.3	Selection auxiliary mechanisms	30
3	Related work	31
3.1	Literature review	31
3.2	Software tools and frameworks	34
4	Methodology and solution development	37
4.1	Design requirements	39
4.2	Design problem definition	40
4.3	System and knowledge representation	46
4.3.1	Hierarchical/multi-layer structural model	46
4.3.2	Design variables	49
4.3.3	Domain model	56
4.4	Simulation-driven design preparation	62
4.4.1	Scalarization	63
4.4.2	Simulation model	64
4.4.2.1	Computer simulation	66
4.5	Knowledge based evolutionary strategy	68
4.5.1	Top-down initialization	71
4.5.2	Design variation	75
4.5.2.1	Cumulative candidate offspring	76
4.5.2.2	Target selection	76
4.5.2.3	Operators order	77
4.5.2.4	Operators distribution	77
4.5.2.5	Variation operator failover	78
4.5.2.6	Population injection	78
4.5.2.7	Knowledge based variation operators	79
4.5.2.7.1	Preconfigured variation operator	79
4.5.2.7.2	Corrective variation operator	87

4.5.2.8	Classical evolutionary variation operators	91
4.5.3	Generation selection	95
4.5.4	Simulation-driven design integration	96
4.5.5	Specific implementations	96
4.5.5.1	I: Minimal evaluations per generation	97
4.5.5.2	II: Distributed variation operators	97
4.5.5.3	III: Maximum probability of success	98
4.5.5.4	IV: Medium number of evaluations per generation	98
4.6	Complementary system representation	98
5	Experiments and results	103
5.1	Experiment description	103
5.2	Results	106
5.2.1	Results analysis	106
5.2.1.1	Individual objectives	107
5.2.1.2	Multivariate analysis	108
5.2.2	Initial shared population	111
5.2.2.1	Design variables	111
5.2.2.2	Visual models	114
5.2.3	Fittest design concepts	117
5.2.3.1	Design variables	117
5.2.3.2	Visual models	119
5.3	Additional experiment: weather enabled	127
5.3.1	Results	128
6	Conclusions	133
A	Design variables reference	i
B	Aircraft various reference views	viii
B.1	Aircraft axes	viii
B.2	Aircraft directions	ix
B.3	Aircraft design variables	x
	References	xvi

List of figures

1.1	Design operations	2
2.1	Data, information and knowledge	16
2.2	EAs general schema	19
2.3	Fitness proportional selection (FPS)	28
2.4	Stochastic universal sampling (SUS)	29
4.1	Overall solution activity diagram	38
4.2	Gliding mission	39
4.3	Design problem definition procedure	40
4.4	Aircraft horizontal flight paths	42
4.5	Great-circle distance	43
4.6	System representation - structure	46
4.7	Fixed wing aircraft structural model	47
4.8	System representation - structure + abstraction layers	48
	a Abstraction layers	48
	b Extended model	48
4.9	Full system representation model	50
4.10	Aircraft family layer - wing shape family classification	52
4.11	Aircraft configuration layer - swept type classification	52
4.12	Generic domain model (extension point elements in blue)	57
4.13	Lofting geometry example	58
4.14	Some superellipse based cross sections	59
4.15	Fixed wing aircraft domain model (extension point elements in blue and green)	60
4.16	Fuselage mid section cross section generated by a superellipse with parameters $\check{a} = 1.1\check{b}, \check{m} = 1.98, \check{n} = 1.98$	61
4.17	National Advisory Committee for Aeronautics (NACA) Airfoil examples	66
4.18	Knowledge based evolutionary strategy - overall activity diagram	70
4.19	Top-down initialization activity diagram	73
4.20	Aircraft conceptual design initialization (partial)	74
4.21	Variation operator overall activity diagram	75
4.22	Preconfigured variation operator application activity diagram	80
4.23	Monoplane preconfigured variation example	84
4.24	Vertical stabilizer on/off preconfigured variation example	84
4.25	Swap swept direction preconfigured variation example	85
4.26	X-wing preconfigured variation example	86
4.27	Corrective variation operator application activity diagram	87
4.28	Corrective variation example	90

4.29	Evolutionary strategy implementation I	97
4.30	Evolutionary strategy implementation II	98
4.31	Evolutionary strategy implementation III	99
4.32	Evolutionary strategy implementation IV	99
4.33	Fixed wing aircraft visual model - front wireframe	100
4.34	Fixed wing aircraft visual model - front solid	100
4.35	Fixed wing aircraft visual model - top wireframe	101
4.36	Fixed wing aircraft visual model - top solid	101
4.37	Fixed wing aircraft visual model - side wireframe	102
4.38	Fixed wing aircraft visual model - side solid	102
4.39	Fixed wing aircraft visual model - isometric solid view solid	102
5.1	Global convergence curve	106
5.2	Global results by objective I	109
5.3	Global results by objective II	110
5.4	Multivariate parallel coordinates plot I	110
a	DE*	110
b	CDE*	110
c	CEO	110
d	KBO	110
5.5	Multivariate parallel coordinates plot II	111
a	I	111
b	II	111
c	III	111
d	IV	111
5.6	Initial shared population aircraft design concept - pointed wing shape family representative	114
a	Wireframe top view	114
b	Solid top view	114
c	Wireframe front view	114
d	Solid front view	114
e	Wireframe side view	114
f	Solid side view	114
g	Wireframe isometric view	114
h	Solid isometric view	114
5.7	Initial shared population aircraft design concept - constant chord wing shape family representative	115
a	Wireframe top view	115
b	Solid top view	115
c	Wireframe front view	115
d	Solid front view	115
e	Wireframe side view	115
f	Solid side view	115
g	Wireframe isometric view	115
h	Solid isometric view	115
5.8	Initial shared population aircraft design concept - trapezoidal wing shape family representative	116
a	Wireframe top view	116
b	Solid top view	116

	c	Wireframe front view	116
	d	Solid front view	116
	e	Wireframe side view	116
	f	Solid side view	116
	g	Wireframe isometric view	116
	h	Solid isometric view	116
5.9		Global fittest aircraft design concept generated by algorithm IV	119
	a	Wireframe top view	119
	b	Solid top view	119
	c	Wireframe front view	119
	d	Solid front view	119
	e	Wireframe side view	119
	f	Solid side view	119
	g	Wireframe isometric view	119
	h	Solid isometric view	119
5.10		Fittest aircraft design concept generated by algorithm II	120
	a	Wireframe top view	120
	b	Solid top view	120
	c	Wireframe front view	120
	d	Solid front view	120
	e	Wireframe side view	120
	f	Solid side view	120
	g	Wireframe isometric view	120
	h	Solid isometric view	120
5.11		Fittest aircraft design concept generated by algorithm I	121
	a	Wireframe top view	121
	b	Solid top view	121
	c	Wireframe front view	121
	d	Solid front view	121
	e	Wireframe side view	121
	f	Solid side view	121
	g	Wireframe isometric view	121
	h	Solid isometric view	121
5.12		Fittest aircraft design concept generated by algorithm III	122
	a	Wireframe top view	122
	b	Solid top view	122
	c	Wireframe front view	122
	d	Solid front view	122
	e	Wireframe side view	122
	f	Solid side view	122
	g	Wireframe isometric view	122
	h	Solid isometric view	122
5.13		Fittest aircraft design concept generated by algorithmic variant KBO	123
	a	Wireframe top view	123
	b	Solid top view	123
	c	Wireframe front view	123
	d	Solid front view	123
	e	Wireframe side view	123
	f	Solid side view	123

g	Wireframe isometric view	123
h	Solid isometric view	123
5.14	Fittest aircraft design concept generated by algorithmic variant CEO	124
a	Wireframe top view	124
b	Solid top view	124
c	Wireframe front view	124
d	Solid front view	124
e	Wireframe side view	124
f	Solid side view	124
g	Wireframe isometric view	124
h	Solid isometric view	124
5.15	Fittest aircraft design concept generated by algorithm DE*	125
a	Wireframe top view	125
b	Solid top view	125
c	Wireframe front view	125
d	Solid front view	125
e	Wireframe side view	125
f	Solid side view	125
g	Wireframe isometric view	125
h	Solid isometric view	125
5.16	Fittest aircraft design concept generated by algorithmic variant CDE*	126
a	Wireframe top view	126
b	Solid top view	126
c	Wireframe front view	126
d	Solid front view	126
e	Wireframe side view	126
f	Solid side view	126
g	Wireframe isometric view	126
h	Solid isometric view	126
5.17	Weather enabled experiment convergence curve	128
5.18	Weather enabled experiment objectives	129
5.19	Fittest aircraft design concept generated by algorithm III with weather enabled	132
a	Wireframe top view	132
b	Solid top view	132
c	Wireframe front view	132
d	Solid front view	132
e	Wireframe side view	132
f	Solid side view	132
g	Wireframe isometric view	132
h	Solid isometric view	132
B.1	Aircraft axes	viii
B.2	Aircraft directions	ix
B.3	Wing type part design variables I	x
B.4	Wing type part design variables II	xi
B.5	Wing type part design variables III	xi
B.6	Wing type part design variables IV	xii
B.7	Wing type part profile design variables	xii
B.8	Horizontal stabilizer design variables (partial)	xiii

B.9 Vertical stabilizer design variables (partial)	xiii
B.10 Vertical/horizontal stabilizer design variables (partial)	xiv
B.11 Fuselage design variables	xv

List of tables

4.1	Aircraft performance objectives	41
4.2	Aircraft design objectives	42
4.3	Design variables vector example, a design concept with 2 wing planes, a taper ratio λ_1 of 0.43 for the first wing plane, and a taper ratio λ_2 of 0.76 for the second wing plane	56
4.4	Domain model generic classes	58
4.5	Utopian aircraft design concept objectives weights	63
4.6	Utopian aircraft design concept performance	64
4.7	Artificial environment conditions	68
4.8	Simulation settings	68
5.1	Experiment general parameters and settings	104
5.2	Knowledge based variation operators parameters and settings	104
	a Corrective variation	104
	b Preconfigured variation	104
5.3	Recombination parameters and settings	105
5.4	Mutation parameters and settings	105
5.5	Global results	107
5.6	Global results by objective	108
5.7	Initial population design concepts design variables	111
5.8	Global fittest design concepts design variables	117
5.9	Additional experiment weather conditions	127
5.10	Weather enabled experiment global results	128
5.11	Additional experiment fittest design concept design variables	129
A.1	Aircraft design variables reference	i

List of algorithms

2.1	Generalized Evolutionary algorithm (EA) based on [8]	21
4.1	Knowledge base evolutionary strategy - overall	71
4.2	Top-down initialization	74
4.3	Preconfigured variation operator	83
4.4	Corrective variation operator	89
4.5	Custom recombination operator	93
4.6	Mutation operator	94

Glossary

***k*-means** A clustering method that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster[79]

aft Towards the stern (Fig. B.2)

bow Front part of an ship

chord length The length of the straight line connecting the leading edge and trailing edge of a wing section (cross section)

cruise Level flight after an aircraft climbs to a set altitude and before it begins to descend[78]

cumulus A type of tall, white cloud with a wide, flat base and rounded shape[12]

drag A force acting opposite to the oncoming flow

fore Towards the bow (Fig. B.2)

framework A conceptual structure intended to serve as a support or guide for the building of something that expands the structure into something useful[66]

glide Unpowered flight distance

gust A sudden strong wind[12]

heading Aircraft orientation about its vertical axis (Fig. B.1)

leading edge The wing front border

moment The force or torque that makes an airfoil tends to pitch down or up

momentum The force that keeps an object moving or keeps an event developing after it has started[12]

outboard Right side of an aircraft (from a top view, nose at top, tail at bottom) (Fig. B.2)

parametric simulation model A simulation model not defined by mathematical equations but by a set of parameters

payload The amount of passengers and/or cargo that an aircraft can carry[12]

pitch Aircraft orientation about its lateral axis (Fig. B.1)

port Left side of an aircraft (from a top view, nose at top, tail at bottom) (Fig. B.2)

reynolds number The ratio of inertial forces to viscous forces within a fluid which is subjected to relative internal movement due to different fluid velocities[80]

roll Aircraft orientation about its longitudinal axis (Fig. B.1)

shear A change in wind direction

span The length of a wing (pair of wings) from tip to tip

stern Rear part of a ship

systems engineering A theoretical model that lies between pure mathematics and theories of specialized disciplines that has the goal of designing an engineering system[70]

thermal A rising flow of warmer air

trailing edge The wing rear border

true airspeed The speed of the aircraft relative to the airmass in which it is flying[81]

wind tunnel A closed passage or room through which currents of air are forced in order to study the effects of moving air on aircraft[12]

yaw Aircraft movement around its vertical axis (Fig. B.1)

Abbreviations

2D two dimensional

3D three dimensional

AI artificial intelligence

CAD computer-aided design

CDE* custom differential evolution algorithm (adjusted to $\mu = 3$ with differential arithmetic recombination)

CEO classical evolutionary only strategy variant

DE differential evolution

DE* differential evolution algorithm (adjusted to $\mu = 3$)

EA evolutionary algorithm

EC evolutionary computing

EP evolutionary programming

ES evolution strategy

FAA Federal Aviation Administration

FPS fitness proportional selection

fwd forward

GA genetic algorithm

JSON JavaScript Object Notation

KB knowledge base

KBO knowledge based only strategy variant

KBS knowledge based system

mid middle

NACA National Advisory Committee for Aeronautics

NASA National Aeronautics and Space Administration

PDF probability density function

PSO particle swarm optimization

RBS rule based system

SBO surrogate-based optimization

SDK software development kit

STAGN Stagnation

SUS stochastic universal sampling

TAS true airspeed

XML Extensible markup language

Units of measurement

Unit	Name	Description
deg ²	square degree	Unit of measure of solid angle equal to $(\frac{\pi}{180})^2$
ft	foot	Unit of length equivalent to 0.3048 meters exactly
kt	knot	Unit of speed equal to 1 NM per hour
lb	pound-mass	Unit of mass legally defined as exactly 0.45359237 kilograms
NM	nautical mile	Unit of distance equivalent to 1852 meters exactly

Nomenclature

Symbol	Name
l	cumulative
m	sampling frequency
c_{Pf}	constant value for differential mutation probability
Pf	mutation probability for custom differential mutation/arithmetc recombination operator
O	set of design objectives
g_e	operator application frequency across generations
g_0	operator application starting generation number
d_X	method of combination of domains in recombination and differential mutation
d_M	type of scope of domain used in mutation
R_{\oplus}	earth mid radius
$\Lambda^{\bar{f}}$	set of parameters of fitness function
Λ^{ι}	set of parameters of initialization function
Λ^I	set of parameters of injection operator
Λ^{ς}	set of parameters of selection function
Λ^{σ}	set of parameters of target selection function
Λ^{Υ}	set of parameters of variation operators
η	failover variation operator
h_f	final altitude
g	generation
Q	aircraft glide end point
\check{G}	glide
d_{gc}	great-circle distance
ψ	heading
h_0	initial altitude
ι	initialization function (i.e., top-down initializer)
TAS_0	initial TAS
t_0	initial time

I	injection operator
τ^I	injection operator threshold
κ	number of members in x
$\dot{\varphi}$	latitude
$\dot{\lambda}$	longitude
g_{MAX}	maximum number of generations termination criterion
μ	population mean
t	n th simulation measurement
\tilde{x}	median
\vec{f}_{MIN}	minimum fitness evaluation value termination criterion
Mp	differential mutation probability
k	number of design objectives
\dot{k}	number of performance objectives
w	design objective weight
ϱ	variation operator ranking function (i.e., variation operator ranker)
\dot{r}	recombination and differential mutation parent selection function (i.e., parent selector)
\dot{r}	recombination and differential mutation parent index
\dot{O}	set of performance objectives
θ	pitch
μ	population size
X	population
r	radius
α	low probability/factor value for a random probability/factor function
β	high probability/factor value for a random probability/factor function
ρ	ranking function (i.e., ranker)
P	aircraft release point
φ	roll
\bar{x}	arithmetic sample mean (average)
κ	differential recombination scale factor
F	differential mutation scale factor
ς	selection function (i.e., selector)
τ^S	stagnation threshold
σ	target selection function (i.e., target selector)
C	set of termination criterion
T	total number of simulation measurements
Cr	uniform recombination probability
σ^2	population variance

Υ	set of variation operators
CE	classical evolutionary
E	evaluated
\vec{f}	scalarized fitness function of a design concept
KB	knowledge based
l	n th parameter
Λ^C	set of parameters of termination criteria
m	number of x in \vec{x}
\mathbb{N}	set of natural numbers
o	n th design objective
Ω	feasible design space
p	Minkowski distance order
R	ranked
\mathbb{R}	set of real numbers
\vec{u}	utopian design variables vector (i.e., utopian design concept)
x	design variable
\vec{x}	design variables vector (i.e., design concept)
X^{CE}	classical evolutionary local offspring
X^{KB}	knowledge based local offspring
\mathbb{Z}	set of integer numbers
\mathbb{Z}^+	set of positive integers
\mathbb{Z}^-	set of negative integers

Aircraft design nomenclature

Symbol	Name
\check{a}	superellipse width parameter
$afLT$	wing airfoil thickness level
$aflt$	wing airfoil type
$aflt_h$	horizontal stabilizer airfoil type
\mathcal{A}	aspect ratio: the ratio of the wing span to the chord length
\mathcal{A}_h	horizontal stabilizer aspect ratio
$\mathcal{A}l$	aspect ratio level
$\mathcal{A}l_h$	horizontal stabilizer aspect ratio level
\mathcal{A}_v	vertical stabilizer aspect ratio: the ratio of the height of the vertical stabilizer to the chord length
\mathcal{A}_{vh}	vertical/horizontal stabilizer aspect ratio
\check{b}	superellipse height parameter
c	airfoil chord
$\frac{\delta h_{max}}{c}$	horizontal stabilizer airfoil max camber
$\frac{\delta h_{max}}{c} \chi$	horizontal stabilizer airfoil max camber location
$\frac{x \delta_{max}}{c} \chi$	airfoil max camber location
$\frac{\delta_{max}}{c}$	airfoil max camber
$\frac{\delta v h_{max}}{c}$	vertical/horizontal stabilizer airfoil max camber
$\frac{\delta v h_{max}}{c} \chi$	vertical/horizontal stabilizer airfoil max camber location
$\frac{\delta v_{max}}{c}$	vertical stabilizer airfoil max camber
$\frac{\delta v_{max}}{c} \chi$	vertical stabilizer airfoil max camber location
$\frac{\delta_y}{c}$	airfoil thickness ordinate
ϵ	twist: The absolute difference between the incidence angle at the root and the incidence angle at the tip. Twist is negative if the leading edge of the tip is below the root leading edge, and positive otherwise
ϵ_h	horizontal stabilizer twist
ϵt	wing twist type
ϵt_h	horizontal stabilizer twist type

ϵt_{vh}	vertical/horizontal stabilizer twist type
ϵ_v	vertical stabilizer twist
ϵ_{vh}	vertical/horizontal stabilizer twist
f_{Cd}	fuselage drag coefficient
fh	fuselage height
$f_{\check{m}}$	mid fuselage cross section superellipse \check{m} parameter
$f_{\check{n}}$	mid fuselage cross section superellipse \check{n} parameter
fw	fuselage width
Γ	dihedral angle: the angle between the wing and the aircraft lateral axis, when viewed from the front
Γ_h	horizontal stabilizer dihedral angle
Γt	wing dihedral angle type
Γt_h	horizontal stabilizer dihedral angle type
Γt_{vh}	vertical/horizontal stabilizer dihedral angle type
Γ_v	vertical stabilizer dihedral angle
Γ_{vh}	vertical/horizontal stabilizer dihedral angle
$hpos_h$	horizontal stabilizer horizontal position
$hpos_w$	wing plane horizontal position
i	incidence: the angle between the the chord line of the wing at root and the aircraft longitudinal axis, when viewed from the side
i_h	horizontal stabilizer incidence
it	wing incidence angle type
it_h	horizontal stabilizer incidence angle type
it_{vh}	vertical/horizontal stabilizer incidence angle type
i_v	vertical stabilizer incidence
i_{vh}	vertical/horizontal stabilizer incidence
Λ	sweep angle: the angle between the lateral axis of the aircraft and a chord-line going from a chordwise point at the wing root to the same chordwise point at the tip
λ	taper ratio: the ratio of the wing chord length at root to the chord length at tip
Λd	sweep direction
Λd_h	horizontal stabilizer sweep direction
Λd_{vh}	vertical/horizontal stabilizer sweep direction
Λ_h	horizontal stabilizer sweep angle
λ_h	horizontal stabilizer taper ratio
ΛL	sweep level
Λl	sweep location: a location expressed as a percentage of the wing chord length (starting at leading edge) where a sweep angle is measured
λl	wing taper ratio level

ΛL_h	horizontal stabilizer sweep level
Λl_h	horizontal stabilizer sweep location
λl_h	horizontal stabilizer taper ratio level
ΛL_v	vertical stabilizer sweep level
Λl_v	vertical stabilizer sweep location
λl_v	vertical stabilizer taper ratio level
ΛL_{vh}	vertical/horizontal stabilizer sweep level
Λl_{vh}	vertical/horizontal stabilizer sweep location
λl_{vh}	vertical/horizontal stabilizer taper ratio level
lm	fuselage mid section length
l_n/d	fuselage nose section (fwd) length to diameter ratio
λt	wing taper ratio type
l_t/d	fuselage tail section (aft) length to diameter ratio
λt_h	horizontal stabilizer taper ratio level
λt_v	vertical stabilizer taper ratio level
λt_{vh}	vertical/horizontal stabilizer taper ratio level
Λ_v	vertical stabilizer sweep angle
λ_v	vertical stabilizer taper ratio
Λ_{vh}	vertical/horizontal stabilizer sweep angle
λ_{vh}	vertical/horizontal stabilizer taper ratio
m	2nd position digit in NACA 4-digit-series airfoil short-code corresponding to $\frac{\delta_{max}}{c}$
\check{m}	superellipse vertical curvature parameter
MLW	maximum landing weight in lb
\check{n}	superellipse horizontal curvature parameter
Nh	number of horizontal stabilizers
Nv	number of vertical stabilizers
Nvh	number of vertical/horizontal stabilizers
Nw	number of wing planes
n_y	fuselage nose tip vertical location
p	1st position digit in NACA 4-digit-series airfoil short-code corresponding to $\frac{x\delta_{max}}{c}x$
$pmxx$	NACA 4-digit-series airfoil short-code
S	planform surface: the wing shape projected surface, when viewed from above in ft^2
Sf	shape family
Sf_h	horizontal stabilizer shape family
Sf_v	vertical stabilizer shape family
Sf_{vh}	vertical/horizontal stabilizer shape family
S_h	horizontal tail planform surface

$\frac{S_h}{S_{vh}}$	vertical/horizontal stabilizer - horizontal tail ratio
S_v	vertical tail planform surface
$\frac{S_v}{S_{vh}}$	vertical/horizontal stabilizer - vertical tail ratio
SW_{vh}	vertical/horizontal stabilizer planform surface
$\frac{t_{h_{max}}}{c} x$	horizontal stabilizer airfoil max thickness location
$\frac{t_{h_{max}}}{c}$	horizontal stabilizer airfoil max thickness
$\frac{t_{max}}{c} x$	airfoil max thickness location
$\frac{t_{max}}{c}$	airfoil max thickness
$\frac{t_{vh_{max}}}{c} x$	vertical/horizontal stabilizer airfoil max thickness location
$\frac{t_{vh_{max}}}{c}$	vertical/horizontal stabilizer airfoil max thickness
$\frac{t_{v_{max}}}{c} x$	vertical stabilizer airfoil max thickness location
$\frac{t_{v_{max}}}{c}$	vertical stabilizer airfoil max thickness
t_y	fuselage tail tip vertical location
$\frac{t_y}{c}$	airfoil thickness ordinate
$vpos_h$	horizontal stabilizer vertical position
$vpos_w$	wing plane vertical position
$\frac{x}{c}$	chordwise abscissa
x_L	airfoil lower surface abscissa
x_U	airfoil upper surface abscissa
x_{vh}	vertical/horizontal stabilizer relative lateral arm
xx	3rd and 4th position digits in NACA 4-digit-series airfoil short-code corresponding to $\frac{t_{max}}{c}$
y_h	horizontal stabilizer relative vertical arm
y_L	airfoil lower surface ordinate
y_U	airfoil upper surface ordinate
y_{vh}	vertical/horizontal stabilizer relative vertical arm
y_w	wing plane relative vertical arm
z_h	horizontal stabilizer relative longitudinal arm
z_v	vertical stabilizer relative longitudinal arm
z_{vh}	vertical/horizontal stabilizer relative longitudinal arm
z_w	wing plane relative longitudinal arm

Introduction

1.1 Background

Based on the definition of National Aeronautics and Space Administration (NASA), an aircraft is a contrivance designed to be supported by a mass of air, either by the dynamic action of the air upon the aircraft surfaces (aerodynamic lift), or by its own buoyancy (aerostatic lift).

A fixed-wing aircraft is a class of aircraft that become airborne by aerodynamic lift (a force that is perpendicular to the oncoming flow direction and in opposition to weight) produced by its wings and air. The wings are static planes extending either side of the aircraft, when the aircraft travels forwards, air flows over and under the wings which are shaped to create lift.

Formally speaking, an aircraft is indeed a concrete instance of an engineering system, defined in [55] as a set of mutually related elements artificially built and assembled together in some specific order to perform an intended function (a flight mission) that can influence or be influenced by natural systems (e.g., flight environment, control systems, etc).

Most of those kinds of systems (including an aircraft) are considered as complex systems in relation to the intrinsic “complexity” of its structure (number of components, number of relationships between them, etc), and the number of functions the system performs[55] (an aircraft is a system capable of generating aerodynamic lift, carrying on payload, take-off, climb, navigate, descend, land, taxing, accelerate, glide, etc).

From system engineering perspective, an engineering system such as an aircraft is organized in top-down system hierarchy approach. This hierarchy goes from the system level, where the purpose of the system is achieved, through other functional units that are aggregates of lower level elements (e.g., subsystems, assemblies, components, sub components, etc), down to parts at the lowest level of the hierarchy, which are functional units assumed indivisible in the context of the system being considered[55].

1.1.1 Aircraft design

Aircraft design is a separate discipline of aeronautical engineering, different from the analytical disciplines such as aerodynamics, structures, controls and propulsion (an aircraft designer spends little time performing such analysis)[63].

From the process exhaustively described in [67], designing an aircraft is a multidisciplinary and iterative process that involves performing two types of engineering design activities: problem solving through mathematical calculations, and logical selection of a preferred one among alternatives (configuration selection), within cycles of three major design operations (Fig. 1.1): analysis (prediction of the performance or behavior of a design candidate), synthesis (the creative process of putting known things together into new and more useful combinations—new values, features, characteristics, or parameters are determined—) and evaluation (the process of performance calculation and comparing the predicted performance of each feasible design candidate, candidate designs that fail to satisfy the requirements are retired) across aircraft main components and the aircraft as a whole itself.

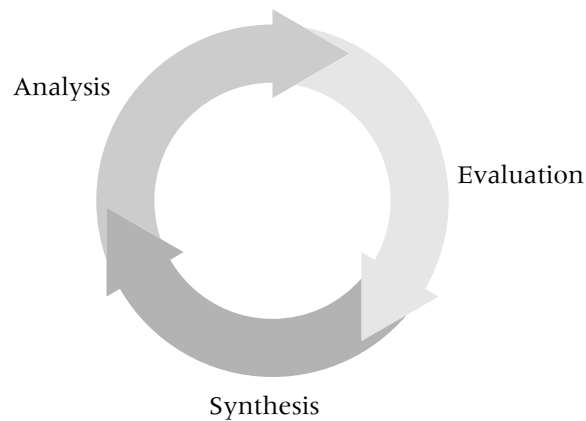


Figure 1.1: Design operations
Source: Based on [67]

The aircraft design process is the means through which an aircraft design is synthesized, starting from the requirements and finalizing to a state prior to manufacturing, and consists according to the system engineering discipline in four major design phases: conceptual design, preliminary design, detail design and test and evaluation.

Conceptual design

According to [67], it is the first and most important phase of the design process. It commences with the identification of the requirements in specific qualitative and quantitative terms.

Aircraft conceptual design is a high-level activity with potential to establish, commit, and otherwise predetermine high-level qualities, including the shape of the desired aircraft[67].

Its primary objective is the selection of a preferred aircraft configuration that meet the requirements. The outcome of this phase is a design concept or configuration which does not

necessarily accompany any details[67].

During conceptual design only components that are directly responsible for the conceptual design requirements (dominant components) are usually considered (the other kind of components are servant components that “serve” dominant components, yet those components are part of the scope of the detail design phase)[67].

Answers to questions like: Will it work?, What does it look like?, What requirements drive the design?, etc, represent a way to visualize the purpose of the aircraft conceptual design phase[63].

Preliminary design

The essential purpose of this phase is to determine features of the basic components and sub-systems.

Common output for this phase includes: major technical data, some mathematical models, materials specifications, performance technical measures at subsystem level, functional analysis at subsystem level, detail design requirements, trade-off studies at subsystem level, full system mock-up and model, operational studies, interface specifications, plans for verification and verification tests, among others[67].

Detail design

This phase establishes detailed design decisions at component and parts level, mathematical models for evaluation, low level design requirements, extensive design operations (e.g., technical/mathematical calculations), detailed trade-off analysis, integration of system, subsystems and components, physical prototype model, detailed design data, planing and conducting of tests and evaluations, among others[67].

Test and evaluation

During this phase different tests at a system integrated level are performed on a physical prototype of the aircraft (i.e., aircraft aerodynamic testing in a wind tunnel, aircraft dynamic testing, structural and propulsion test)[67].

1.2 Motivation

Engineering design is creative work. It requires ingenuity, intuition and good judgment, all talents that are typically human and not easily implemented in a computer program... [73].

According to the aircraft design process, conceptual design phase is a key step from where subsequent phases will depend on to further develop the design of the aircraft. It has the potential to determine high-level features (or at least strongly influence) including the shape of the aircraft and a complete high-level configuration are expected to be obtained from it.

Conceptual design phase is also not so firmly dependent on mathematical analysis nor mathe-

mathematical problem solving but with high-level selection of alternatives (i.e., configuration building).

On the other hand, selection from alternatives would mean that prior domain knowledge and experience became key factors for making decisions effectively —is reported from interviews with designers in the US aircraft industry that 56 % of product development delays were due to lack of documented design knowledge. This underlines the importance of reuse of ideas and engineering knowledge that is established and form part of a company’s corporate knowledge— [73].

A concept is an ‘on the way’ solution (that results from a conceptualization process, consisting in manipulation and combination of ideas), and thus, partial and intermediate in its nature, is a design proposal that is detailed enough to justify if it is a good answer to the task and intention[4].

An aircraft, including the concrete case of a fixed-wing aircraft is a very complex engineering system, but when placed at a conceptual level (i.e., aircraft configuration), most of its complexity is “postponed” to subsequent phases, that is, some levels of the system hierarchy are completely discarded, and the number of subsystems, components and their relationships are considerably reduced.

The fundamental objective of the aircraft design process is to synthesize an aircraft design that better meet the requirements stated at the beginning of the process, and in the special case of conceptual design, a design concept that mean to be a promising starting point for further design phases.

This process shares the characteristics of an optimization problem and particularly a design optimization problem (also called engineering optimization problem), where a design should be found that optimally, or near optimally meet an objective or design requirements.

From a mathematical perspective, a design optimization problem regards to find —a set of values (discrete, continuous or mixed) for a corresponding set of design variables that represent a design— that minimizes or maximizes a function that represents a measure of quality of the design with respect of the objectives to be met.

A common approach in engineering for design optimization is through mathematical methods of optimization, where an equation or equations (engineering performance) have to be minimized or maximized in order to find specific values for parameters or variables. Design optimization by mathematical methods is generally utilized in the detail phase of engineering design, where specific values (sizing) are determined for each part and component of the system.

Considering that conceptual design is about building a configuration, mainly through selection of alternatives, it becomes difficult to express the system at that level by using only mathematical expressions[59].

A particular motivation emerges to build a solution as means for conceptual design automation of an aircraft. Since conceptual design phase does not regard to any physical model of any kind of the aircraft being designed, a type of digital aircraft prototype must be the outcome from the automatic process, that is, the synthesis of the design concept.

According to systems engineering discipline, and the previously presented iterative aircraft de-

sign process, a synthesized design must be evaluated in order to discard non compliant designs, and move the process towards the optimal design. This evaluation can be driven by using existing simulation tools and particularly a flight simulation tool, given that a simulator of this type expects an integrated version of an aircraft (i.e., no separated components, subsystems or parts).

Generating a digital conceptual design of an aircraft that is flyable (to some extent) inside a flight simulator, requires some level of completeness of the design, a task that might be achievable at a conceptual level considering that, as stated by [55], the sum of the functioning of the parts is quite often not equal to the functioning of the whole, therefore the synthetic mode of thinking of the systems approach, seeks to overcome the often-observed predisposition to perfect details and ignore the system outcomes from the analytical mode thinking.

Furthermore, considering that conceptual design relies mainly on making selection decisions, an approach based on evolutionary algorithms sounds promising to enable automatic conceptual design. An evolutionary algorithm is a kind of bioinspired metaheuristic method to solve general complex optimization problems that do not have to fulfill any assumptions or limiting properties, that is, the problem is considered as a black box[44] (just like the problem of designing an aircraft, at least from the computational perspective).

Based on the concept of evolutionary design[23], a solution based on the evolutionary computing principles could provide access to a considerably bigger set of alternatives to choose from, without requiring going too deep into the mathematical characteristics of the design problem itself. Moreover, given the importance that prior domain knowledge, experience and human creativity play in conceptual design, rather than mathematical problem solving; incorporating pieces of domain knowledge into the automatic design process to improve the search of an optimal design seems convenient, besides that integration could also enable designers and the automatic conceptual design process itself, to deliberately incorporate new design ideas (either human-made or synthetic) or existing configurations.

1.3 Objectives

The general objective of this work is to synthesize a flight simulator flyable fixed-wing aircraft that fulfill specific needs and performs favorably, through an evolutionary strategy supported by domain knowledge.

The specific objectives are:

- To define and model a specific aircraft design as a test scenario, and its corresponding set of design objectives, design variables, fixed parameters and evaluation mechanisms.
- To build a knowledge base containing pieces of domain knowledge that support the evolutionary strategy and the test scenario modeling.
- To craft a custom evolutionary strategy for the aircraft conceptual design automation.
- To synthesize a flight-simulator-flyable fixed-wing aircraft as a possible solution for a specific scenario.

1.4 Disclaimer

Since this work pretends to follow a direct simulation-driven approach, it is taken for granted the computational expensiveness factor it might mean. For this reason, and besides —the purpose of incorporating pieces of domain knowledge within the evolutionary process—, a decision is made from the beginning to follow an evolutionary design optimization approach of crafting a custom evolutionary strategy for the aircraft conceptual design automation problem, instead of going through a process of —selecting, tuning and discarding— a set of existing evolutionary algorithm implementations.

Part of this thesis is the crafting of an evolutionary strategy, it should not be confused with the term “evolution strategy” nor its acronym “ES” which is a mainstream type of evolutionary algorithm. The evolutionary strategy of this work is not considered a type of “evolution strategy” nor inspired by it, but a custom strategy based on evolutionary computing principles.

1.5 Document outline

Subsequent content of this work is organized as follows:

Chapter 2

Within this chapter, a theoretical framework for the development of the proposed solution is presented. Starting from the most essential application field of this work: “design”, that, although is a very common term used in many fields of science, engineering, arts, and daily life; a specific definition must be selected to avoid ambiguities.

From that starting point, the theoretical framework is elaborated from broader concepts into a specific body of knowledge adapted to the particular needs of the problem being studied and the objectives set for this research work.

Evolutionary computing and knowledge based systems theoretical review is not performed in a survey style but from a generalization and conceptual perspective, an approach conducive to crafting a custom solution based on those fields of knowledge principles, therefore, evolutionary components and mechanisms are explored with —the mixed nature of an aircraft conceptual design solution, a mix of continuous and discrete values at the optimization solution representation level and augmented to categorical values at the initialization and system representation level— in mind.

Chapter 3

A literature review is presented with a summary at the end of that section, as well as a short description of the software tools used during the development of the solution (that were not built as part of this work).

Chapter 4

Through this chapter, the solution proposed is fully elaborated. Starting with an overall view of the followed methodology and main components involved, continuing with the definition of the

requirements of the design problem, the problem definition itself, including the rendering of the design objectives and their respective objective functions (domain specific methods are provided to construct those functions), followed by the modeling of the aircraft as an engineering system (structurally) and the knowledge regarding that system.

The design representation model is analyzed and implemented to be used by the evolutionary strategy and simulation tool. A domain model, including geometric aspects and domain specific features of the system is built to serve as a bridge between the underline design representation and the models required by the simulator and complementary views of the system (like visual or human readable format models) in terms of the application domain.

In the second part of this chapter, the evolutionary strategy is described in detail, providing a nomenclature, algorithms and diagrams, including selection operators (both target selection and offspring selection operators), mechanisms to improve population diversity, mechanisms to provide flexible alternatives for performance tuning and mechanisms to help increase probability of success.

The proposed knowledge based variation operators are defined, fully described and implemented (including the knowledge representation for their functioning) and custom evolutionary operators are presented.

At the end of the chapter four implementations of the strategy are defined and illustrated.

Since the aircraft or more specifically the fixed-wing aircraft is a particular case of an engineering system, the terms: aircraft, fixed-wing aircraft, “system” and “engineering system” are used interchangeably (for contextual purposes) to refer to the fixed-wing aircraft being designed as the implementation of the solution proposed.

Chapter 5

In this chapter, an experiment (namely main experiment) and its corresponding results are presented to test the modeling and definition of the design problem, and the evolutionary strategy proposed in this work.

The proposed strategy is tested through the execution of a battery of conceptual design processes, implemented in four variants of the strategy. The strategy is compared with a custom differential evolution algorithm and three additional special algorithmic implementations of the strategy designed to isolate both, the knowledge based variation behavior and the pure stochastic behavior of the evolutionary classical variation operators.

An additional “short or mini” experiment is included in order to test a dynamic flight environment scenario. Weather is enabled to produce dynamic perturbations on the simulation model.

Chapter 6

General and specific conclusions and considerations derived from this research work and the results obtained are exposed and explained, as well as ideas postponed to future work.

2

Theoretical framework

In the process of building a theoretical framework for this research work, some reference concepts need to be set in order to avoid ambiguity during the development of the research.

Domain, type and characteristics of the problem being studied are also required to be defined in the interest of establishing a starting point and concepts path towards a solution approach and specific methodology.

2.1 Problem domain

This work deals with aircraft conceptual design; however, design is a term that “design” has several meanings depending on the context it is used, and its grammatical role.

Given that an aircraft is a type of engineering system, a proper definition for design to be adopted in this work, could be one based on the engineering design practice, where design is considered both, the action of determining (a verb) —and the determined (noun)— parameters that enable the construction of an optimal engineering system[58] (with respect of requirements and constraints), by applying scientific and engineering knowledge.

The type of design may be characterized by its scope: a novelty, adaptive design or variant design[58].

2.2 Problem type

Aircraft conceptual design automation is a problem that perfectly fits in the definition of a design optimization problem, in the sense that the aircraft conceptual design process aims to obtain an aircraft design concept that meet the requirements (e.g., flight mission) stated at the beginning of the process.

2.2.1 Optimization

Broadly speaking, optimization means to find, within a given search space, a solution (for an optimization problem) represented by a value or a set of values for their corresponding parameters or variables that cause an objective function value to be optimal (minimal or maximal).

An objective function is a numerical mathematical expression for the objective that needs to be achieved by a problem solving method[18].

An objective function measures the “goodness” or quality of a feasible solution, understood as a parameter or set of parameters (or variables) whose values satisfy all the imposed side conditions (constraints), if any[18].

An objective function is called by different names depending on the context of use and whether the objective is a minimal or maximal, in order to provide a more meaningful sense when used in a specific problem (e.g., cost function —the least value the better performance—, profit function, quality function, fitness function, gain function, distance function, etc).

2.2.1.1 Optimization problem characterization

Based on [64], [32], an optimization problem can be classified depending on the characteristics of its objective function and its search space:

Number of optimals	An optimization problem is mono-modal if the objective function only has one optimal value, or multi-modal if has more than one. That means that in a multi-modal problem there could be several local optimal values.
Number of objectives	If the objective function has a scalar value the problem is considered a mono-objective problem, or multi-objective if the objective function value is a vector. This classification can also be observed from the number of objectives of the problem definition itself.
Constraints	If there are constraints that partition the search space, the problem becomes a constrained problem, otherwise unconstrained problem.
Search space domain	An optimization problem, depending on the search space nature, could be continuous (for continuous search space), combinatorial (for discrete search spaces), or mixed.
Optimization purpose	An optimization problem is a parametric or static optimization problem when the purpose of the optimization is to find values of a set of parameters (i.e., design variables), and control or dynamic optimization problem when the purpose is to find actions to be taken depending on the changes in the state of the problem.

Since actions could still be considered or modeled like variables or

parameters, therefore a more determining element to distinguish between parametric and dynamic optimization problems regards with the role of time in the objective function evaluation and problem behavior.

Objective function nature Per the mathematical characteristics of the objective function, an optimization problem might be linear or non-linear, differentiable (objective function gradient exists) or not differentiable, and, if differentiable, it can differentially computable or not computable, depending on whether its derivative can be calculated or not.

2.2.2 Design optimization

A design optimization problem is an optimization problem that has the purpose of finding, within a design space, a design of the system that meet some performance or design objectives. In a design optimization problem, the design fitness or quality (determined by fitness and objective functions) is the measure that tells how close or far a given design is from achieving the objective or objectives.

A design of the system from the design optimization perspective, is a set of design variables values. Once the design variables are given values, they represent a design of the system (a candidate design solution for the optimization problem). Different values for the design variables produce different designs, and the number of those variables gives the design degrees of freedom for the problem[5].

A design can be defined by different sets of design variables [5]. This characteristic allow defining different models (views or abstraction layers) of the system, that is, representations that capture or emphasize only certain properties of interest in the modeled system, while the fidelity of the model to the actual system is intentionally reduced or limited in other ways[51].

2.2.2.1 Design optimization problem formulation

The first step in the solution of a design optimization problem, is to formally formulate it (success in solving the problem is not guaranteed by properly formulating the problem; however, failure is almost guaranteed if the problem is poorly or mistakenly formulated), a common approach is the one described by [5]:

Design problem description

Translating a textual statement that describes the overall objectives and requirements of the problem into a well-defined mathematical definition.

Data/information/tools collection

Identify and collect analysis procedures and tools that will support the evaluation of candidate designs (e.g., domain specific knowledge procedures, equations, applicable theory, simulation tools, etc).

Design variables identification

An optimization problem should depend only on a single selected set of design variables, those variables should be selected according to the following considerations:

- Design variables should be independent of each other, or if dependency exists, that must be controlled by constraints.
- The minimum number of design variables to represent the design should be identified.
- Fixed values should be incorporated as potential design variables (at least at initial formulation) with assigned fixed values (equality constraints).
- A design variable should be identifiable.

Optimization criterion

Selecting a proper objective function or a set of them, to enable ranking among candidate designs based on a value of merit assigned to each of them.

An objective function to be valid, must be influenced directly or indirectly by the set of design variables, in some problems however, it is not obvious what the objective function should be, or how it should relate to the design variables, in that case, some domain knowledge and experience may be needed.

Formulation of constraints

Restrictions of any kind and performance requirements (not related to the objective function), need to be identified and mathematical expressions should be developed for them as bounds of a feasible design space (i.e., a set of designs that are acceptable, feasible or workable).

If a design does not meet at least one constraint it is considered an unfeasible or unacceptable design. These constraints must depend on the design variables (i.e., must be functions of at least one design variable). It must be considered that, if constraints are too restrictive, there may not be any feasible solution for a given problem.

2.3 Solution approach

2.3.1 Simulation-driven design

According to [44], in practice, many optimization problems behave as “black box” problems, with many knowledge gaps regarding the problem internal details (e.g., whether the objective function is linear or not, the problem is mono or multi-modal, etc). In “black box” type design optimization problems, the value of objective functions usually cannot be obtained by simply solving an equation, but with data provided by external computer-based simulation tools or solvers.

Computer simulations allow observing a system behavior and characteristics in a digital prototype before the physical system is actually built, this mechanism may enable an economically efficient computed-based design optimization process of the prototype or its components[44].

According to [43], the use of computer simulations in engineering to assist design optimization, besides being used for verification purposes, has been more commonly adopted as they have become more reliable tools, specially where traditional theoretical models might be no longer adequate to fully assist the design, or with the necessity of more accurate simulations of increasingly more complex systems.

In the direct simulation-driven design optimization approach, the candidate solutions generated by the optimization algorithm are evaluated through high fidelity simulation tools for verification purposes, and to provide the optimizer with useful information to search for better designs[43].

Simulation times may vary from seconds, to minutes, hours, days and even weeks, depending on the complexity of the system being simulated and the fidelity level. This could make their usage prohibitive for some applications, when using optimization algorithms that require numerous evaluations during their searching process.

A possible approach to tackle the computational expensiveness problem of high fidelity simulations, is the application of surrogate-based optimization (SBO), consisting in replacing the expensive high fidelity simulation model with one or more lower fidelity representations called surrogate models. The surrogate model is then used in an optimization process to identify promising areas of the design space that eventually might drive to generate high fidelity models and verify them in the computationally expensive high fidelity simulation tools[43].

At a conceptual level, the usage of direct simulation-driven design optimization might be still practical and useful. A digital aircraft conceptual design could be indeed, considered as a lower fidelity level model of the full system design model. Only a small subset of the design variables and components, and only one or a very small number of disciplines are partially covered.

From the SBO perspective, a direct simulation-driven conceptual design optimization might become in some degree the surrogate model of subsequent optimization design phases (e.g., conceptual multi-disciplinary, preliminary and detailed design) of the aircraft, or just being the high fidelity model at a conceptual level, in which case the simulation tool would be the high fidelity simulation tool at that level as well.

2.3.2 Hybrid metaheuristic approach

A metaheuristic is a method for solving general types of problems. A metaheuristic guides the process of finding high quality solutions by coordinating the application of search operators (i.e., mechanisms and procedures that perform movements within the search space) within a large solution space. A metaheuristic uses mechanisms to avoid falling into local optima to perform a robust search without requiring hypotheses on the optimization problem nor any kind of prior knowledge on the objective function, that is, the objective function is treated as a black box[30], [24], [10].

No guarantee can be provided about the quality of the solution obtained (i.e., finding the global optimum is not guaranteed), but a well-designed heuristic method usually can provide a solution that is at least, nearly optimal[41].

Metaheuristic optimization methods are generally inspired by analogies and have stochastic

nature, a feature that allow them to manage the combinatorial explosion nature of a conceptual design problem. They also have the advantage to be extensible to multi-objective, multi-modal, dynamic and parallel optimization implementations[69].

There are several types of metaheuristic algorithms and one could think of picking one to enable aircraft automatic conceptual design as design optimization method; however, according to the no-free-lunch theorem of optimization, a general “best” optimization method cannot exist to solve all kind of problems (i.e., a method that outperforms the rest of methods in a specific application), and the only way a method can outperform another (in a specific problem), is if it is specialized to the structure of the specific problem[36].

This situation leads to define a hybrid metaheuristic that, according to [62], is an algorithm that do not strictly follows a traditional metaheuristic established algorithm, but combine various algorithmic ideas in order to improve performance and capabilities of solving specific problems by synergy of concepts.

In other words, “exploiting problem-specific knowledge in the best possible ways, picking the right algorithmic components, and combining them in the most appropriate way, may lead to build the most appropriate algorithm to solve a specific optimization problem[62], [36]”.

2.3.3 Optimization problem scalarization

Scalarization is a common approach to deal with the usually, multi-objective nature of a design optimization problem without requiring extending the base optimization framework available, by virtually lower the n-dimensionality of the objective or fitness function into a single-objective, that is, replacing a multi-objective optimization problem by a suitable mono-objective optimization problem (replacement of a vector by a scalar[39]).

This approach is a type of “Preference based method”[22] of resolving multi-objective optimization problems, where assumptions are made about the importance of objectives prior the optimization process.

This approach is supported by the principle in vector optimization that optimal elements of a subset of a partially ordered linear space can be characterized as optimal solutions of certain scalar optimization problems[39].

Weighted sum method

The weighted sum method is a linear aggregating function used to compute the scalar fitness of a solution[14] using:

$$f(x) = \sum_{i=1}^k w_i f_i(x)$$

where $w_i \geq 0$ and $i = 1 \dots k$ are the weighting coefficients representing the relative importance of the k objective functions of the multi-objective problem. It is usually assumed for normalization that:

$$\sum_{i=1}^k w_i(x) = 1$$

Solving the new formulated single objective optimization problem for a certain number of different weight combinations yields a set of solutions[86].

Weighted metric method

The weighted metric method[22], [54], is another approach of combining multiples objectives into a single objective through a weighted metric such as the L_p (Minkowski) distance metric. The weighted L_p distance measure of any solution from a utopian solution u^* (a non-existing solution in terms of the values of its individual objective functions) is the norm L_p of their difference:

$$f(x) = \left(\sum_{i=1}^k w_i |f_i(x) - f_i(u^*)|^p \right)^{1/p}$$

The parameter p can take any value ≥ 1 . As of the weighted method, different executions using different sets of weights produce a set of solutions. Any optimal solution obtained by L_p depends on the parameter p , which works as a kind of resolution parameter. It has been observed that, as p parameter starts increasing from $p = 1$, the number of optimal solutions reachable with this weighted metric also increases[22], as well as the diversity of the solutions (when comparing with the weighted sum method starting at $p = 2$). It is also observed that, as p increases, the problem becomes non-differentiable, making this method useless for gradient-based methods, yet promising with evolutionary approaches.

2.3.4 Evolutionary design

Evolutionary design and particularly evolutionary engineering design, is an approach to coming up with optimal or innovative design solutions by using evolutionary computing (EC) principles through which different design concepts (in the particular context of conceptual design) can be evolved by formulating the design task as an evolutionary optimization procedure[23].

2.3.4.1 Evolutionary approach suitability

Evolutionary algorithms are becoming increasingly popular in engineering design due to the flexibility, self-adaptive properties and search capabilities, making them a suitable approach to tackle engineering design tasks, usually presenting the following characteristics:

- Objective functions are usually non-linear, non-convex, discrete, and non-differentiable.
- There can be more than one optimal solution, requiring a global optimization method.
- There can be many objectives, often conflicting with each other.
- Objective functions can be noisy, causing classical methods to get stuck in the details.
- Objective functions usually are computationally daunting to evaluate (as well as the procedures used to assist evaluation).

- Design variables are usually mixed: real-valued, discrete integers, categorical, encoded chain of events, permutations, etc.
- Number of design variables, objectives and constraints may be large, causing “curse of dimensionality” problems (as dimensionality increases, the volume of the space increases so fast that the available data become sparse).

2.3.4.2 Evolutionary approach implementation

According to [23], the advantages of the evolutionary design approach do not come free; its application must use appropriate and customized evolutionary operators specific to the problem, and develop a methodology based on customization and hybridization.

When designing the specific algorithm, as much customization as possible must be utilized in creating the initial population, and developing evolutionary operators[23].

Possible generic strategies would be[23]:

- Developing seed solutions in terms of building blocks.
- Considering some operators as a primary creation operators and others to repair existing solutions.
- Taking advantage of knowledge of one or more existing solutions, and use them as seed solutions.

2.3.5 Domain knowledge incorporation

According to [47], there are some options to incorporate domain knowledge into an evolutionary algorithm (EA), either to speedup convergence, or to reduce the computational cost of the algorithm. Prior domain knowledge may be used to build a set of references for conducting a more efficient search, to seed good solutions in the initial population, or to add knowledge to variation operators in order to facilitate the generation of good solutions.

A rule based system (RBS) is a relatively simple type of knowledge based system (KBS) (from implementation perspective). A RBS is suitable for applications of diagnosis (classification), advisory systems (let the user make the final decision), dictatorial systems (make decisions without consulting a user), systems based on experience, among others; however, a RBS can be adapted to almost any number of problems as long as the knowledge in the problem area can be modeled in the form of if-then rules or premises.[33].

2.3.5.1 Knowledge

Human knowledge is the result of cultural development and is composed of scientific discoveries, industrial advances and innovations, but also human ingenuity and experience[4].

Based on [42] analysis, knowledge is the result after understanding and internalizing collected

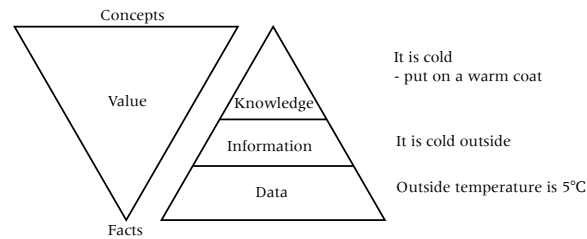


Figure 2.1: Data, information and knowledge
Source: Based on [42]

information about an area of concern (i.e., domain) for its later application, usage to support taking decisions, solving problems, and further develop new knowledge.

Information as a source of knowledge, is data processed or transformed into a form or structure suitable for its usage. Data are streams of raw facts or recorded symbols representing events. The perceived value of data increases as it is transferred into knowledge through information (Fig. 2.1).

From a computational perspective, and in the context of a KBS, knowledge is anything that can be expressed through a rule[42], and in general by any knowledge representation mechanism such as semantic nets, ontologies, custom Extensible markup language (XML), procedures (direct programming), etc.

2.3.5.2 Knowledge based system

A KBS is a computational system that reasons on continuously subject to change knowledge, to solve complex problems. It requires two types of knowledge to be functional and useful, the domain knowledge and the knowledge of how to manipulate the domain knowledge and reason on it[52].

A KBS comprises two essential components[52], [33]:

Knowledge base A computational system that captures the domain knowledge through a specific knowledge representation, that is, symbols and structures used to represent knowledge.

Inference engine A set of algorithms that enables reasoning (generate conclusions by applying logic) on the knowledge, that is, manipulation of knowledge representations to produce representations of new knowledge.

Developing a KBS requires translating the domain knowledge coming from an expert source (e.g., a human expert, a book, etc) into a set of computational representations of that knowledge.

2.3.5.3 Rule based system

A RBS (also known as production systems) is a type of KBS that uses rules as knowledge representation mechanism, a RBS is a very simple model that can be adapted and applied for a large set of types of problems if and only if, the domain knowledge can be expressed in the form of IF-THEN rules[33].

An IF-THEN rule is a statement that relates an antecedent or premise in the IF part to a consequent or conclusion in the THEN part:

IF antecedent THEN consequent

For example: IF aircraft main wing dihedral angle < 0 THEN aircraft has anhedral main wing configuration

Premises are facts (data and information) subject to conditions expressed by logical operators (conjunction \wedge , disjunction \vee , and negation \neg), whereas conclusions are new fact assertions (and some times triggered actions). The rule tests the logical expression in the premise and, if the expression evaluates to true, it then asserts that a fact about a thing or a class of things is true[33].

In the example, if the premise evaluates to true, new knowledge is generated that tells that the aircraft main wing configuration is of the type anhedral (a negative angle of the wing with respect to the lateral axis of an aircraft).

In compliance with the components of a KBS, a RBS consists of a knowledge base and an inference engine. In particular, the knowledge base in this kind of systems is composed by one or more fact bases (set of data and information assertions) and one or more rule bases (knowledge encoded as a set of rules).

An inference engine in the context of a RBS, reasons on the knowledge in two modes:

Forward chaining or data-driven approach Starts from facts entered by the user (i.e., working memory facts) and searches through the rule base to assert new facts or generate conclusions.

Actions embedded into rules and other chained rules are fired, and new facts are entered into the working fact base whenever a condition in a visited rule is true within the searching path.

The engine pass through the rule set as many times as required until no more rules can be fired (i.e. no more rules conditions are true).

Backwards chaining or goal-driven approach Starts from a hypothesis and proceeds backwards by matching provided hypothesis against existing conclusions, formulating new hypotheses from found conclusions resolved by knowledge represented as facts, either in fact bases or residing in working memory, until the original hypothesis gets confirmed or disproved.

Backwards chaining can also serve as questioning mechanism, through which a user may formulate a factual query, and the engine would determine the facts that must be asserted

to resolve the query[34].

2.4 Evolutionary computing framework

The word evolution has several meanings, however, when used in the context of an EA, it refers to the biological evolutionary process, where a population of individuals evolve (change) over time by consequence of variation and selection behavior.

Previous conceptualization only corresponds to the analogy the type of algorithm is inspired by (a metaheuristic feature), the real evolutionary behavior of an EA has to do with taking advantage of its own adaptive capability to solve difficult computational problems in a large solution space. In other words an EA is inspired by the features of biological evolution but is not constrained by them[21].

An EA is a general purpose metaheuristic population based problem solver, part of EC, an artificial intelligence (AI) family of computational models[72] inspired by biological evolution to progressively find high quality solutions for complex problems within a large search space.

2.4.1 General evolutionary algorithm model

Based on [21], [8], an EA (Fig. 2.2) solves complex problems based on the following components and properties:

Individuals

A representation (or encoding) of a possible or candidate solution of the problem, modified as necessary for a particular EA instance[14].

An individual is usually in the form of vectors[14] composed by design variables values, whose dimensionality either fixed or variable[48] is problem specific.

Many other representations are possible (e.g., strings — binary, integer, and real-valued—, trees, direct representations[65], and any other custom representation specific to a problem).

Population

The algorithm maintains a population (set) of individuals (population based nature).

Population is initialized typically at random, but it is possible to initialize a population with known starting individuals, and even by using domain-specific knowledge[72] to bias the search of a solution.

Individual fitness

Individuals have “fitness”, used by the algorithm to influ-

ence the population evolutionary cycle (i.e., generation of new solutions, retaining and discarding existing solutions).

Fitness of an individual is determined by a fitness function, a type of objective function that maps an individual to a real value called “fitness” that tells how close that individual is from achieving the problem objective.

Variational reproduction notion [10] New solutions (offspring) are generated from others (parents). Offspring or descendant solutions are generated by the application of variation operators, usually mutation or recombination.

Selection

Individuals are selected to survive, or to die (i.e., some solutions are retained and some are discarded from the population) at the end of a generation.

Individuals are selected also to be parents of other individuals.

Selection favors better individuals to survive and reproduce more often than those that are relatively worse.

Progressive evolution

EA cycle is repeated for several generations until a termination criterion is met.

By the EA biased selective nature, a steady improvement in the fitness of the population is produced (i.e., quality is optimized).

Termination criteria

A set of conditions that determine whether the algorithm should stop [26] (e.g., minimum fitness achieved, maximum number of evaluations reached, maximum number of generations, fitness improvement stagnation, etc [40]).

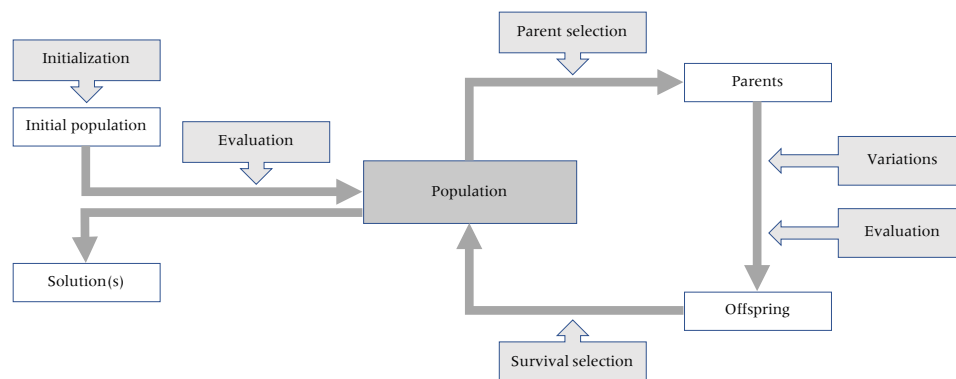


Figure 2.2: EAs general schema
Source: Based on [27]

2.4.2 Evolutionary algorithm components and properties

A general framework of an EA based on [8], is defined as:

- A I set of individuals $a \in I$.
- Parent population size μ , and the representation of a population at generation g as $P(g) = \{a_1(g), \dots, a_\mu(g)\} \in I^\mu$.
- An initializer or initialization procedure ζ that generates population $P(1)$.
- Fitness function $F: I \rightarrow \mathbb{R}$.
- An evaluator or evaluation procedure ϵ that determines (or helps the process of determining) the fitness values of the individuals.

Fitness evaluation may be as simple as computing a mathematical function or as complex as running an elaborate simulation[72].

- Recombination offspring size μ' , and the representation of recombination offspring population at generation g as $P'(g) = \{a_1(g), \dots, a_\mu(g)\} \in I^{\mu'}$.
- Mutation offspring size μ'' , and the representation of mutation offspring population at generation g as $P''(g) = \{a_1(g), \dots, a_\mu(g)\} \in I^{\mu''}$.
- Since EAs are frequently used on problems with unknown fitness landscapes, the size of population and offspring are generally chosen via some preliminary experimentation[21].
- By keeping separated parent and offspring population sizes either, recombination or mutation might be absent in the evolutionary loop if the particular EA requires it (e.g., $\mu = \mu'$ would mean absence of recombination whether $\mu' = \mu''$ would mean absence of mutation).
- Recombination (recombinator), mutation (mutator) and selection (selector) are described as operators r , m , and s , defined as population transformations $r: I^\mu \rightarrow I^{\mu'}$, $m: I^{\mu'} \rightarrow I^{\mu''}$, and $s: I^{\mu''} \rightarrow I^\mu$.

The generalized approach of presenting these operators allow covering different instances of EA.

- Recombination or mutation operator (variation operators in general), and also a selection operator can be generalized further by defining r , m and s as multiple application of operators r' , m' and s' .

They can also be reduced to the level of single individuals for even more flexibility.

- A termination criteria C described as a termination criterion $\{c \rightarrow \{\text{true}, \text{false}\}\}$ set.
- Sets Ξ_ζ , Ξ_ϵ , Ξ_s , Ξ_r , Ξ_m and Ξ_C represent additional parameter sets, often required by initializers, evaluators, selectors, variation operators, and termination criteria; that are characteristic for the function or operator, the particular instance of EA and the representation

of individuals.

2.4.3 Evolutionary algorithm generalized algorithm

From the general model, components and properties, a generalized evolutionary algorithm based on [8], can be defined as follows:

Algorithm 2.1: Generalized Evolutionary algorithm (EA) based on [8]

Input: $\mu, \mu', \mu'', \zeta, C, \epsilon, r, m, s, \Xi_\zeta, \Xi_\epsilon, \Xi_s, \Xi_r, \Xi_m, \Xi_C$
Output: a^* , the fittest individual found, or P^* , the best population found.

```

begin
   $g \leftarrow 1$  // First generation
   $P(g) \leftarrow \zeta(\mu, \Xi_\zeta)$  // Initialization
   $F(g) \leftarrow \epsilon(P(g), \mu, \Xi_\epsilon)$  // Evaluation of initial population
  // while termination criteria not met
  while  $c(P(g), \Xi_C) \neq \text{true} \forall c \in C$  do
     $P'(g) \leftarrow r(P(g), \mu', \Xi_r)$  // Recombination
     $P''(g) \leftarrow m(P'(g), \mu'', \Xi_m)$  // Mutation
     $F(g) \leftarrow \epsilon(P''(g), \mu'', \Xi_\epsilon)$  // Evaluation
     $P'(g+1) \leftarrow s(P''(g), F(g), \mu, \Xi_s)$  // Selection
     $g \leftarrow g + 1$  // Generation incremental
  end
end

```

2.4.4 Evolutionary algorithm instantiation

Mainstream instances of EAs use previously presented model and framework in different combinations and approaches, as well as any other hybrid or custom EA[8].

When instantiating a particular EA for a particular class of problems, some decisions have to be taken[21]:

Solution representation

How are the individuals represented in the population?

Population size

How big should the population be?

Initialization method

How should the initial population be generated?

Evaluation method

How are the individuals evaluated?

Target selection method

How are target individuals (parents) selected?

Variation operators

How are offspring produced from parents?, and, how many descendants should be generated?

Selection method

How are the survivors chosen?

Termination criteria

Under which conditions the algorithm should stop the evolutionary cycle?

2.4.5 Mainstream evolutionary algorithms instances

- Genetic algorithms (GAs) Introduced by John Henry Holland in 1975, rely heavily on recombination and keep mutation as a background operator, they use a binary representation of individuals (genes) mostly influenced by genetics analogy.
- Evolution strategies (ESs) Proposed by Rechenberg in 1973, rely on both mutation and recombination as essential operators for searching. Strategy parameters are usually part of the individual representation, parent and offspring population sizes are usually not the same and utilize deterministic selection.
- Evolutionary programming (EP) Developed by Fogel in 1962, emphasizes mutation and does not use recombination at all, selection is probabilistic and also extends optimization of parameters as ESs do.

2.4.6 Evolutionary operators

Evolutionary operators are the means used by an EA to bias the evolutionary cycle[21]. From a generic optimization perspective, they are a type of search operators acting within the search space (shaping the topological structure of the search space[26]) towards a solution to a problem.

From the EC perspective they are the mechanisms to achieve both, population diversity and survival-of-the-fittest biological evolution analogies.

Evolutionary operators are responsible of the exploratory and exploitative behaviors of an EA. Exploration is the process of visiting entirely new regions of a search space, whereas exploitation is the process of visiting those regions of a search space within the neighborhood of previously visited individuals[15].

Those types of behavior should be kept in balance in EAs. It is generally accepted that selection operators are the source of pure exploitation; and variation operators, specifically mutation and recombination account for pure exploration effects. However, it still persists a debate whether a variation operator may be the source of exploitation[25].

There are hypotheses proposing that a variation operator might be considered to display exploitation behavior when using information to variate individuals, either to exploit promising regions of the search space or to avoid unpromising areas[25].

Another possibility of recognizing some exploitation behavior in a variation operator could be in the case of individuals generation by the action of a variation operator without disrupting the parents building-blocks[25].

2.4.6.1 Variation operators

Variation operators create new individuals from existing ones by modifying one or more variables values.

Variation operators in EAs are divided in two types: mutation (mutator) and recombination (recombinator) variation operators (although some particular EAs instances present variation operators that combine both recombination and mutation characteristics into a single operator[61]).

The difference between the two basic types of variation operators lies in the way both types of operators perform those changes, their arity[26] (the number of “target” individuals used by a particular operator to generate a new individual), their exploratory power, and their survival/-construction effects due to disruption[72].

Disruption can be observed by comparing building blocks[31] (also originally referred as schemata or hyperplanes[37]), before and after applying a variation operator.

A building block is disrupted if the offspring generated after the application of a variation operator does not belong to that building block. If, after the building block is disrupted it still exists, it is said to be a survival of the operator application. Another possibility would be that, after disruption, a new building block of higher order is created, in which case the disruption is said to display a constructive effect[72].

A building block within the evolutionary algorithm search space or within a generational population, is a conceptual structure that might be more easily visualized like a multi-level overlapping clustering of individuals within that space or population, that is, given a search space or population, one could build overlapped clusters of individuals as follows: a cluster might group individuals with the same value at variable x_1 , another cluster might be conformed by individuals with the same value at variables x_2 , another one, sharing values at variables x_1 and x_2 simultaneously, and so on.

One could build clusters by exhausting almost all possibilities the search space provides to a point just before building a cluster that would cover the entire search space.

The important concept regarding those building blocks is that they actually represent building blocks of solutions for a particular problem and, depending on the dimensionality of the building block, there could be low-order building blocks (low dimensionality) and high-order building blocks (high dimensionality)[77].

2.4.6.1.1 Recombination variation operator (recombinator)

EA recombination is an n-ary variation operator (usually binary)[26] that generates a new individual also called child or recombinant, by exchanging information between two or more

target existing individuals (usually called parents).

It is of stochastic nature in the sense that the decisions of what parts of each parent are combined and how they are combined depend on randomness[26].

Recombination is sometimes called “crossover”, particularly in GAs, derived from the analogy inspired on genetic crossover, but also because there is a slightly difference in meaning. Strictly speaking, recombination is the mixing of variables and crossover is the mixing of the values of variables; however, from a mathematical point of view that distinction is unnecessary[57].

The main principle of a recombinator (that comes from biological evolution inspiration), is that a new individual created by combining features of other individuals having desirable features, will likely result in an also desirable combination of those features; however, despite the fact this is often true in nature, it does not work that way often in EC; recombination does not guarantee not generating unimproved or even worse individuals (indeed most of recombinants result in those cases), although some improved ones are still expected[26].

Uniform recombination Uniform recombination[74], also called uniform crossover or discrete recombination, is a binary recombination operator that generates a recombinant by exchanging information between two parents.

This method of recombination can be applied to any form of individual representation (making it suitable as a custom algorithm starting point) and its mechanism of operation consists in stochastically decide, for each variable j of a pair of individuals A and B , whether a variable of A or B pass down to the recombinant C :

$$C_j = \begin{cases} A_j & \text{if } \text{rand}([0, 1]) \leq R_p \\ B_j & \text{otherwise} \end{cases} \quad \forall j$$

Every choice is performed by a generated uniform (usually) random number $r \in [0, 1]$, and a given R_p mutation probability parameter also within the interval $[0, 1]$. If $r \leq R_p$, the child inherits A variable value, or otherwise, from B .

If $R_p = 1$ the resulting new individual will be an A clone, or a B clone on the opposite case $R_p = 0$. $R_p = 1/2$ may have the most disruptive and constructive effect (when consistently applied within a population)[72], that is, the parameter R_p actually works as a recombination rate by setting the number of variables that, in average, are inherited from one or the other parent.

Uniform recombination in general has a stronger effect in both, disruption and construction that the effect observed in the n -point recombination operator used in GAs. Its construction effect does not appear to change as the population homogeneity changes, but is only affected by the R_p value[72].

Uniform recombination also shows higher exploratory power than n -point recombination[72].

Differential arithmetic recombination Differential arithmetic recombination is a 3-ary variation operator derived from the “either-or”[60] variant of differential evolution (DE) algorithm (developed to enhance recombination characteristics of the DE combined mutation/recombination operator), where a recombinant is generated as the result of an arithmetic linear combination of three parents:

$$u = x_{r_1} + K(x_{r_2} + x_{r_3} - 2x_{r_1})$$

where K is a scale factor for the arithmetic recombination, usually calculated fixed at $K = 0.5(F + 1)$, where $F \in [0, 1]$ is a positive real number scale factor (coming from the original combined mutation/recombination operator in DE, to avoid increasing the number of parameters of classic DE), and $x_{r_1} \neq x_{r_2} \neq x_{r_3}$ are randomly selected population member indices.

2.4.6.1.2 Mutation variation operator (mutator)

EA mutation is a usually unary variation operator[26] that generates a new individual or “mutant” by making perturbations in one or more variables from the base of a target existing individual (also called parent).

It is always of an stochastic unbiased nature in the sense that the changes the operator performs rely only and exclusively on random choices[26].

Uniform mutation Uniform mutation[26] is a mutation operator that creates a mutant M from a target A by considering each j variable separately, that is, it replaces each variable value with a uniform distributed random value within that variable lower L_j and upper U_j bounds. Each replacement is usually performed with positionwise mutation probability Mp :

$$M_j = \begin{cases} \text{rand}([L_j, U_j]) & \text{if } \text{rand}([0, 1]) \leq Mp \\ A_j & \text{otherwise} \end{cases} \quad \forall j$$

Uniform mutation can be used with any type of representation as long as a suitable random values generation function is available for that representation.

Differential mutation Differential mutation is a 3-ary variation operator originally proposed by the DE algorithm. The operator creates a mutant v from a base target individual x_{r_1} by adding to each j variable of x_{r_1} a scaled mutation step, calculated as the difference between x_{r_2} and x_{r_3} additional selected individuals:

$$v = x_{r_1} + F \cdot (x_{r_2} - x_{r_3})$$

where the scale factor $F \in [0, 1]$ is a positive real number that controls the mutation step size and as a consequence influences the convergence speed[60], and $x_{r_1} \neq x_{r_2} \neq x_{r_3}$ are randomly selected population member indices.

Differential mutation is originally designed to be used with real-valued representations, although it can also be used with discrete integer-valued representations through rounding techniques such as uniform quantization, a function that transforms a continuous range of values into a set of evenly spaced values, like integers[61]:

$$Q(y) = \frac{\text{floor}(k \cdot y)}{k}$$

where the *floor* function returns the integer part of its argument, when $k = 1$, the function returns the integer part of y .

2.4.6.1.3 Combined variation operators

A variation operator might combine both, recombination and mutation behaviors into a single operator definition, for example, there is the particular case of the differential mutation/recombination operator defined in classical DE algorithm and some of its derived variations.

Behaviors can be combined by simply algorithmic implementation convenience, but also to randomize the behavior of a variation operator, that is, either a variation operator may be generating a mutant or a recombinant, depending on parameters.

Differential mutation/recombination Although differential mutation and recombination operators are described separately, they are usually combined into a single differential mutation/recombination operator as follows[61]:

$$x'_{i,j} = \begin{cases} x_{r1,j} + F \cdot (x_{r2,j} - x_{r3,j}) & \text{if } (rand_j(0,1) \leq Cr \vee j = j_{rand}) \\ x_{i,j} & \text{otherwise} \end{cases}$$

where F is the scale parameter, j is the variable index, the recombination probability $Cr \in [0, 1]$ is a user defined value that controls the fraction of parameter values that are copied from the mutant, $rand_j(0,1)$ is the output of a uniform random number generator that if less than or equal to Cr , the candidate solution parameter is inherited from the mutant or, otherwise, from the original population member; and j_{rand} is a randomly chosen parameter index that is taken from the mutant to ensure that the candidate solution does not duplicate x_i .

“Either-or” differential mutation/recombination In the same way, a combined operator (differential mutation/differential arithmetic recombination) defined in the “either-or”[60] variant of DE works as follows:

$$u_i = x_{r1} + \begin{cases} F \cdot (x_{r2,j} - x_{r3,j}) & \text{if } rand_i(0,1) \leq Pf \\ K(x_{r2} + x_{r3} - 2x_{r1}) & \text{otherwise} \end{cases}$$

where Pf is the probability of the algorithm of deciding whether a given solution will compete against its mutant or its recombinant when generational selection comes.

2.4.6.1.4 Mutation and recombination effects

Mutation effects As intuitively suggested, the survival effect of mutation operators is high in general when mutation level is low, and low (more disruption) when mutation level is high[72].

Although more disruptive recombination operators achieve higher levels of construction, this is not observed in mutation operators, despite the fact that high levels of mutation are the most disruptive, they also achieve the worst levels of construction, making a significant difference between mutation and recombination operators[72].

Recombination effects Disruptive effect of any recombination operator is dramatically affected by the homogeneity of the population, disruptive effect is higher on high-order building blocks and become less disruptive as the population converges[72].

Although survival effect is desirable in a recombination operator to some extent, constructive capability to build higher-order building blocks seems more valuable and, in fact it has been showed that the more disrupted nature of a recombination operator more likely its constructive effect[74].

Interestingly, statistical studies have proved the applicability of the no-free-lunch theorem with respect to the survival and constructive effects of disruption in recombination operators. When averaged, all recombination operators are equivalent, any loss in disruption (gain in survival) is offset by a loss in construction [72].

Mutation vs recombination Exhaustive theoretical and experimental analysis of disruption effect in [72] indicates that, in general, mutation is able to cause any level of disruption as low as the lowest levels of recombination and up to the highest levels, even higher than recombination, yet with the poorest effect in construction, that is, mutation can be in general, more powerful than recombination from a survival point of view.

On the other hand, in general, recombination is more capable than mutation to produce constructive effects, a feature that can be translated into performance advantage of an EA, when the constructed higher-order building block is of higher quality than the parents lower-order building blocks[72].

In a case where using uniform recombination and uniform mutation with probabilities Pr and Pm respectively, it is observed that starting from an absence of disruption scenario $Pr = 0$ and $Pm = 0$, that is, mutation and recombination are turned off, if recombination is turned on and Pr starts increasing, disruption will start increasing also, as well as construction effect until a maximum point at $Pr = 1/2$, then, if mutation is turned on and starts to increase, construction will start to decrease until a point $Pm = 1$, where construction will be absent[72].

An even more interesting result from [72] analysis is that both, uniform recombination and uniform mutation are identical from both, survival and constructive aspects when binary strings are used as representation mechanism and the population diversity is maximum (i.e., every variable of every member in a population is different from each other), an unlikely situation in an EA.

From exploratory power perspective, mutation operators have much higher power than recombination, specially when population starts to converge[72].

2.4.6.2 Selection operators (selectors)

A selection operator is used whether a set of parents or target individuals must be selected to generate offspring or to apply a variation operator; or when selecting the next generation (survivors) members at the end of a generation cycle (also called replacement operator).

Selection plays a major role in evolutionary algorithms since it determines the direction of

search. The term selection pressure is used to denote the strength of selection to influence the directedness of the search. High selection pressure emphasizes exploitation of information gained so far and leads to high convergence speed, on the contrary, low selection pressure allows diversity and exploratory search behavior, leading to good convergence reliability[85].

Fitness proportional selection

Fitness proportional selection (FPS)[37] is a type of selection operator that sets the probability that an individual x_i is selected among a set of n individuals, proportionally to its fitness (quality) with respect of the accumulated fitness values of the rest of the $n - 1$ individuals:

$$Ps(x_i) = \frac{f(x_i)}{\sum_{j=1}^n f(x_j)}$$

where f is the fitness or objective cost type function; n probabilities are arranged into the sequence:

$$\langle Ps(x_k) : k \in \mathbb{Z}^+, 1 \leq k \leq n, Ps(x_k) \leq Ps(x_{k+1}) \forall k < n \rangle$$

Selection occurs (Fig. 2.3) by generating a uniform distributed random number r within the interval $[0, 1]$, a particular individual is finally selected by the function:

$$s(r) = \begin{cases} x_k & \text{if } x \leq Ps(x_k) \\ x_{k+1} & \text{if } x \leq Ps(x_k) + Ps(x_{k+1}) \\ \dots & \dots \\ x_{k+(n-1)} & \text{if } x \leq \sum_{l=k}^{n-1} Ps(x_l) \end{cases}$$

FPS method requires repeating the procedure as many times as required to generate a desired subset of m designs.

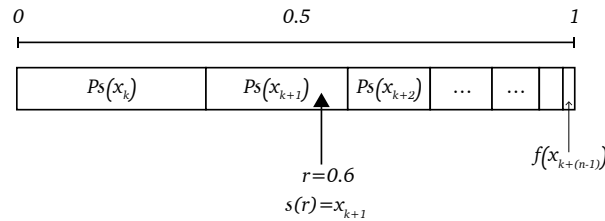


Figure 2.3: Fitness proportional selection (FPS)

Stochastic universal sampling

Stochastic universal sampling (SUS)[9] is a selection operator that reduces the spread (the range of possible actual values, given a probability)[56] observed in FPS, by calculating $m - 1$ evenly spaced intervals to select m designs at once.

Individual probabilities are calculated and arranged by using the same procedure than FPS, next, a uniform distributed random number r is generated within the interval $[0, d]$, where $d = \frac{1}{m}$ is the allocation distance. A sequence of m pointers p_o is generated:

$$\langle p_o : p_o = r + (o - 1)d, \forall o \in \mathbb{Z}^+, 0 \leq o \leq (m - 1) \rangle$$

Finally, selection (Fig. 2.4) of m individuals is performed by using the function $s(p_o)$ for each pointer.

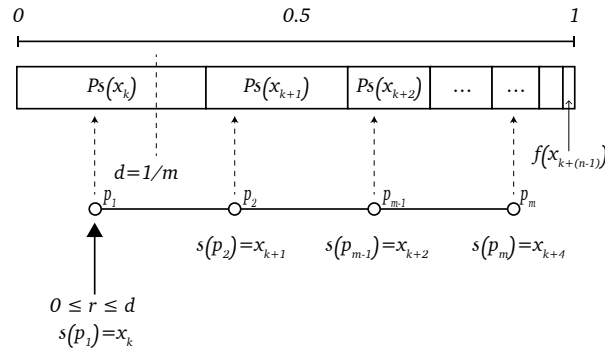


Figure 2.4: Stochastic universal sampling (SUS)

2.4.7 Extension points

There are some techniques out there that permit customizing an EA by modifying the default behavior of its operators.

2.4.7.1 Parameters as functions

Most EA parameters are initially defined to be constant, that is, the parameters are chosen and then fixed for an evolution execution.

It is observed (although none formal definition has been found) that many of these parameters could be defined as functions instead of fixed parameters, for example, it is common to find references where researchers propose randomizing some parameters[84] (specially rate, probabilities, velocity and scale parameters), or whenever an algorithm is said to be “adaptive” or “variable” in some aspect it could mean that some parameters behave as functions instead of fixed values.

Converting a fixed parameter definition into a function requires the definition of new parameters, for example, as pointed out in [61], randomizing a parameter involves selecting a probability density function (PDF) and defining an interval for that parameter.

It could be sound disadvantageous to increase the number of parameters of an algorithm but it could also provide gains in performance in particular problems or situations.

2.4.7.2 Variation rates sampling frequency

Most mutation and recombination operators provide rate, probability or scale parameters, used to control the magnitude of the effect of the variation operator within a given population in the EA cycle.

Although particular EA instances usually dictates the precise moment where a random number is generated (sampling), either to be compared against the rate or probability parameter to decide if a given operation should be applied or not (e.g., probability of mutation or probability of recombination), or to compute an actual scale factor (e.g., scale factor in differential arithmetic recombination or differential mutation); it is possible to modify default behaviors by parameterizing the sampling frequency (The idea was proposed by [61] as PDF sampling frequency and inspired by [84] in the context of DE tuning).

This parameter would offer two behavioral options:

Dither or dithering when sampling anew value for each individual.

Jitter or jitternig[84] as the practice of sampling for every variable of an individual.

2.4.7.3 Selection auxiliary mechanisms

The following are selection mechanisms rather than proper selection operators and may be incorporated to any selection operator to modify the selection pressure behavior, or as the case of the solution proposed in this work, as part of custom variation operators design and implementation.

Ranking

Is a mechanism that literally arrange a set of n individuals x_i from best to worst (or backwards) according to the sequence:

$$\langle x_o : o \in \mathbb{Z}^+, 1 \leq o \leq n, f(x_o) \leq f(x_{o+1}) \forall o < n \rangle$$

where f is a cost type objective function.

Once the ranking is available, a direct selection based on the ranking could be performed.

Furthermore, a proportional selection method like FPS or SUS might be used to select individuals by using the ranking order $o - 1$ for each individual as its “fitness” value instead of the actual fitness. This approach is known as ranking selection method[56].

Elitism

Any selection method implementation may use a mechanism of elitism, a condition first introduced in[20] that forces a selection operator to directly select specific individuals.

This mechanism is useful mainly to retain the fittest individual (or n fittest individuals).

Related work

3.1 Literature review

Evolutionary aircraft design optimization

Evolutionary computing is known to be first applied as approach to tackle aircraft conceptual design optimization problems since (Crispin, 1994 [16]). In that work the author proposed applying simulated evolution (i.e., genetic algorithm (GA)) to optimize the design of an aircraft at a conceptual level.

The optimization problem consisted in minimizing total weight of an aircraft considering 8 design variables, 6 for main wing and 2 for fuselage (other components remained fixed), 4 constraints, and initial population of 25 members. The objective function, that is, the weight of the aircraft, was calculated by accumulating the estimated weight of the wing, fuselage, engines, fuel and payload for each mission segment.

(Singh, Sharma, and Vaibhav, 2016 [71]) retakes the problem of minimizing total weight of an aircraft with a GA, an approach very similar to the first known work, now considering 9 design variables and 15 constants, optimizing a population of 100 individuals after 200 generations.

More recently (Iemma, Vitagliano, and Centracchio, 2017 [38]), took a financial approach for aircraft conceptual design by incorporating financial models of commercial aircraft extended to the long-term investments of a group of stakeholders.

Optimization per se was performed by a multi-objective genetic algorithm and a particle swarm optimization (PSO), another nature inspired metaheuristic.

Their model consisted in 5 design variables of the main wing (other components remained fixed) and two experiments, one experiment considering two minimization objectives: aircraft price and negative cashflow, and other experiment to minimize fuel consumption and noise level. They optimize the design after 1000 generations/iterations with a population/swarm size of 50 (for each experiment).

Domain knowledge incorporation

(Liu, Sun, Bai, *et al.*, 2007 [50]) incorporated domain knowledge into the optimization process of a military conceptual aircraft design through a simulated annealing algorithm (a kind of metaheuristic).

The authors implemented a single design principle that establishes guidelines to optimize edges present in the aircraft external structure that would affect the aircraft “invisibility” capabilities, a critical feature for a military aircraft survivability.

(Alonso, LeGresley, and Pereyra, 2009 [3]) proposed a method to optimize a supersonic aircraft design while maximizing its range and minimizing its sonic boom, by the use of surrogate models.

They executed a multi-objective GA for 1000 generations with initial population size of 64. The particular characteristics of this work reside in the method used to generate the initial population for the optimizer.

They built each individual through different processes and tools. First, they generated 300 candidate solutions evaluated by an aerodynamic solver and processed by a geometry builder tool, both external tools, resulting in designs represented by 8 variables. After that, 10 neural networks with one hidden layer were used to generate corresponding sonic boom effect models (1 variable per neural network).

At the end, the candidate solutions were composed by the full set of design variables plus other fixed parameters from which 64 randomly distributed candidate solutions were selected as initial population for the optimization process.

(Yang, Liu, Wang, *et al.*, 2013 [83]) conducted a very interesting research that, although it is not related with aircraft design, nor design optimization, it is about reusing design knowledge to promote innovative design.

The authors of this research developed a knowledge based system containing an ontological model of product design knowledge (custom furniture design) with implicit and explicit knowledge for some specific design cases.

They attached a user interface with the intention to test it with a group of designers. The interface and the model behind included features that would help the designers to reconstruct design cases and to propose new design alternatives.

An experiment was conducted with 4 groups of designers, 2 groups were allowed to use the system while the others were not. The designers who used the system produced in average about 40 of 60 effective designs that were classified as derivative work from the existing design cases, and about 20 effective designs resulted with no correlation with the base designs but high degree of innovation.

Another finding was that in cases where specific designers produced less effective, non correlated designs, were younger designers with less experience, and the designers who were not assisted by the knowledge based system produced fewer designs than the ones assisted.

Finally, another key finding was the fact that some expert designers complained about the sys-

tem only provided good examples as base cases; they claimed that bad examples are also a good way to learn and get experience.

This work results suggest that domain knowledge and experience might improve the design process and increase the level of innovation to some extent, something that could be extrapolated to an artificial design agent towards achieving exploration advantages and increasing probability of success.

(Li, Zhang, Tang, *et al.*, 2014 [49]) presented a reuse knowledge system for conceptual design as means to enable rapid evolutionary conceptual design.

They proposed a structure composed by three components of knowledge: mechanism knowledge (general formula, procedures, empirical knowledge, professional knowledge, etc), case library (design parameters, design templates, etc) and evaluation knowledge (government standards, economic conditions, technology conditions, etc).

They claimed that knowledge is not static but dynamic, and new knowledge is created by evolution.

Although this work comes out with a software to be used by designers (conceptual designers), the concepts behind seem promising as a way to enhance automatic design by incorporating some knowledge based guidance.

(Byrne, Cardiff, Brabazon, *et al.*, 2014 [11]) introduced the usage of a multi-objective GA algorithm as means to explore possible better aircraft designs based on existing designs.

Their strategy was to generate a parametric model from the genetic bit string representation of the solutions to build a geometric model of an aircraft and then evaluate its aerodynamic lift and drag measurements through an external solver.

Their solution was able to find possible improvements to existing aircraft by optimizing initial populations of 50 members through 50 generations, each member composed by a set of 7 design variables, 2 for airfoil selection and 5 for the main wing.

(De Gaspari and Ricci, 2015 [19]) presented a method to optimize the wing of a reference aircraft (existing aircraft) by incorporating domain knowledge into a GA loop.

Specialized knowledge rules determine re-shaping of the wing (introduced as parametric representations) that is generated in three dimensional (3D) space by a coupled computer-aided design (CAD) module, to output an aerodynamic and geometric model that subsequently is evaluated with the help of a solver.

Direct simulation-driven design

(Agte, Borer, and Weck, 2010 [2]) was found to be the first and only work until now (and to the knowledge of this research) making use of the flight simulator approach as an engineering analysis tool.

The purpose of that work was to analyze the behavior of an aircraft when facing failures with respect to some design variables being perturbed within some intervals.

The decision to use a flight simulator in that work was, according to its authors to facilitate a global approach for concurrent analysis of the aircraft performance and availability.

Summary

There is diversity among research works regarding evolutionary aircraft conceptual design optimization: most of them (including some works not included in this chapter) focus on minimizing the total weight of the aircraft by using weight decomposition estimation procedures.

Most of the works concentrate in main wing variables, then some fuselage variables, and almost none consider stabilizers into account, although some of them include propulsion and fuel consumption, also computed by estimation procedures and, at a conceptual level (propulsion as a component rather than an aggregate of parts and subsystems).

Different sets of design variables are selected, indeed in some cases there are sets of variables of higher level than most common low level design variables, for example, one approach might consider taper ratio, wing planform area and sweep angle to determine the basic geometry of a wing, while other may describe wings by other variables such as chord at root, chord at tip and span of the wing; a situation explained perhaps by the existing coupling between the variables and the estimation procedures used to compute objective functions.

No work has been found maintaining a very small population of design concepts (3 to 5) within the evolutionary process.

This review includes some works displaying novelty in either their methods of construction of their representations, domain knowledge assistance, model surrogate strategies, etc; a possible indicator of the pertinence of the custom approach adopted for this work, suggesting that it hardly exists a method of solution, even a solution approach applicable to any kind of problems without requiring customization and hybridization of methods and composition of related tools.

3.2 Software tools and frameworks

The following software tools are used within the development of the solution proposed in this thesis:

Flight simulator

X-Plane flight simulator is used as a black-box simulation tool to support the evaluation of generated design concepts.

X-Plane is a multi-platform desktop flight simulator being developed by the company Laminar Research since the second half of the nineties. This software is currently available in its last version 11 (although consumer version 10.50r3 is used in this work) in both, a consumer desktop version and a Federal Aviation Administration (FAA) certifiable professional version.

According to [82] this simulator differentiates itself from others of its kind by implementing a flight physics model called “blade-element theory”.

Traditionally, flight simulators emulate the real-world performance of an aircraft by using em-

pirical data in predefined lookup tables to determine aerodynamic forces such as lift and drag, which vary with different flight conditions.

Although by this approach the flight characteristics of an aircraft are sufficiently simulated for those with known aerodynamic data (known aircrafts), this become useless in design work since do not predict the performance of an aircraft when the actual figures are not available.

With blade-element theory, a surface (e.g., wing) may be made up of many sections (1 to 4 is typical), and each section is further divided into as many as 10 separate subsections. After that, the lift and drag of each section are calculated, and the resulting effect is applied to the whole aircraft. When this process is applied to each component, the simulated aircraft will fly similar to its real-life counterpart.

This approach allows users to design aircraft quickly and easily, as the simulator engine immediately illustrates how an aircraft with a given design might perform in the real world. X-Plane can model fairly complex aircraft designs, including helicopters, rockets, rotorcraft, and tilt-rotor craft.

The simulator also provides a plugin interface that can be used to extend the simulator with external modules, flight model modifications or new features. X-Plane also ships with other software companion tools to build and customize aircrafts (PlaneMaker), airfoils(AirfoilMaker) and sceneries.

According to the development company, X-Plane is used by world-leading defense contractors, air forces, aircraft manufacturers, and even space agencies for applications ranging from flight training to concept design and flight testing. X-Plane has received certification from the FAA for use in logging hours towards flight experience and ratings[46].

Airfoil analysis

JAVAFOIL is used in this work to build parametric simulation models of various airfoils. It is an airfoil analysis tool developed in 2001 (last version 2.22, updated in 2014) by Dr. Martin Hepperle, a German researcher specialized in airfoil and propeller design and analysis[35].

JAVAFOIL was derived from CalcFoil (also from his authorship) and has the purpose of determining lift, drag and moment characteristics of airfoils for each angle of attack and specific Reynolds number[35].

The tool supports NACA 4-digit, modified 4-digit, 5-digit, 16-type, 6 and 6-A airfoil-series, as well as other 15 non-standard types of airfoils[35].

JAVAFOIL provides a scripting interface that allow automating the execution of simulations in batch and several file formats for exporting, including the .afl airfoil parametric simulation model format used by X-Plane flight simulator[35].

Details regarding the scientific and engineering knowledge behind this tool are available in the reference source[35].

PyKE

Rule bases and fact bases used in this work are implemented in Pyke [28], [29], a framework that provides a knowledge engine that can do both forward-chaining (data driven) and backward-chaining (goal directed) inferencing.

The knowledge base (KB) engine supports: multiple fact bases, each with its own list of facts, both forward-chaining rules and backward-chaining rules, multiple rule bases, each with its own list of forward-chaining and/or backward-chaining rules[28].

Potential Pyke applications are: complicated decision-making applications, diagnosis systems, back-end (code generation and optimization) of compilers, etc[29].

4

Methodology and solution development

This thesis proposes a solution composed by a knowledge based evolutionary strategy and a system definition and modeling framework that may enable a support tool for automatic conceptual design of an aircraft. (Fig. 4.1).

Within this work, an aircraft is considered a particular instance of an engineering system, therefore the definition and modeling aspects of the solution aim to support the characteristic complexity that an engineering system may entails, and to work as a mechanism to drive the design process aligned with design requirements (i.e., requirements-driven design).

Definition and modeling aspects are indeed highly inspired by system engineering practices (the reference aircraft design process is actually systems engineering oriented), however, they are not a full or strict system engineering implementation.

Conceptual design automation per se is performed by a custom evolutionary strategy supported by domain knowledge related to the system being designed (the aircraft).

This combination is key aspect of the work developed in this thesis as means to increase probability of success, but also to tackle the trade-off between having simulation tools available that may test complex systems defined at conceptual levels and the simulation computational cost associated.

This synergy also provides collateral benefits as a conceptual design support tool, enabling human designers and researchers for studying impact on design of both, particular design ideas and possible design failure solutions.

A RBS is used as a mechanism to manage every aspect related to knowledge as either facts or rules and the reasoning on them.

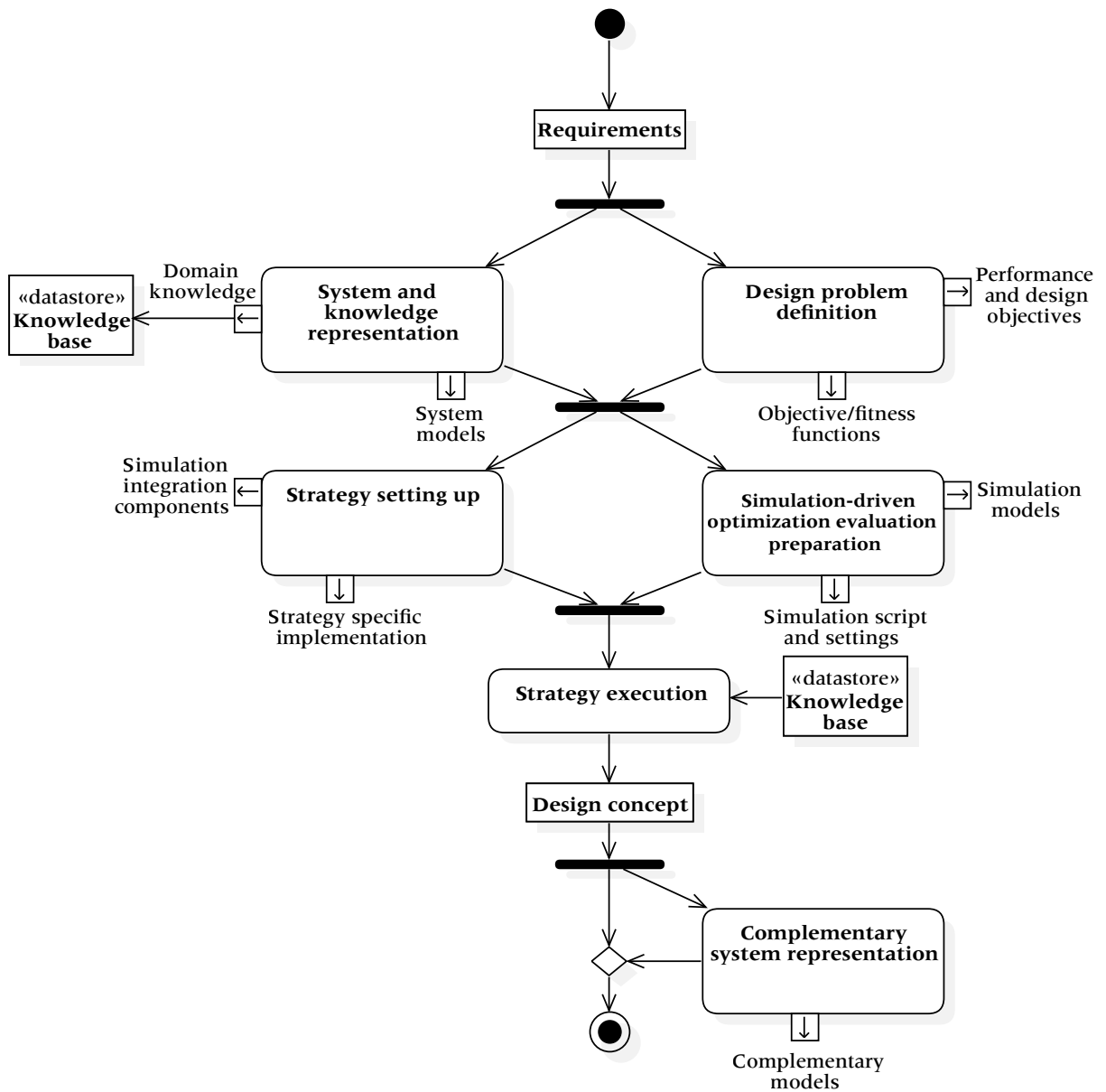


Figure 4.1: Overall solution activity diagram

4.1 Design requirements

An aircraft conceptual design process always starts with some kind of requirements (e.g., aircraft description, mission statement, features list, requirements list, etc), that describe the purpose of the aircraft to be designed and constraints that bound the feasible design space Ω , that is, a space composed by all \vec{x} design concepts that fulfill those constraints.

Requirements

Implementation details 1

The implementation objective of this work is to automatically generate a design concept of a fixed-wing aircraft that performs a simple gliding mission.

For the sake of this implementation a hypothetical mission statement is provided to describe the purpose of the aircraft:

The fixed-wing aircraft to be designed has to be able to glide as far as possible after being released (unpowered) in the air.

Just before the instant t_0 when the aircraft is released, the aircraft flies heading east $\psi = 90^\circ$ (e.g., secured to another aircraft, by its own propulsion, towed by another aircraft, etc) straight and level (pitch $\theta = 0^\circ$, roll $\varphi = 0^\circ$) (Fig. B.1) at $h_0 = 21\,250\text{ft}$ and $\text{TAS}_0 = 180\text{kt}$ (Fig. 4.2).

In order to glide, the aircraft should make the most of its aerodynamic capabilities, descent phase should be steady and safe with minimum or no disturbances in roll and heading, and should be stable enough by its own without any control input.

Although pitch oscillations during gliding are expected, the aircraft should maintain safe pitch and roll angles (safe for crew or payload).

The aircraft weight, when released is 42 000 lb and is expected to be constant until landing.

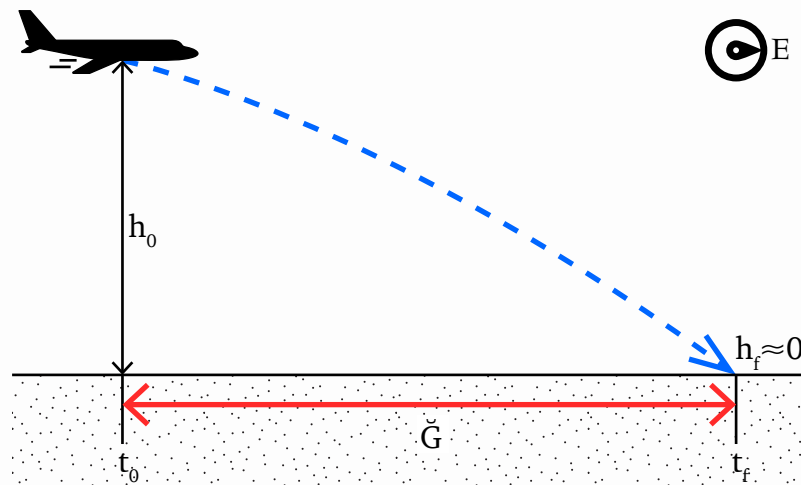


Figure 4.2: Gliding mission

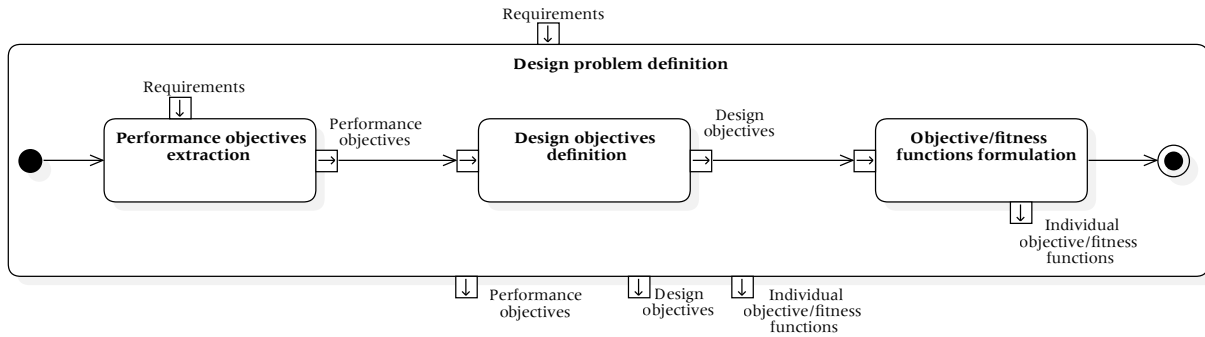


Figure 4.3: Design problem definition procedure

Remark. Besides the mission statement, some conditions are considered for the implementation:

- Maximum cruise altitude and speed, aircraft weight, fuselage overall dimensions and planform areas of wing and stabilizers are taken from specifications of the real aircraft DHC-7 (i.e., . Dash 7) introduced by de Havilland Canada in 1978.
- Propulsion, landing gear and control surfaces systems are not considered in the design process. This decision is not based on a framework limitation itself but to focus design automation in aerodynamic capabilities and to reduce the overall complexity of the system.
- Landing quality is not being analyzed.
- A more complex mission (i.e., take-off, climbing, cruise, descent, landing) is not implemented mainly because the lack of a smart auto-pilot plug-in that works as a test-pilot agent for the generated design concepts.
- No control input is received by the aircraft, the aircraft is on its own (as if inside a wind tunnel).

Implementation details 1

4.2 Design problem definition

Starting from the requirements, some analytical actions are performed in order to properly define the design problem before the execution of the actual automatic design strategy (Fig. 4.3).

Performance objectives extraction

The system description is broken down into a set of k performance objectives $\dot{O} = \{\dot{O}_1, \dots, \dot{O}_k\}$ (the design of a complex system could hardly be set in terms of a single objective, there are

usually more than one objective to met).

Performance objectives

Implementation details 2

The mission statement is broken down into 3 performance objectives:

Mission statement	Performance objective
“glide as far as possible”	$\dot{O}_1 = \text{Maximize glide}$
“minimum or no disturbances in roll”	$\dot{O}_2 = \text{Minimize roll disturbances}$
“... and heading”	$\dot{O}_3 = \text{Maintain aircraft heading}$

Table 4.1: Aircraft performance objectives

Implementation details 2

Design objectives definition

Performance objectives are decomposed into a set of k minimization or maximization of specific qualities design objectives $O = \{O_1, \dots, O_k\}$ (there might be or not a one-to-one mapping relationship between performance objectives and design objectives).

Design objectives

Implementation details 3

Performance objectives are analyzed to define specific design objectives:

Maximize glide

Glide can be maximized by simply maximizing the total horizontal displacement from the point the aircraft is released to the landing location.

Minimize roll disturbances

Two combined strategies are applied to minimize roll disturbances: to minimize the changes the aircraft perform in its lateral inclination during rolling, and to minimize that inclination magnitude (an aircraft can roll from a few degrees up to 180° , that is, flying inverted).

Maintain aircraft heading

Maintaining the aircraft heading during flight is equivalent to minimize or eliminate (unlikely) heading variations.

From performance objectives, a set of 4 design objectives are defined (Table 4.2):

Performance objective	Design objective
$\dot{O}_1 = \text{Maximize glide}$	$O_1 = \text{Maximize glide}$
$\dot{O}_2 = \text{Minimize roll disturbances}$	$O_2 = \text{Minimize roll variation}$
	$O_3 = \text{Minimize lateral inclination}$
$\dot{O}_3 = \text{Maintain aircraft heading}$	$O_4 = \text{Minimize heading variation}$

Table 4.2: Aircraft design objectives

Implementation details 3

Objective/fitness functions formulation

From an optimization problem perspective, design objectives are formulated as minimization (exclusively) of a set of k individual objective or fitness functions $\{f_1(\vec{x}), \dots, f_k(\vec{x})\}$.

Objective functions 1/4

Implementation details 4

Maximize glide Glide distance \check{G} or simply glide is the horizontal displacement in NM from the point P where the aircraft is released at h_0 , to the landing point Q at $h_f \approx 0.0$ ft.

During gliding, the aircraft follows a trajectory through the horizontal plane, it can be a straight or almost straight path in the case of zero or minimal heading variations, a random curved path if heading variations are more significant, and even a path in spiral if heading variance is high.

This path is known as traveled distance, that is, the length of this path if it were stretched to a straight line; however, this length can't be considered as the glide distance since it does not represent the true displacement over the ground (i.e., the aircraft could have flown several miles in closed circles describing an almost null displacement) (Fig. 4.4).

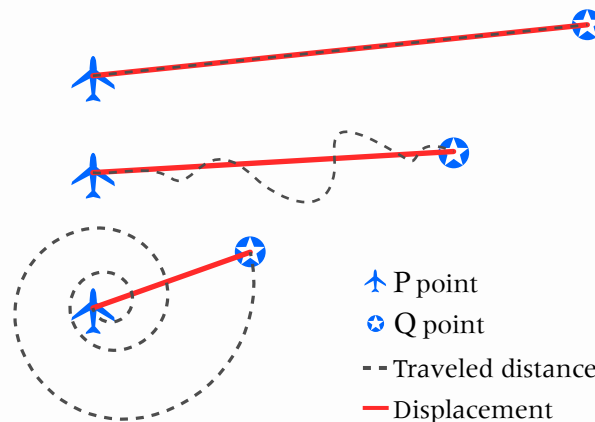


Figure 4.4: Aircraft horizontal flight paths

The aircraft moves inside a nearly spherical three-dimensional space (earth). Its instan-

taneous position along the earth surface is given in geographical coordinates latitude $\dot{\varphi}$ and longitude $\dot{\lambda}$, therefore the distance between P and Q points if calculated as a straight distance (euclidean distance) might be not precise enough to determine the horizontal displacement.

A much better approximation to the true displacement (spherical) would be to calculate the great-circle distance (Fig. 4.5), which is the shortest distance between two points on a spherical surface.

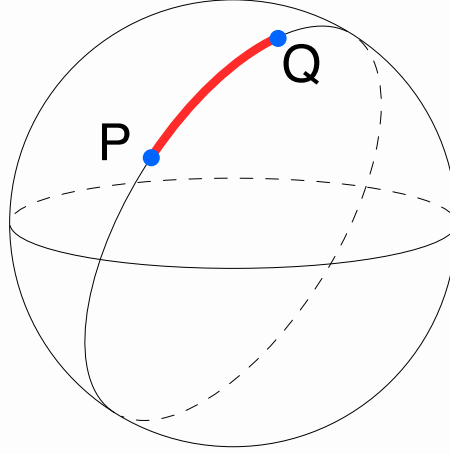


Figure 4.5: Great-circle distance

Source: "A diagram illustrating great-circle distance" by Wikimedia Commons user CheCheDaWaff, used under CC BY-SA-4.0 / Antipodal points removed from original[13]

Among some methods in the literature to calculate the great-circle distance, there is a formula derived from the *haversine formula* [68] that provides a good approximation to compute the aircraft true glide distance as a function of the P and Q latitudes and longitudes:

$$d_{gc} = 2r \arcsin \sqrt{\text{hav } \Delta\dot{\varphi} + \cos \dot{\varphi}_P \cos \dot{\varphi}_Q \text{hav } \Delta\dot{\lambda}} \quad (4.1)$$

where d_{gc} is the great-circle distance between P and Q , r is the radius of the sphere (earth), $\dot{\varphi}_P$ and $\dot{\varphi}_Q$ are latitudes and $\dot{\lambda}_P$ and $\dot{\lambda}_Q$ longitudes of points P and Q respectively, $\Delta\dot{\varphi} = (\dot{\varphi}_Q - \dot{\varphi}_P)$ and $\Delta\dot{\lambda} = (\dot{\lambda}_Q - \dot{\lambda}_P)$ are the differences in latitudes and longitudes, and *hav* is the *haversin*^a function, an archaic trigonometric function defined as:

$$\text{hav } \theta = \frac{1 - \cos \theta}{2} = \sin^2 \frac{\theta}{2} \quad (4.2)$$

The $O_1 = \text{"Maximize glide"}$ design objective is then re-formulated as:

$$O_1 = \min_{\vec{x} \in \Omega} f_1(\vec{x}) \quad (4.3)$$

where Ω is the feasible design space, and $f_1(\vec{x})$ is the glide distance function defined as:

$$\check{G}(\vec{x}) = 2R_{\oplus} \arcsin \sqrt{\text{hav}(\dot{\varphi}_Q(\vec{x}) - \dot{\varphi}_P(\vec{x})) + \cos \dot{\varphi}_P(\vec{x}) \cos \dot{\varphi}_Q(\vec{x}) \text{hav}(\dot{\lambda}_Q(\vec{x}) - \dot{\lambda}_P(\vec{x}))} \quad (4.4)$$

where $\dot{\varphi}_P(\vec{x})$ is the aircraft latitude at point P , $\dot{\varphi}_Q(\vec{x})$ its latitude at point Q , $\dot{\lambda}_P(\vec{x})$ its longitude at point P , $\dot{\lambda}_Q(\vec{x})$ its longitude at point Q and R_{\oplus} is the earth mid radius which is 3440 NM.

Implementation details 4

“haversin stands for “half versed sine”, “versed” is the short for “reversed” [68], [76]

Objective functions 2/4

Implementation details 5

Minimize roll variation The roll variation can be determined by computing the statistical population variance σ^2 of all $\varphi(\vec{x}, t)$ roll measurements of the aircraft.

This measure indicates how much the aircraft variate its rolling attitude during flight, that is, the instantaneous magnitude of the aircraft roll angle changes combined with the frequency of those changes.

A zero variance would mean that the aircraft did not change its initial roll angle at all during the whole gliding (an utopian scenario), a small variance would mean that the aircraft rolling changes by a few degrees, and a big variance would tell that the aircraft flight is all but steady and free of disturbances.

The $O_2 = \text{“Minimize roll variation”}$ design objective is formulated as:

$$O_2 = \min_{\vec{x} \in \Omega} f_2(\vec{x}) \quad (4.5)$$

where $f_2(\vec{x})$ is the statistical population variance of all roll angle measurements of the aircraft:

$$\sigma_{\varphi(\vec{x}, t)}^2 = \frac{\sum_{t=1}^T \left(\varphi(\vec{x}, t) - \mu_{\varphi(\vec{x}, t)} \right)^2}{T} \quad (4.6)$$

where T is the total number of roll measurements, $\varphi(\vec{x}, t)$ is the aircraft roll angle at t measurement, and $\mu_{\varphi(\vec{x}, t)}$ is the roll angle measurements mean:

$$\mu_{\varphi(\vec{x}, t)} = \frac{\sum_{t=1}^T \varphi(\vec{x}, t)}{T} \quad (4.7)$$

Implementation details 5

Objective functions 3/4*Implementation details 6*

Minimize lateral inclination The roll angle tells how much the aircraft is rotated with respect of its longitudinal axis (i.e., how much it is left or right inclined) and the sign of the angle indicates the rolling direction (left or right).

In order to minimize the aircraft lateral inclination, it is only relevant the magnitude (absolute value) of the roll angle, ignoring its direction.

From this, the $O_3 = \text{“Minimize lateral inclination”}$ design objective is re-formulated as:

$$O_3 = \min_{\vec{x} \in \Omega} f_3(\vec{x}) \quad (4.8)$$

where $f_3(\vec{x})$ is defined as:

$$f_3(\vec{x}) = \max_{1 \leq t \leq T} |\varphi(\vec{x}, t)| \quad (4.9)$$

where T is the total number of roll measurements and $|\varphi(\vec{x}, t)|$ is the absolute value of the aircraft roll angle at t measurement.

*Implementation details 6***Objective functions 4/4***Implementation details 7*

Minimize heading variation The heading variation (Fig. 4.4) is determined by computing the statistical population variance σ^2 of all $\psi(\vec{x}, t)$ heading measurements to indicate how much the aircraft changes its heading during flight either from changes in yaw, roll or a combination of both.

The $O_4 = \text{“Minimize heading variance”}$ design objective is then re-stated as:

$$O_4 = \min_{\vec{x} \in \Omega} f_4(\vec{x}) \quad (4.10)$$

where $f_4(\vec{x})$ is the statistical population variance of heading measurements of the aircraft:

$$\sigma_{\psi(\vec{x}, t)}^2 = \frac{\sum_{t=1}^T \left(\psi(\vec{x}, t) - \mu_{\psi(\vec{x}, t)} \right)^2}{T} \quad (4.11)$$

where $\psi(\vec{x}, t)$ is the aircraft heading at t measurement, and $\mu_{\psi(\vec{x}, t)}$ is the heading angle measurements mean.

Implementation details 7

4.3 System and knowledge representation

Subsequent strategy execution initializes and manipulates candidate design concepts to eventually obtain an optimal design.

For this to be accomplished it is required that target system and the domain knowledge are computationally represented as data structures useful for the strategy and even further analysis of the system design.

The system representation fully relies on creation of models or views of the system, these models are defined depending on the aircraft complexity, design objectives and constraints.

This thesis proposes a hierarchical/multi-layer structural model, and a set of design variables as mechanisms to represent the system being designed.

4.3.1 Hierarchical/multi-layer structural model

Structural model

The system is modeled as a collection of hierarchically connected elements, a structure captured through generic “whole – part” aggregation relationships (Fig. 4.6), no matter the role the element plays in real life system (i.e., System, sub-system, assembly, sub-assembly, component, sub-component, section, sub-section, part, sub-part, etc).

This model, by removing details that are not relevant for the strategy execution, can be flexible enough to support different hierarchies, and establishes the base infrastructure to construct other models of the system and the generated design concepts themselves.

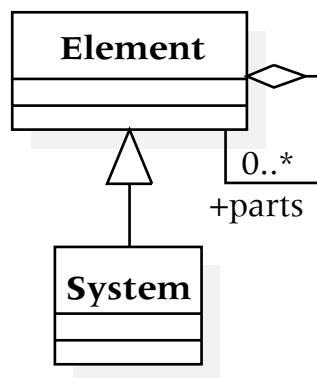


Figure 4.6: System representation - structure

Structural model

Implementation details 8

The fixed-wing aircraft is modeled as follows (Fig. 4.7):

- A fuselage assembly composed by three sections:

- Fwd fuselage section.
- Mid fuselage section.
- Aft fuselage section.
- A wing element modeled as a collection of 1 to 3 wing plane parts.
- A tail assembly composed by:
 - An optional horizontal stabilizer part.
 - An optional vertical stabilizer part.
 - An optional vertical/horizontal stabilizer part.

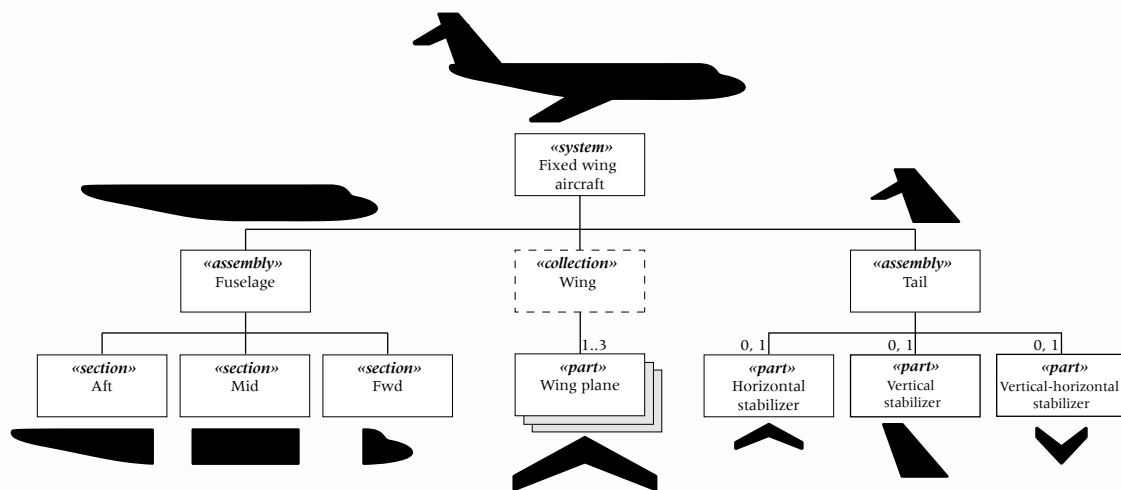


Figure 4.7: Fixed wing aircraft structural model

Implementation details 8

The structural model is defined declaratively through knowledge facts, being the first pieces of knowledge introduced into the knowledge base.

A *system(systemElement)* fact is required to define the system itself and several *part_of(partElement, wholeElement)* facts are used to break down the full structure.

Structural model facts

Implementation details 9

```

system(Aircraft)
...
part_of(Wing, Aircraft)
part_of(Wing Plane, Wing)
...
part_of(Tail, Aircraft)
...

```

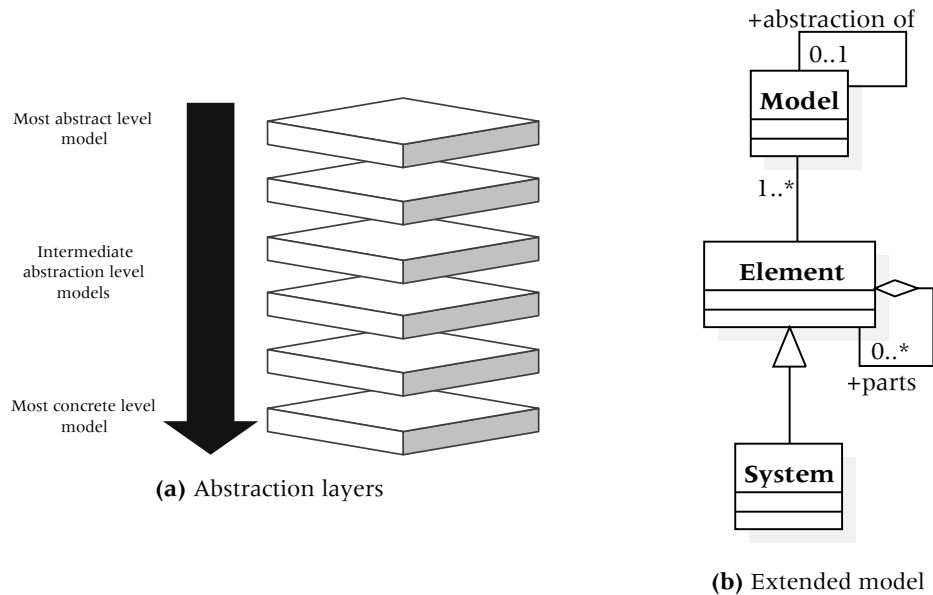


Figure 4.8: System representation - structure + abstraction layers

```
part_of (Horizontal Stabilizer, Tail)
...
```

Implementation details 9

Abstraction layers

Several models or abstraction layers can be defined for the aircraft being designed (the strategy requires at least one model). Each model configures a different view of the same system (Fig. 4.8).

By modeling the system from more abstract models down to more concrete models makes possible in the first place to obtain the structure required by the initialization method proposed in the evolutionary strategy, but it also provides a mechanism to capture classification hierarchies of the system and means to generate design concepts at the corresponding abstraction layer (e.g., if more than one simulation tool are available to test models at different abstraction levels).

Abstraction layers

Implementation details 10

Three abstraction layers are defined for the aircraft, the wing as a key component of the system is chosen as a reference to break the layers down to the most concrete level.

Main wing is the component of a fixed-wing aircraft with the highest impact in flight physics of the aircraft (the airfoil or wing cross-section, in many respects, is the heart of the aircraft and affects overall aerodynamic efficiency during all phases of flight[63]), so it might be an ideal base to develop the design process.

Family (the most abstract)

This layer captures a classification of aircraft based on three distinct wing basic shapes:

- Constant chord (parallelogram shape excluding rhombus).
- Trapezoidal.
- Triangular.

This layer allows to partition the set of possible design concepts that could be generated based on three highly distinguishable clusters or high-order building blocks.

Configuration (intermediate)

The purpose of this layer is to model other classification hierarchies (building-blocks) based on different wing configurations (e.g., Airfoil type, wing vertical position, etc).

The aircraft as defined in the structural model has at least one and up to three wing planes, it may include a vertical stabilizer, a horizontal stabilizer and a vertical/horizontal stabilizer, these elements share several characteristics and can be considered as specific types of wings, making sense to use this layer to generate more or less evenly diverse design concepts at an intermediate level.

Prototype (the most concrete)

This is the model that is used directly by the strategy to build design concepts, the purpose of this layer is not clustering but full detailed characterization of an aircraft (at a conceptual design level).

Implementation details 10

A *model_of(model, system)* fact is required to define each abstraction layer, then the hierarchy is built top-down with *abstraction_of(highLevelAbstractionModel, lowLevelAbstractionModel)* facts.

Abstraction layers definition facts

Implementation details 11

```

model_of(Family, Aircraft)
model_of(Configuration, Aircraft)
model_of(Prototype, Aircraft)
...
abstraction_of(Family, Configuration)
abstraction_of(Configuration, Prototype)

```

Implementation details 11

4.3.2 Design variables

Every element of the system structure and the system itself can be described by a set of design variables.

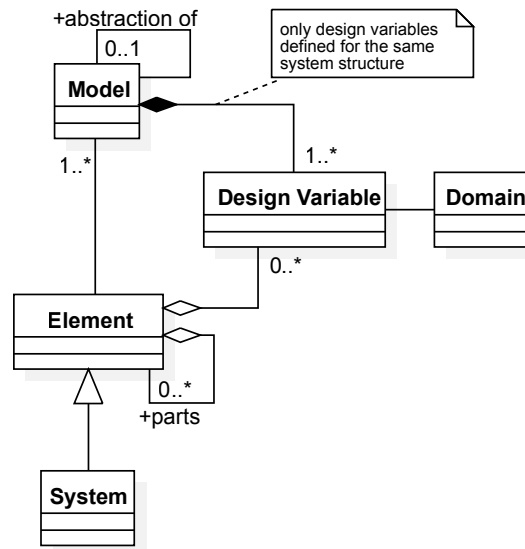


Figure 4.9: Full system representation model

A design variable represents a characteristic or group of characteristics of the associated element, it can take numerical or categorical values and is defined within a specific domain (Fig. 4.9).

Design variables are the key pieces that actually define a system abstraction layer or model in the sense that an abstraction layer is outlined by a subset of the set of all design variables defined for the same system structure the model is representing, or even by different sets of design variables.

Design variables are associated directly to system structure elements, the system itself or abstraction layers, and are implemented declaratively as *variable_of(variable, context)* facts, where *context* can be a part element, system element or model already defined in the knowledge base.

Different effects on system modeling and the design process may be achieved depending on how the variables are defined:

Early design decisions

Some system characteristics might be required to be propagated from higher level models during initialization.

In this case, the variables are configured to be inherited from the level they are initially defined, down to the most concrete models that utilize those variables.

This inheritance is implemented by defining the same variable within each layer where the variable is propagated (i.e., via *variable_of(variable, model)* facts).

Design constraints

Most design constraints can be introduced into the design process by simply defining the domain of the variables accordingly.

Fixed parameters and some equality constraints can be modeled by setting the domain of a variable to a specific single constant value (e.g., 4, 759.50, categorical value).

Most inequality constraints can be defined by setting the domain of the variables as intervals or extensional categorical sets.

In fact, every variable must have an associated domain but the domain may be as broad or specific as required (e.g., $\{1\}$, $\{10.34\}$, $\{\text{categorical value}\}$, \mathbb{N} , \mathbb{R} , \mathbb{Z} , \mathbb{Z}^+ , \mathbb{Z}^- , etc).

Domains and their association with variables are implemented declaratively in a single step with *domain_of(domainSpecification, variable)* facts, or by setting each value of the domain with *domain_value_of(value, variable)* facts. The latter actually builds a domain as a union of all single values.

Remark. Whenever a numerical value is assigned to a design variable having as domain an interval (or union of intervals) either real or discrete, the supplied value is normalized to the design variable current interval domain. This is done with a technique based on modular arithmetic.

This feature is also useful when combining design variables values through arithmetic operations.

Classification

A domain can be associated to a variable in the context of another categorical variable specific domain value, this is indeed, the mechanism through clustering is implemented and it is defined with *domain_of(domainSpecification, variable, contextVariableValue)* or *domain_value_of(value, variable, contextVariableValue)* facts.

Abstraction layer specific characterization

A variable may be defined only in specific abstraction layers, indicating that the associated characteristic is only applicable to that level (e.g., a categorical variable defined for clustering purposes).

Collections of elements

A structural element may have zero or more instances of the same part element, for this reason a design variable can be set as a collection size controller and is implemented declaratively with *counter_of(variable, element)* facts.

Aircraft system full representation

Implementation details 12

The fixed-wing aircraft system is fully modeled at design variable level from the most abstract layer (family) down to the most concrete (prototype), some examples are provided next in (Figs. 4.10 and 4.11).

Actual design variables values are not assigned declaratively but initialized or computed during the evolutionary strategy execution.

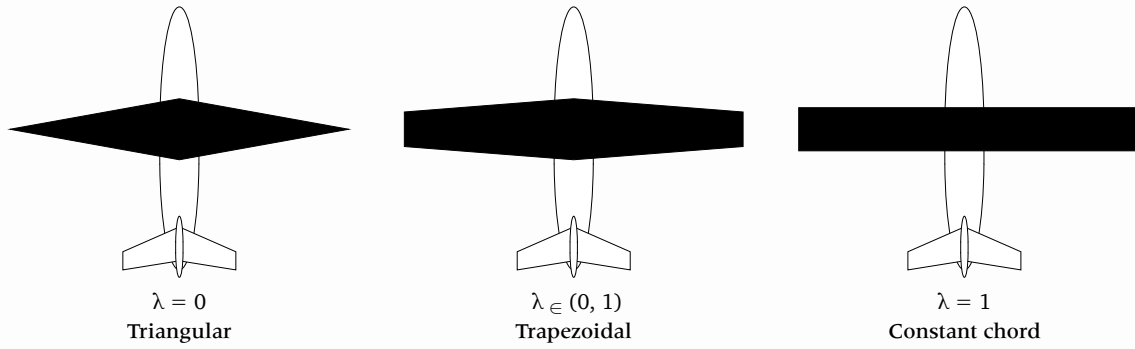


Figure 4.10: Aircraft family layer - wing shape family classification

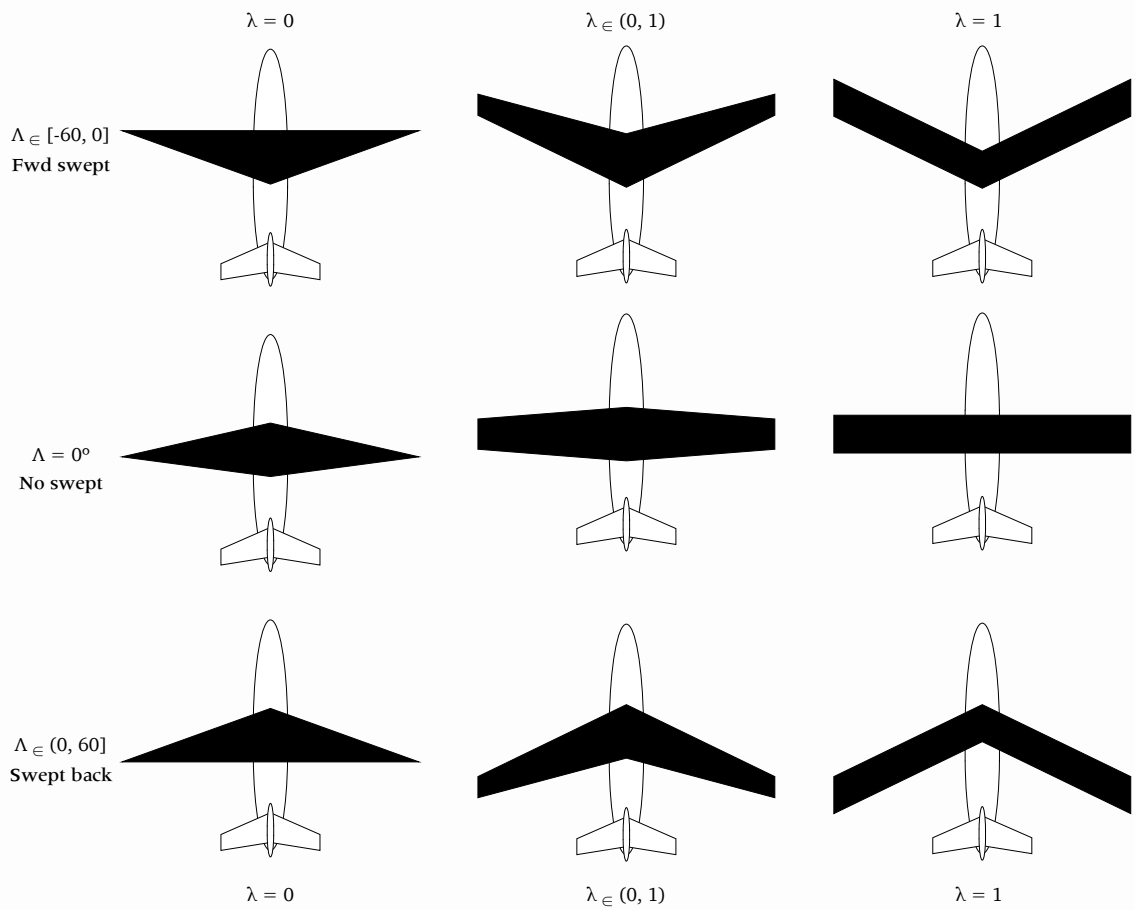


Figure 4.11: Aircraft configuration layer - swept type classification

Implementation details 12

Aircraft full modeling facts

Implementation details 13

The full fixed-wing aircraft system model consists of: 1 system, 2 assemblies, 1 part collection, 3 sections, 1 to 6 parts, 3 abstraction layers, 4 to 9 design variables for the family layer, 14 to 56 design variables for the configuration layer and 32 to 103 variables for the prototype layer (some design variables are configured as fixed parameters):

```

system(Aircraft)
...
part_of(Wing, Aircraft)
part_of(Wing Plane, Wing)
...
part_of(Tail, Aircraft)
...
part_of(Horizontal Stabilizer, Tail)
...
model_of(Family, Aircraft)
model_of(Configuration, Aircraft)
model_of(Prototype, Aircraft)
...
abstraction_of(Family, Configuration)
abstraction_of(Configuration, Prototype)
...
variable_of(S, Aircraft)           // Aircraft total planform surface in ft2
domain_of(860.00, S)             // Fixed parameter, S = 860.00ft2
...
variable_of(Nw, Wing)             // Number of wing planes
domain_of({1..3}, Nw)           // Nw ∈ {x ∈ ℤ+ | x ≤ 3}
counter_of(Nw, Wing plane)       // Wing plane collection size controller
...
variable_of(Sf, Wing Plane)       // Wing plane shape family
domain_of({Constant chord, Trapezoidal, Pointed}, Sf)
...
variable_of(λl, Wing Plane)       // Wing plane taper level
// λl domain depends on Sf value context
domain_value_of(Pointed, λl, Pointed)
domain_value_of(Low, λl, Trapezoidal)
domain_value_of(Intermediate, λl, Trapezoidal)
domain_value_of(High, λl, Trapezoidal)
domain_value_of(Constant chord, λl, Constant chord)
...
variable_of(λ, Wing Plane)       // Wing plane taper ratio
// λ domain depends on λl value context
domain_of(0.0, λl, Pointed)
domain_of((0.0, 0.3), λ, Low)
domain_of([0.3, 0.6), λ, Intermediate)
domain_of([0.6, 1.0), λ, High)
domain_of(1.0, λ, Untapered)

```



```

...
variable_of(Nw, Family)
variable_of(Sf, Family)
...
variable_of(Nw, Configuration) // Inherited from family
variable_of( $\lambda$ , Configuration)
...
variable_of(S, Prototype)
variable_of(Nw, Prototype) // Inherited from configuration
variable_of( $\lambda$ , Prototype)
...
There are 24 constraints implemented as fixed parameters:
...
variable_of(S, Aircraft) // Aircraft total planform surface in ft2
domain_of(860.00, S)
variable_of(MLW, Aircraft) // Aircraft maximum landing weight in lb
domain_of(42000.00, MLW)
...
// Horizontal stabilizer planform surface in ft2
variable_of(Sh, Horizontal stabilizer)
domain_of(217.00, Sh)
...
// Vertical stabilizer planform surface in ft2
variable_of(Sv, Vertical stabilizer)
domain_of(170.00, Sv)
...
// Wing plane airfoil maximum thickness location (%), and homologous
variables for horizontal, vertical and vertical-horizontal stabilizers
variable_of( $\frac{t_{max}}{c}x$ , Wing plane)
domain_of(30,  $\frac{t_{max}}{c}x$ )
...
// Vertical stabilizer airfoil camber (%)
variable_of( $\frac{\delta v_{max}}{c}$ , Vertical stabilizer)
domain_of(0,  $\frac{\delta v_{max}}{c}$ )
// Vertical stabilizer airfoil maximum camber location (%), and homologous
design variables for vertical-horizontal stabilizer
variable_of( $\frac{\delta v_{max}}{c}x$ , Vertical stabilizer)
domain_of(0,  $\frac{\delta v_{max}}{c}x$ ) // Vertical stabilizer kept symmetrical
...
// Vertical stabilizer dihedral angle in degrees
variable_of( $\Gamma_v$ , Vertical stabilizer)
domain_of(90,  $\Gamma_v$ ) // Vertical wing
// Vertical stabilizer incidence angle in degrees
variable_of( $\iota_v$ , Vertical stabilizer)
domain_of(0.0,  $\iota_v$ ) // Vertical stabilizer kept directionally straight
variable_of( $\epsilon_v$ , Vertical stabilizer) // Vertical stabilizer twist in degrees
domain_of(0.0,  $\epsilon_v$ ) // Vertical stabilizer kept directionally straight

```

```

...
// Fuselage drag coefficient
variable_of( $f_{Cd}$ , Fuselage)
domain_of(0.075,  $f_{Cd}$ )
// The ratio of fuselage length of forward (fwd) (nose) section to diameter
ratio
variable_of( $l^n/a$ , Fwd fuselage)
domain_of(1.61,  $l^n/a$ )
// The ratio of fuselage length of aft (tail) section to diameter ratio
variable_of( $l^t/a$ , Aft fuselage)
domain_of(4.26,  $l^t/a$ )
variable_of( $lm$ , Mid fuselage)           // Fuselage length of mid section in ft
domain_of(24.33,  $lm$ )
variable_of( $fh$ , Mid fuselage)           // Fuselage maximum height in ft
domain_of(8.83,  $fh$ )
variable_of( $fw$ , Mid fuselage)           // Fuselage maximum width in ft
domain_of(9.74,  $fw$ )
// Fuselage mid section shape superellipse  $\check{m}$  parameter
variable_of( $f_{\check{m}}$ , Mid fuselage)
domain_of(1.98,  $f_{\check{m}}$ )
// Fuselage mid section shape superellipse  $\check{n}$  parameter
variable_of( $f_{\check{n}}$ , Mid fuselage)
domain_of(1.98,  $f_{\check{n}}$ )
// Fuselage nose tip y location from fuselage center in ft
variable_of( $n_y$ , Fwd fuselage)
domain_of(-1.16,  $n_y$ )
// Fuselage tail tip y location from fuselage center in ft
variable_of( $t_y$ , Aft fuselage)
domain_of(4.41,  $t_y$ )

```

See (Table A.1 and ??) for full aircraft conceptual design variables reference.

Implementation details 13

Design variables vector

The strategy makes use of a specific model (from the abstraction layers) to collect all relevant m design variables x and build up a minimal representation of the system or design variables vector $\vec{x} = [x_1 \ x_2 \ \dots \ x_m]^T$ that, when containing actual values, fully characterizes a design concept of the system.

Each design variable x_i is in fact an κ tuple $(x_{i1}, \dots, x_{i\kappa})$ containing at least 3 members:

- The variable name.
- The variable assigned value.
- The variable current domain (specific domain built through abstraction layers concretization).

, plus optional members like:

- Variable full domain (the union of all specific domains).
- Inheritance relationship.
- Collection relationship.
- Other required custom members depending on the application.

The vector size m can be fixed or variable, depending on the system structure (i.e., system structures having optional elements and/or collections of elements).

Actions to manipulate a design during the strategy execution (i.e., variation operators) always operate on the design variables vector.

Design variables vector

Implementation details 14

The evolutionary strategy uses an aircraft design variables vector built from the prototype abstraction layer, that is, the most concrete model of the aircraft system.

The design variables vector is a variable size vector going from 32 up to 103 positions (the vector of tuples is represented as a matrix):

Variable	S	...	Nw	λ_1	...	λ_2	...
Value	860.00	...	2	0.43	...	0.76	...
Domain	860.00	...	{1..3}	[0.3, 0.6)	...	[0.6, 1.0)	...

Full domain	860.00	...	{1..3}	[0.0, 1.0]	...	[0.0, 1.0]	...

Table 4.3: Design variables vector example, a design concept with 2 wing planes, a taper ratio λ_1 of 0.43 for the first wing plane, and a taper ratio λ_2 of 0.76 for the second wing plane

Implementation details 14

4.3.3 Domain model

This work proposes a domain model of the system being built from the design variables vector.

It might and usually expands the number of features that describe the full system being designed in terms of the application domain.

This model provides the designers with a specific application domain perspective of the system, and can be used as a bridge between the design concepts generated by the strategy (as design variables vectors) and parametric simulation models or other analysis models.

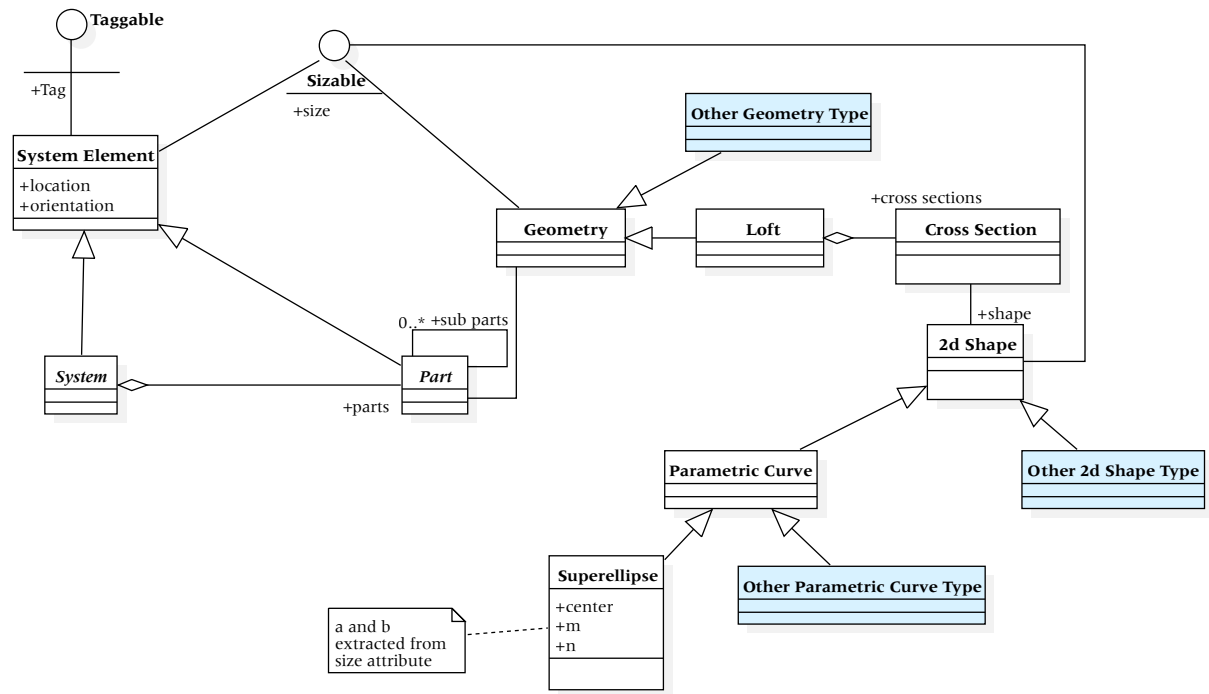


Figure 4.12: Generic domain model (extension point elements in blue)

Domain knowledge can be introduced in the knowledge base to assist building the domain model (e.g., equations from physics) or can be embedded in the domain model logic directly.

An abstract object graph is proposed as extension points to implement application specific domain models (Fig. 4.12 and Table 4.4).

This hierarchy captures the structural model, and contains extensions to implement the geometry of the system and its parts, and a mechanism for fine grain identification of systems and parts through the design process.

Lofting geometry

A lofting technique (Fig. 4.13) mechanism is provided to define the geometry in 3D space of the system part elements.

Lofting is a common technique used by the aircraft industry when defining the external geometry of the aircraft[63].

A loft or lofting is the creation of a 3D solid by specifying a series of cross sections that define the shape of the resulting solid, at least two cross sections are required to build a loft[7].

Cross sections of the loft structure might be any planar 2D shape, although parametric superellipse shaped cross sections are provided (Fig. 4.14).

Superellipse curves allow to define a broad range of cross sections, from astroids (four-armed stars with concave sides) to rounded rectangles including rhombus like shapes, circles, ellipses and squircles (most of aerodynamic bodies are not axisymmetric and fuselage cross sections are

Class	Type	Purpose / Description
Taggable	Interface	Makes an object identifiable by a unique tag, the tag may keep a hierarchical chained composite tag (e.g., for identification and tracking purposes)
Sizable	Interface	Makes an object sizable, that is, an object having either two dimensional (2D) size (height, length) or 3D size (width, height, length)
System Element	Abstract	The generic structural element of the system, contains location (x, y, z) and orientation (θ, ψ, ϕ) in 3D space
System	Abstract	The generic structural root element (i.e., . system) as aggregation of parts
Part	Abstract	The generic structural part element (i.e., . part), may be an aggregation of sub parts and has an associated geometry
Geometry	Abstract	The method a part element is shaped in 3D space
Loft	Concrete	A 3D shape formed by a collection of joined cross sections (similar to an extrusion)
Cross Section	Concrete	A slice of a loft defined by a 2D shape
2D Shape	Abstract	A shape in 2D space
Parametric curve	Abstract	A shape constructed by a parametric function in 2D space
Superellipse	Concrete	A closed parametric curve that is a generalization for circles, ellipses, squircles, astroids, etc

Table 4.4: Domain model generic classes

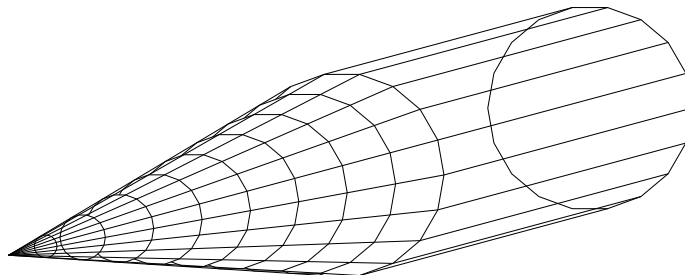


Figure 4.13: Lofting geometry example

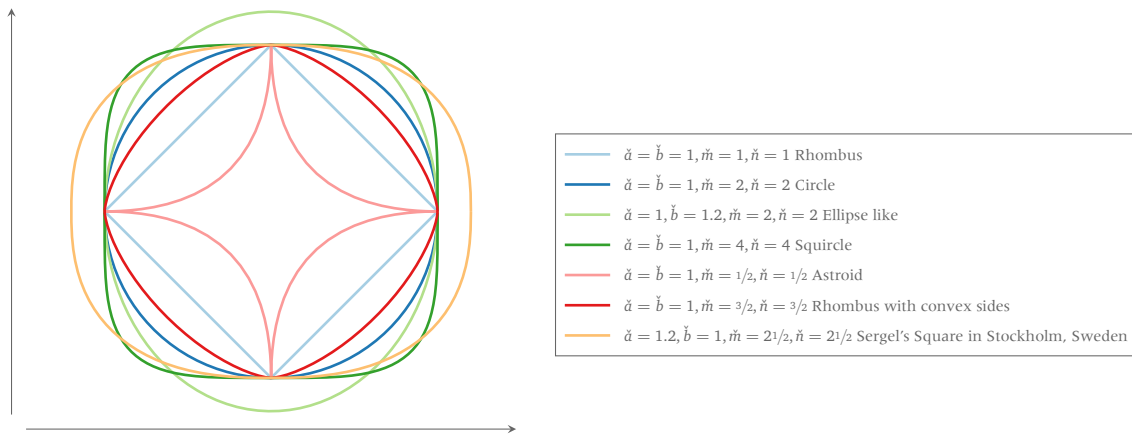


Figure 4.14: Some superellipse based cross sections

not round[17]).

Superellipse cross-section shapes can be used to develop more aerodynamic models and they have been used to study geometric shaping effects on forebody aerodynamic characteristics[17].

Superellipse curves are parametric functions in the form:

$$\begin{aligned} x(\theta) &= |\cos \theta|^{\frac{2}{\check{m}}} \check{a} \operatorname{sig}(\cos \theta) \\ y(\theta) &= |\sin \theta|^{\frac{2}{\check{n}}} \check{b} \operatorname{sig}(\sin \theta) \end{aligned} \quad (4.12)$$

where θ is the parameter of the function with domain $[0, 360]$ degrees, \check{a} is the shape width, \check{b} is the shape height, \check{n} is the horizontal curvature parameter > 0 , and \check{m} is the vertical curvature parameter > 0 .

Remark. Lofting geometry can be extended to work with other types of parametric closed curves and other 2D shapes in general. Even other geometry techniques different from lofting can be adapted as far as they provide size and 3D shape of the parts.

Aircraft domain model

Implementation details 15

The domain model of the fixed-wing aircraft is an extension of the domain model base hierarchy.

There are few domain specific new attributes that are derived from the design variables vector but there is no direct one-to-one mapping between the structural model and the domain model.

This is explained by the fact that the domain model is coupled to the application domain (i.e., aircraft conceptual design, aeronautics, etc), and the structural model focus on capturing structure only from the systems engineering perspective.

Fuselage sections are generalized into “No-lift bodies” to distinguish parts that does not produce lift (in the simulator) from the “Wing” type parts that produce lift.

A wing plane is actually modeled as a pair of twin wings: port and outboard wings.

The fixed-wing aircraft domain model is implemented through an object graph (Fig. 4.15) that is an extension from the generic domain model.

From this model, human readable text representations in JavaScript Object Notation (JSON) format are generated during the execution of the strategy.

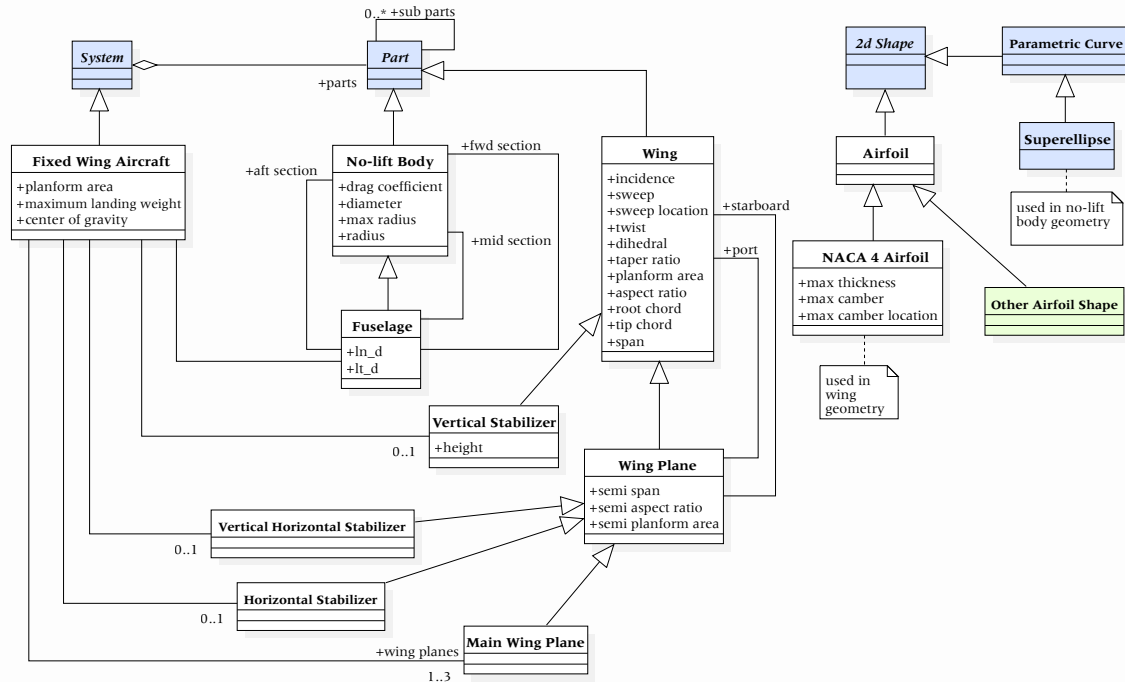


Figure 4.15: Fixed wing aircraft domain model (extension point elements in blue and green)

Geometries of the aircraft parts are implemented through the lofting technique.

Superellipse based geometries Superellipse cross sections are used for the fuselage sections.

The fuselage fwd (fore) section loft consists of two cross sections: a tip point, and a rear cross section.

The fuselage mid section contains two identical cross sections (Fig. 4.16), forming a kind of cylinder.

The aft section is composed by a front cross section and a tail tip point.

In order to join the three sections, the rear cross section of the fwd fuselage section, the front cross section of the aft fuselage section must

be equal in shape and dimensions to the mid section cross sections.

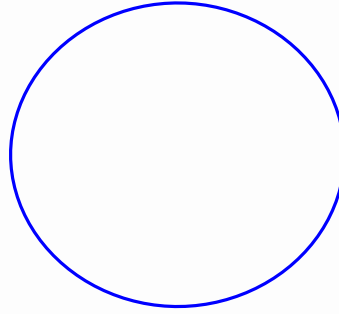


Figure 4.16: Fuselage mid section cross section generated by a superellipse with parameters $\check{a} = 1.1\check{b}$, $\check{m} = 1.98$, $\check{n} = 1.98$

Airfoil shape based geometries Specialized airfoil shapes (wing profile or wing section) are used for the wing type parts cross sections.

Every wing type part is composed by 6 evenly distributed cross sections. Their size and orientation are determined by the corresponding design variables (Fig. B.7).

The airfoil shapes in this work are limited to the 4-digit series of the NACA airfoils standard, however, the domain model can be extended to other airfoil shape types.

A 4-digit series NACA airfoil is designated by a short-hand code representing the essential elements controlling the shape of the generated cross section.

The short-hand code is a 4-digit code of the form $pmxx$, where p is the location of the maximum camber as a percentage of the chord line $\frac{x\delta_{max}}{c}x$ multiplied by 10, m is the maximum camber as a percentage of the chord length $\frac{\delta_{max}}{c}$ multiplied by 100, and xx is the thickness-chord ratio $\frac{t_{max}}{c}$ multiplied by 100, that is, “ $pm12$ ” designates a 12-percent-thick 4-digit airfoil[45].

Symmetrical airfoils in the 4-digit-series family are designated by a 4-digit number of the form NACA 00 xx . The first two digits indicate a symmetric airfoil (no camber); the second two, the thickness-chord ratio[45].

Geometry of the wing sections are computed by the following procedure described and explained in [1], [45]:

Ordinates in the form of ordinate-chord ratio percentages for the NACA symmetric 4-digit airfoil family are described by the following equation:

$$\pm \frac{t_y}{c} = 5 \frac{t_{max}}{c} \left[a_0 \left(\frac{x}{c} \right)^{1/2} + a_1 \left(\frac{x}{c} \right) + a_2 \left(\frac{x}{c} \right)^2 + a_3 \left(\frac{x}{c} \right)^3 + a_4 \left(\frac{x}{c} \right)^4 \right] \quad (4.13)$$

where $\frac{t_y}{c}$ is the thickness ordinate-chord ratio for every point within the upper and lower surfaces of the airfoil, c is the chord length, x is the position along the chord from 0 to c ,

$\frac{t_{max}}{c}$ is the maximum thickness-chord ratio percentage and a_i are constants with values $a_0 = 0.2969$, $a_1 = -0.1260$, $a_2 = -0.3516$, $a_3 = 0.2843$ and $a_4 = -0.1015$.

Actual coordinates (x_U, y_U) (relative to the wing section chord) of the upper airfoil surface, and (x_L, y_L) of the lower surface are $x_U = x_L = \frac{x}{c}$, $y_U = +\frac{t_y}{c}$, and $y_L = -\frac{t_y}{c}$.

For asymmetrical 4-digit airfoil family of wing sections the mean camber line (Fig. B.7) needs to be computed by the equation:

$$\frac{\delta_y}{c} = \begin{cases} \frac{\frac{\delta_{max}}{c}}{\left(\frac{x\delta_{max}}{c}x\right)^2} \left[2\frac{x\delta_{max}}{c}x\left(\frac{x}{c}\right) - \left(\frac{x}{c}\right)^2 \right], & 0 \leq x \leq \frac{x\delta_{max}}{c}xc \text{ (forward of maximum ordinate)} \\ \frac{\frac{\delta_{max}}{c}}{\left(1 - \frac{x\delta_{max}}{c}x\right)^2} \left[\left(1 - 2\frac{x\delta_{max}}{c}x\right) + 2\frac{x\delta_{max}}{c}x\left(\frac{x}{c}\right) - \left(\frac{x}{c}\right)^2 \right], & \frac{x\delta_{max}}{c}xc \leq x \leq c \text{ (aft of maximum ordinate)} \end{cases} \quad (4.14)$$

where $\frac{\delta_y}{c}$ is the camber line ordinate-chord ratio, $\frac{\delta_{max}}{c}$ is the maximum camber as a percentage of the chord c , that is the maximum ordinate of the mean camber line, and $\frac{x\delta_{max}}{c}x$ is chordwise position of the maximum camber.

The coordinates (x_U, y_U) and (x_L, y_L) of the cambered airfoil are $x_U = \frac{x}{c} - \frac{\delta_y}{c} \sin \theta$, $y_U = \frac{\delta_y}{c} + \frac{\delta_y}{c} \cos \theta$, $x_L = x + \frac{\delta_y}{c} \sin \theta$ and $y_L = \frac{\delta_y}{c} - \frac{\delta_y}{c} \cos \theta$, where $\theta = \arctan\left(\frac{d\frac{\delta_y}{c}}{d\frac{x}{c}}\right)$,

$$\frac{d\frac{\delta_y}{c}}{d\frac{x}{c}} = \begin{cases} \frac{2\frac{\delta_{max}}{c}}{\left(\frac{x\delta_{max}}{c}x\right)^2} \left(\frac{x\delta_{max}}{c}x - \frac{x}{c}\right), & 0 \leq x \leq \frac{x\delta_{max}}{c}xc \\ \frac{2\frac{\delta_{max}}{c}}{\left(1 - \frac{x\delta_{max}}{c}x\right)^2} \left(\frac{x\delta_{max}}{c}x - \frac{x}{c}\right), & \frac{x\delta_{max}}{c}xc \leq x \leq c \end{cases}$$

Implementation details 15

4.4 Simulation-driven design preparation

Design concepts are evaluated through objective or fitness functions specifically defined for each design objective.

The mapping between the design concept and the value of those functions is determined with the support of data obtained from computer simulations.

Simulation data usually do not provide a direct objective function value for a given design but further analytical procedures are implemented in those functions to transform data into an output value.

4.4.1 Scalarization

The solution proposed in this work deals with multiple objectives through scalarization techniques that virtually convert a multi-objective design problem into a single-objective problem by combining all minimization of k objective or fitness functions $\vec{f}(\vec{x})$ objectives into a single minimization of a scalarized fitness function objective:

$$\min_{\vec{x} \in \Omega} \vec{f}(\vec{x}) \quad (4.15)$$

where Ω is the feasible design space, and $\vec{f}(\vec{x})$ is the scalarized fitness function:

$$\vec{f}(\vec{x}) = [f_1(\vec{x}) \quad f_2(\vec{x}) \quad \dots \quad f_k(\vec{x})]^\top \quad (4.16)$$

Scalarized fitness function

Implementation details 16

The *weighted metric* method is used to implement the scalarized fitness function $\vec{f}(\vec{x})$ of the fixed-wing aircraft as:

$$\vec{f}(\vec{x}) = \sqrt[p]{\sum_{o=1}^k w_o |\vec{f}_o(\vec{u}) - f_o(\vec{x})|^p} \quad (4.17)$$

where k is the number of objectives w_o is the weight of objective o (Table 4.5), $\vec{f}_o(\vec{u})$ is the individual objective function of an utopian design concept \vec{u} for objective o , and $f_o(\vec{x})$ is the individual objective function of aircraft design concept \vec{x} for objective o . The $p = 4$ parameter is empirically determined, as the parameter gets bigger the fitness distance between aircraft concepts gets smaller (from an aircraft fitter than other), that is, the fitness resolution increases.

The utopian aircraft design concept \vec{u} is defined as an aircraft with a performance (Table 4.6), such that when evaluated, its fitness function $\vec{f}(\vec{u}) = 0.0$.





Design objective O_o	Objective weight w_o	
$O_1 :=$ Maximize glide	$\frac{4}{10.5}$	
$O_2 :=$ Minimize roll variation	$\frac{3.5}{10.5}$	
$O_3 :=$ Minimize lateral inclination	$\frac{2}{10.5}$	
$O_4 :=$ Minimize heading variation	$\frac{1}{10.5}$	

Table 4.5: Utopian aircraft design concept objectives weights

Design objective O_o	Objective function $\vec{f}_o(\vec{u})$	Value
$O_1 :=$ Maximize glide	$\check{G}(\vec{u})$	160.00 NM
$O_2 :=$ Minimize roll variation	$\sigma_{\varphi(\vec{u},t)}^2$ $1 \leq t \leq T$	0.0 deg ²
$O_3 :=$ Minimize lateral inclination	$\max_{1 \leq t \leq T} \varphi(\vec{u}, t) $	0°
$O_4 :=$ Minimize heading variation	$\sigma_{\psi(\vec{u},t)}^2$ $1 \leq t \leq T$	0.0 deg ²

Table 4.6: Utopian aircraft design concept performance

Implementation details 16

4.4.2 Simulation model

Simulation tools define their own technical specifications that input models are required to met, they just cannot take a generated design concept as is and run a simulation.

A conversion has to be done to pass from a design concept generated by the strategy to the parametric simulation model which structure and attributes are fully coupled to the simulation tool.

The domain model proposed by this work may highly simplify the simulation model construction by using it as a bridge between the design variables vector and the parametric simulation model. An advantage of this method is that researchers are able to implement simulation models for more than one simulator and swap them for different design results or to simulate other aspects of the system.

Parametric simulation models

Implementation details 17

Aircraft

In this solution the parametric simulation model is built directly from the domain model, the specific parametric simulation model is a .acf proprietary text based format file for the X-Plane simulator and its aircraft modeling companion utility called Plane-Maker.

The parametric simulation model defines the lofting geometry composed by superellipse based non-regular hexadecagon cross sections, weight, theoretical optimal center of gravity of the aircraft and no-lift bodies drag, besides some domain specific attributes of the aircraft derived from the basic design variables vector (e.g., wing type part chord at root, chord at tip, span, semi-length, etc) that are required by the simulator to reproduce the flight physics.

Airfoils

Regarding airfoils (wing cross sections), the lofting geometry is composed by 14 vertex 2D wing shape cross sections, 7 vertex for the upper surface of the airfoil and 7 for the lower surface.

Geometry of wing type parts and their cross sections are entered into the .acf parametric simulation model; however, aerodynamic performance parameters specific to an airfoil type are not included but referenced to a separated airfoil parametric simulation model in an .afl proprietary text based format file for the X-Plane simulator and its airfoil modeling utility called Airfoil-Maker.

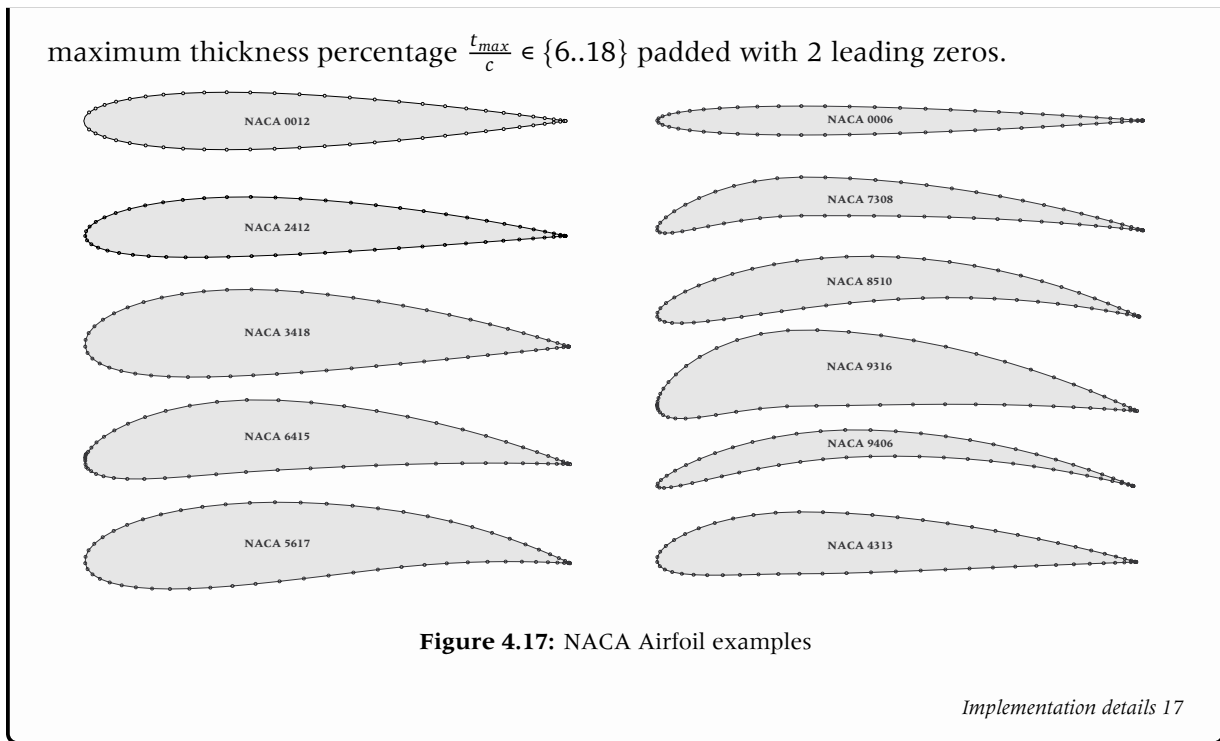
Airfoil parametric simulation models contain a low resolution base cross section geometry of a given airfoil and aerodynamic performance parameters (i.e., lift and drag for a given angle of attack) that the simulator uses (in addition to the other aircraft parameters specified in the .acf file and black box simulation mathematical model) to fully simulate the flight physics of the aircraft inside a synthetic environment.

For the sake of this work, a set of 832 NACA 4-digit-series airfoil parametric simulation models are generated (Fig. 4.17) in batch prior the strategy execution.

Performance data is obtained by other simulation tool specialized in airfoil aerodynamic analysis called J_{AVA}FOIL .

This set of airfoil models works as a pool of airfoils available for the generated aircraft design concepts, whatever combination of airfoil related design variables values are generated by the strategy there will be an airfoil model that matches them. Specifically, 9 groups of 26 airfoils each are generated: 0006 - 0018, 1006 - 1018, 2006 - 2018, 3006 - 3018, 4006 - 4018, 5006 - 5018, 6006 - 6018, 7006 - 7018, 8006 - 8018, and 9006 - 9018.

Airfoil NACA 4-digit-series $pmxx$ parametric simulation models are determined based on the procedure described previously, by using the design variables and their domains: where m is the camber percentage $airfoilmaxcamber \in \{0..9\}$, p is the maximum camber location percentage $\frac{x\delta_{max}}{c}x \in \{0, 10, 20, 30, 40, 50, 60\}$ divided by 10, and xx is the airfoil



4.4.2.1 Computer simulation

Before executing the strategy and start generating design concepts, the simulation tool needs to be prepared in order to fix reproducible conditions and set the testing scenario.

This may include configuring the simulator itself, setting parameters, defining a simulation script, implementing simulation controllers (if required), etc.

Flight simulator

Implementation details 18

A flight simulator (X-Plane) is used to re-create the mission statement defined in the requirements and to produce the data required to evaluate generated aircraft concepts.

Some configuration and customization is required in order to set reproducible conditions and enable evaluation fluency and simulation data retrieval.

Flat synthetic scenery

The simulator provides a geographical scenery that reproduces the actual earth terrain, oceans, etc.

A custom scenery is built for an enclosed region (from 10°00'00"N, 30°00'00"W to 30°00'00"N, 20°00'00"W) as a flat terrain at sea level, free of mountains or any other obstacle.

The scenario is flattened to improve the simulator resource consumption (the simulator has the drawback of being unable to simulate an aircraft without disabling

the graphical user interface).

Simulation controller

A software component is developed by using the simulator plug-in interface software development kit (SDK).

The controller is required to launch the simulation with the same synthetic environment conditions and simulation settings (Tables 4.7 and 4.8) during the strategy execution and to collect and persists simulation data for subsequent evaluation.

Implementation details 18

Simulation script

Implementation details 19

As part of aircraft design evaluation, each generated parametric simulation model is tested in the simulator according to a predefined and reproducible scenario:

Simulation starts with the aircraft positioned in the synthetic world, flying at cruise altitude and speed towards east.

The aircraft does not have any propulsion system but its initial momentum and should start gliding to ground.

As soon as airspeed decreases, lift will drop (if any) and should get a pitch down attitude (Fig. 4.2).

The aircraft should glide down to ground, fall as if in free fall or even float depending on its configuration governed by the design variables.

Simulation ends when the aircraft reaches the ground or whether simulation-time or real-time timeout expires.

Several fine-grained data coming from simulation and aircraft performance itself are collected during the simulation execution.

Implementation details 19

Simulation configuration*Implementation details 20*

Setting/parameter	Value
h_0	21 250 ft
TAS ₀	180 kt
θ	0°
φ	0°
ψ	90° (East)
Initial location P	19°52'48"N, 29°59'24"W
t_0	00:00 Z
Weather	no rain, clear sky, calm wind

Table 4.7: Artificial environment conditions

Setting/parameter	Value
Simulation-time timeout	3 h
Simulation-time compression	up to 32x
Real-time timeout	15 min

Table 4.8: Simulation settings*Implementation details 20*

4.5 Knowledge based evolutionary strategy

The knowledge based evolutionary strategy proposed in this work is inspired by the generalized model of an EA presented in the theoretical framework chapter.

It provides specific procedures, mechanisms and operators defined with —enabling automatic conceptual design not limited by mathematical models or equations— in mind.

Although the strategy defines a common implementation approach, specific algorithms can be tailored based on the same strategy depending on the testing scenario, design objectives and research needs.

In order to deal with the —complex, multi-objective— nature of an aircraft design concept, to maintain a very small population size (to tackle simulation computational expensiveness), and to increase probability of success (i.e., achieve better fitness), the strategy relies on the following key aspects and components:

- Top-down initialization.
- Cumulative candidate offspring.
- Knowledge based / classical evolutionary variation operators combination.
- Highly configurable variation and selection operators.
- Population diversity maintaining mechanisms.
- Simulation-driven design integration.

The overall flow of the strategy (Fig. 4.18) starts at generation $g = 1$ by initializing a population X_1 of μ top-down generated designs concepts. Initial population quality $\{\vec{f}(\vec{X}_{1,1}), \dots, \vec{f}(\vec{X}_{1,\mu})\}$ is assessed by simulation-driven evaluation.

While termination criteria are not met (e.g., desired fitness accomplished, maximum number of generations reached, maximum number of generations with no-improvement exceeded — stagnation—, terminated by user request, etc.) a series of variation operators, both knowledge based and classical evolutionary operators are selectively applied to generate new designs that are variants of initial designs and/or their descendants.

Once variation operators have been applied, a selection operator determines next generation X_{g+1} members.

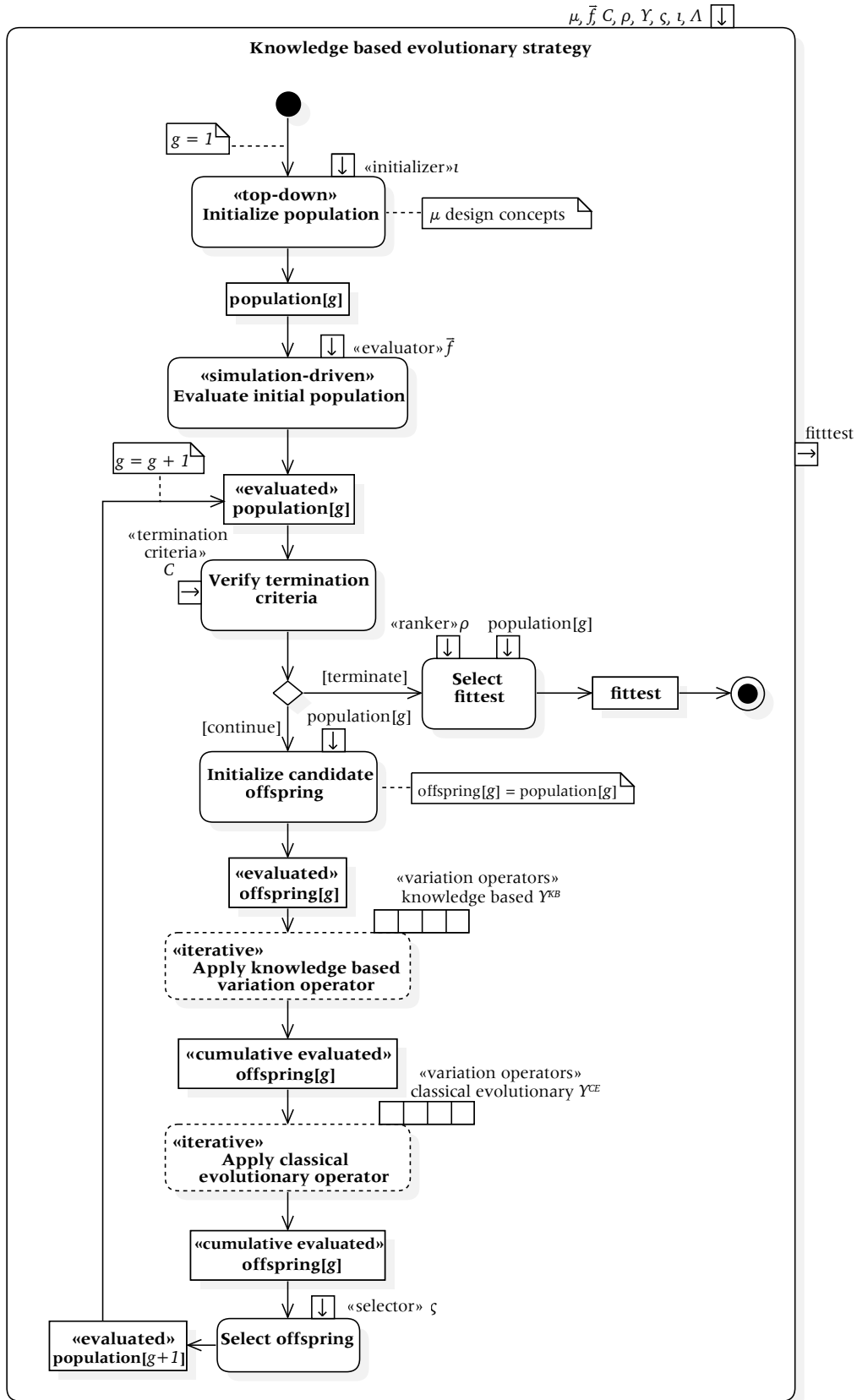


Figure 4.18: Knowledge based evolutionary strategy - overall activity diagram

Algorithm 4.1: Knowledge base evolutionary strategy - overall

Parameters: μ = population size,
 \vec{f} = fitness function, $\Lambda^{\vec{f}} = \{\Lambda_l^{\vec{f}} : \Lambda_l^{\vec{f}} \text{ is a parameter of } \vec{f}\}$,
 $C = \{C_i : C_i \text{ is a termination criterion}\}$,
 $\Lambda^C = \left\{ \left\{ \Lambda_l^{C_i} : \Lambda_l^{C_i} \text{ is a parameter of } C_i \right\} \forall C_i \in C \right\}$,
 $\Upsilon = \langle \Upsilon_i : \Upsilon_i \text{ is a variation operator} \rangle$,
 $\Lambda^\Upsilon = \left\{ \left\{ \Lambda_l^{\Upsilon_i} : \Lambda_l^{\Upsilon_i} \text{ is a parameter of } \Upsilon_i \right\} \forall \Upsilon_i \in \Upsilon \right\}$,
 ς = selection function, $\Lambda^\varsigma = \{\Lambda_l^\varsigma : \Lambda_l^\varsigma \text{ is a parameter of } \varsigma\}$,
 ι = initializing function, $\Lambda^\iota = \{\Lambda_l^\iota : \Lambda_l^\iota \text{ is a parameter of } \iota\}$,
 ρ = ranking function

```

begin
   $X \leftarrow \emptyset$  // population
   $g \leftarrow 1$  // first generation
   $X_g \leftarrow \iota(\mu, \Lambda^\iota)$  // population initialization
   $X_g^E \leftarrow \langle (\overrightarrow{X}_{g,i}, \vec{f}(\overrightarrow{X}_{g,i}, \Lambda^{\vec{f}})) : 1 \leq i \leq \mu \rangle$  // evaluated population
  // while termination criteria not met
  while  $C_i(X_g^E, \Lambda^{C_i} \in \Lambda^C) \neq \text{true} \forall \{C_i : C_i \in C\}$  do
     $X_g^{IE} \leftarrow X_g^E$  // evaluated candidate offspring
    forall  $\langle \Upsilon_i^{KB} : \Upsilon^{KB} \subset \Upsilon, 1 \leq i \leq |\Upsilon^{KB}| \rangle$  do // knowledge based
       $X^{KB} \leftarrow \Upsilon_i^{KB}(X_g^{IE}, \Lambda^{\Upsilon_i^{KB}} \in \Lambda^\Upsilon)$ 
       $X_g^{IE} \leftarrow X_g^{IE} \parallel \langle (\overrightarrow{X}_j^{KB}, \vec{f}(\overrightarrow{X}_j^{KB}, \Lambda^{\vec{f}})) : 1 \leq j \leq |X^{KB}| \rangle$ 
    end
    // classical evolutionary
    forall  $\langle \Upsilon_i^{CE} : \Upsilon^{CE} \subset \Upsilon, 1 \leq i \leq |\Upsilon^{CE}| \rangle$  do
       $X^{CE} \leftarrow \Upsilon_i^{CE}(X_g^{IE}, \Lambda^{\Upsilon_i^{CE}} \in \Lambda^\Upsilon)$ 
       $X_g^{IE} \leftarrow X_g^{IE} \parallel \langle (\overrightarrow{X}_j^{CE}, \vec{f}(\overrightarrow{X}_j^{CE}, \Lambda^{\vec{f}})) : 1 \leq j \leq |X^{CE}| \rangle$ 
    end
     $X_{g+1}^E \leftarrow \varsigma(\mu, X_g^{IE}, \Lambda^\varsigma)$  // next generation
     $g \leftarrow g + 1$ 
  end
   $X_g^R = \rho(X_g^E)$  // ranked population
  return  $X_{g,1}^R$  // fittest
end

```

4.5.1 Top-down initialization

Population X_1 is initialized by generating μ design concepts through an initializer function ι based on the hierarchical/multi-layer structure of the system previously defined.

Each new design is created by initializing every design variable to a randomly generated value within its corresponding domain at a high level of abstraction layer (commonly the highest level).

Afterwards, for each generated design a new one is created (with the same method) for the next lower abstraction layer, this time, the domains used to generate the values of the design

variables are restricted to the context of the parent level of abstraction.

This method is repeated for each abstraction layer down to the most concrete, where the generated designs become the actual initial population of the strategy.

Initialization parameters Λ^l allow achieving one or a combination of the following effects:

Representative population

Generation of designs at one or more abstraction layers might be restricted to generate unique designs for a cluster or class implicitly defined in the system structure, that is, if an abstraction layer contains a categorical design variable specifically defined for classification purposes, the initialization process will generate designs without allowing duplicate values for that variable, hence the resulting designs will be representatives of a class (representative building block).

Usually, only the most abstract layer needs to be restricted to obtain a representative population.

This mechanism allows to generate a kind of evenly distinct designs (representatives) that might compensate some drawbacks of having very small populations.

Bounded initial design space region

The initialization process can be configured to assign preset values to specific design variables at any abstraction level.

This feature allows to bound the initial design space to a specific region (base designs) where the strategy will start to search from.

This parametrization can also be useful to test different initialization variants specific to the application domain without requiring changing the variables domains in the knowledge base.

Remark. The initialization procedure describes the method proposed by this work; however, it is up to the designer or researcher to pick the upper abstraction layer, the lowest and even the intermediate layers from the system structure defined.

This could be useful if one needs to work at some specific abstraction layer of the system.

The initialization procedure can even be reduced to a classical random population initialization if only one layer is selected and no representative population nor preset values parameters are provided.

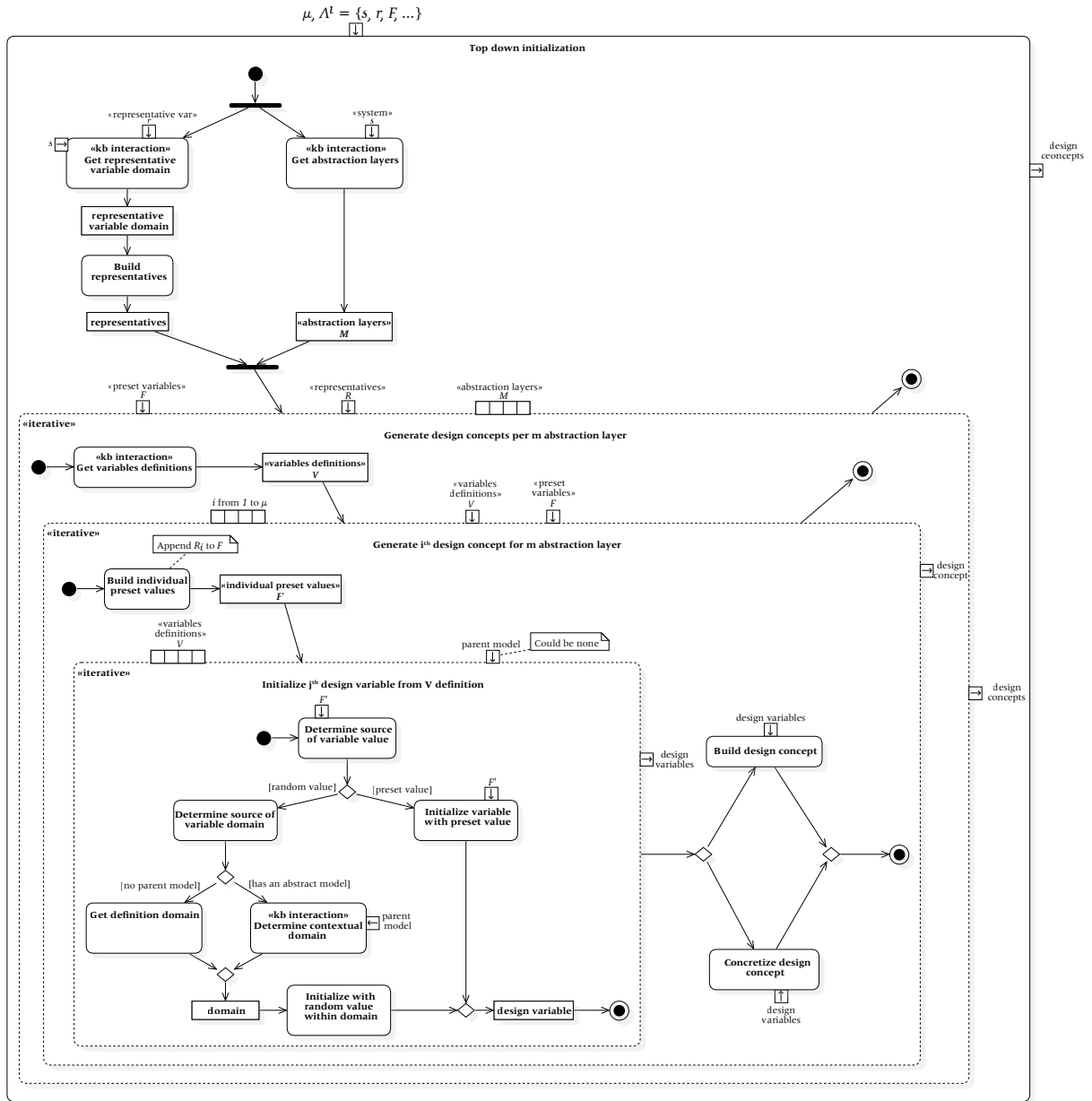


Figure 4.19: Top-down initialization activity diagram

Algorithm 4.2: Top-down initialization

```

Parameters:  $\mu$  = population size,
                $\Lambda^t = \{\text{system } s, \text{ representative var } r, \text{ fixed values } F, \dots, \Lambda_n^t\}$ 
Preconditions:  $|r \rightarrow \text{domain}| \geq \mu$ 
begin
   $Y \leftarrow \emptyset$  // design concepts
   $R \leftarrow \text{domain\_of?}(s, r)$  // representatives
   $R \leftarrow \text{random\_sample}(R, \mu)$  // what if  $|R| > \mu$ 
  forall  $m \in \text{models\_of?}(s)$  do // abstraction layers
     $V \leftarrow \text{variables\_of?}(s, m)$  // design variable definitions sets
    for  $i \leftarrow 1$  to  $\mu$  do
       $Y' \leftarrow \emptyset$ 
       $F' \leftarrow F \parallel R_i$ 
      forall  $v \in V$  do
        if  $v \in F'$  then
           $Y'_v \leftarrow F'_v$ 
        else
          if  $m \neq \text{highest}$  then
            // context = higher abstraction levels concepts
             $d \leftarrow \text{domain\_of?}(s, m, Y_i)$ ;
          else
             $d \leftarrow v[\text{domain}]$ 
          end
           $Y'_v \leftarrow \text{random\_within}(d)$ 
        end
      end
       $Y_i \leftarrow Y'$ 
    end
  end
  return  $Y$ 
end

```

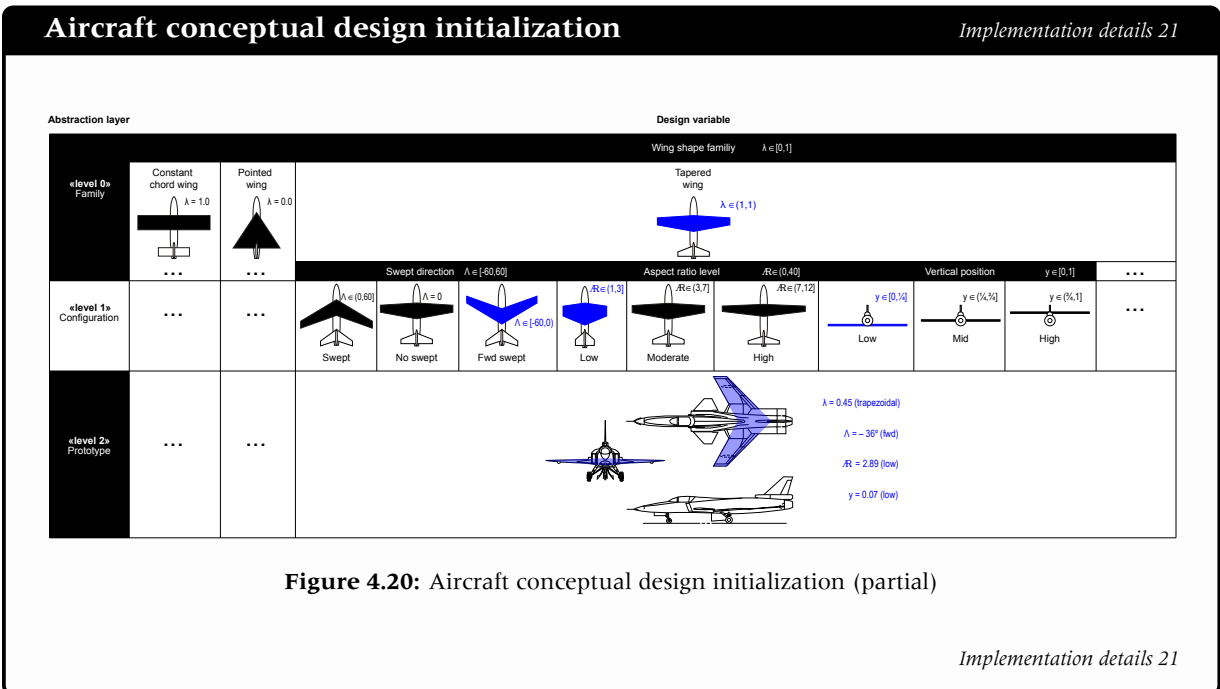


Figure 4.20: Aircraft conceptual design initialization (partial)

Implementation details 21

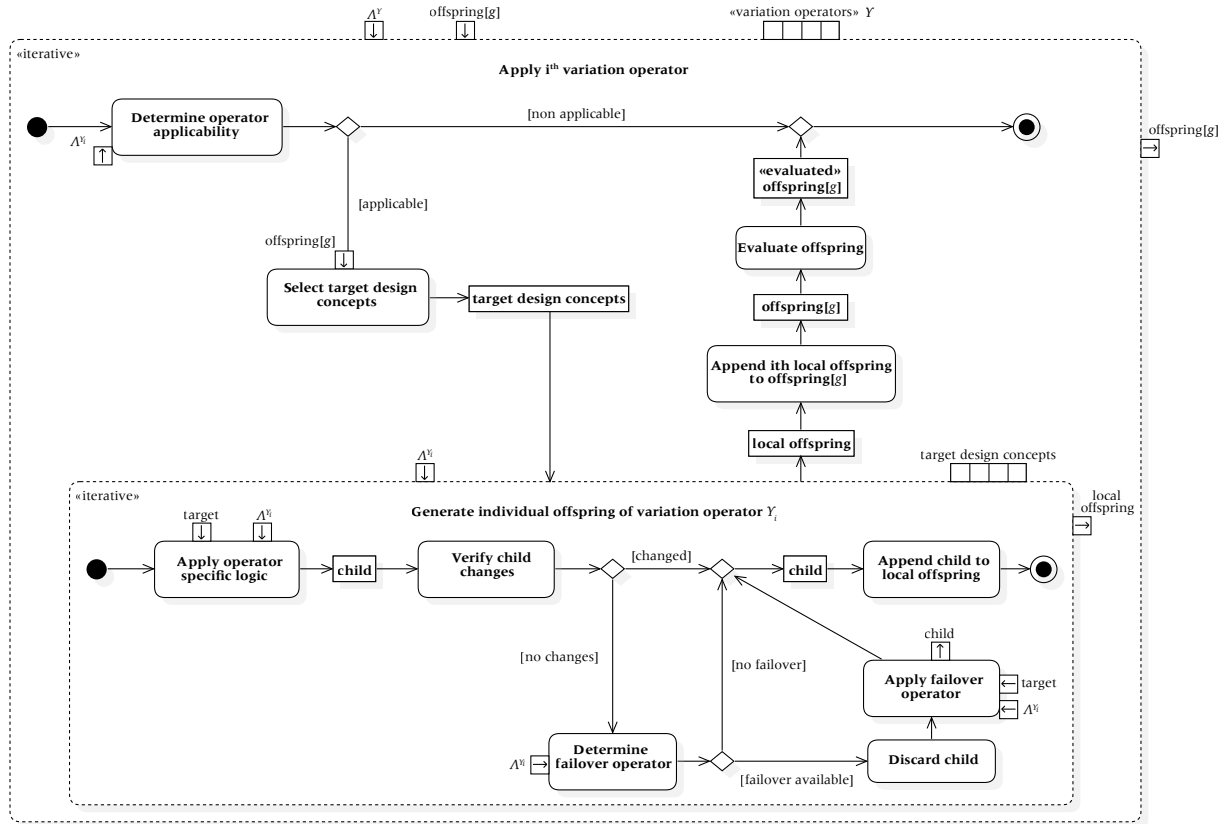


Figure 4.21: Variation operator overall activity diagram

4.5.2 Design variation

Design variation is performed by the application of a set of variation operators Υ (Fig. 4.21).

This work proposes as a design variation mechanism a combination of knowledge based Υ^{KB} variation operators and classical evolutionary Υ^{CE} variation operators (i.e., mutation and recombination) that are applied on a cumulative candidate offspring X_g^{IE} at each g generation.

When comparing the strategy proposed with a pure stochastic approach where domain knowledge is absent, the effectiveness might be affected when the number of design variables gets increased (increasing dimensionality). As the design space gets more spread the less high-order building blocks might be constructed that lead to a good solution.

On the other hand, discarding classical evolutionary variation operators may reduce the search space to a space constructed only by the potential building blocks the knowledge based variation operators may generate.

The proposed strategy provides some mechanisms (i.e., failover variation, population injection), that may help to maintain population diversity and try to keep a good balance between exploration and exploitation capabilities of actual algorithms.

Remark. In general, the strategy weighs domain knowledge slightly more than pure randomness^a.

The intuition supporting this is that knowledge based variation operators may provide a set of high level building blocks (specialization) and classical evolutionary variation ones and specially mutation usually may provide low level (or not as high) building blocks (generalization).

High level building blocks may be more effective when dealing with higher number of variables or complex systems (a high level building block is actually composed by several low level building blocks) and low level build blocks may increase the strategy exploration capabilities.

^aAlthough a specific balance can be set when crafting concrete algorithms

4.5.2.1 Cumulative candidate offspring

This work proposes a method to manage the overall offspring generation called cumulative candidate offspring.

The basic idea is to maintain a pool of design concepts X_g^{IE} generated within a generation.

The cumulative candidate offspring is initialized at the beginning of each generation by cloning (or referencing) the current evaluated population members $X_g^{IE} \leftarrow X_g^E$ (Fig. 4.18 and Algorithm 4.1).

Subsequent designs generated by each variation operator (i.e., local variation operators offspring) are appended to the candidate offspring.

Any variation operator is able to work on any subset of the candidate offspring (i.e., target selection).

The strategy triggers evaluation of the non evaluated candidate offspring members (i.e., last designs generated) after the application of each operator (and after initialization), that way, the candidate offspring may be exploited by subsequent operators (Fig. 4.21 and Algorithm 4.1).

4.5.2.2 Target selection

Variation operators specify a selection function $\sigma(X_g^{IE}, \Lambda^\sigma)$ to selectively work on candidate offspring members (target designs) by a combination of the following methods:

By order

An operator selects its target designs based on their order in which they are originally introduced into the candidate offspring no matter the quality the designs have (e.g., the last generated design, the first three generated designs, exactly the second generated one,

etc).

$$\sigma \left(X_g^{IE}, \{a..b\} \right) = \left\langle X_{g,i}^{IE} : a, b \in \mathbb{Z}^+, 1 \leq a \leq i \leq b \leq |X_g^{IE}| \right\rangle \quad (4.18)$$

By ranking

The selection of the target designs is quality based either by using a ranking function provided to the strategy $\left\{ \vec{f}(\vec{x}) : \vec{x} \in \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n\} \xrightarrow{g} \mathbb{Z}^+ \right\}$, or a custom $\varrho^\Upsilon \in \Lambda^\Upsilon$ ranking function specific to the variation operator.

An order range is also specified, not with respect to the generation order but to a ranking sort order.

$$\sigma \left(\varrho \left(X_g^{IE} \right), \{a..b\} \right) = \left\langle \varrho \left(X_g^{IE} \right)_i : a, b \in \mathbb{Z}^+, 1 \leq a \leq i \leq b \leq |X_g^{IE}| \right\rangle \quad (4.19)$$

By custom selector

The operator may provide a custom selector method, choosing designs by a combination of order, ranking or other criteria.

In addition, variation operators may specify an elitism (based on order or ranking) parameter in order to force selection of specific targets.

4.5.2.3 Operators order

In general, the strategy suggests applying the knowledge based operators first followed by the classical evolutionary operators.

The way the strategy may take advantage of the domain knowledge is to favor knowledge based operators over the classical ones (build a larger percentage of the candidate offspring based on knowledge than on pure randomness).

Classical operators, however, provide the strategy with powerful exploration effects that may help to redirect the search to regions of the design space not covered by the knowledge based generated variations, or to try improving a design concept through recombination.

Operation execution order is set by the order implicit in the variation operators list that is passed to the strategy $\langle \Upsilon_i : 1 \leq i \leq |\Upsilon| \rangle$.

4.5.2.4 Operators distribution

The common approach proposed in this strategy is to apply the full set of variation operators within each generation.

The strategy, however, allow distribution of variation operators across generations, that is, an operator Υ_i may be configured to be applied only every $g_e \in \Lambda^\Upsilon_i$ generations, starting at a specific $g_0 \in \Lambda^\Upsilon_i$ generation (e.g., apply an operator every 3 generations starting at $g = 1$).

This mechanism allow reducing the overall number of evaluations within a generation but

may introduce a stronger selection pressure in the strategy because the cumulative candidate offspring size is reduced in comparison with the “apply all” default approach.

4.5.2.5 Variation operator failover

Knowledge based variation operators proposed in this thesis have the drawback of being limited to the number of pieces of knowledge introduced into the knowledge base and its applicability for a given specific candidate design.

This situation may lead to design evaluation inefficiency. A failover mechanism is proposed to supersede this inconvenient, children designs are checked for actual variations, if no changes are detected in the design variables vector the child is discarded without being evaluated and an optional failover variation operator $\eta \in \Lambda^{\Upsilon_i}$ for a given variation operator Υ_i is used to generate the definitive child design.

The strategy uses a uniform mutation operator as default failover operator.

Remark. Although this situation is much less likely to happen with classical evolutionary operators (due to their pure randomness nature) it may happen that a child design resulted from a recombination displays no changes in its variables values, specially when changes are done on fixed parameter design variables and population is really small.

For this reason, failover variation operators usage is not restricted for knowledge based operators and actually it is a recommended feature specially for recombination operators.

4.5.2.6 Population injection

A side effect of keeping population small is the degradation of its diversity, exploitation behavior of selection makes population members resemble each other more and more across generations and, even when exploration operators may balance that effect, the explored regions might not have better designs than current population.

In order to tackle this issue an injection operator[53] $I \in \Upsilon$ is used to try to reintroduce diversity to the population.

Injection operator generates a new set of μ design concepts either by top-down initialization or any other initialization method. New designs are appended to the cumulative candidate offspring set.

The injection operator is treated by the strategy as any other variation operator. An injection threshold parameter $\tau^I \in \Lambda^I$ sets the number of generations without improvement (fittest design is not improved) before the injection is applied.

When the operator is triggered by the strategy, it first checks for an injection threshold counter (initialized to 0 when the strategy starts), if the counter value is greater or equal to the injection

threshold, then injection is performed and the counter reset back to zero.

Remark. Injection operator is the first variation operator in the sequence of variation operators to allow the injected designs being modified across the rest of operators just like the ones coming from initialization.

If injection would be applied at the end of the sequence, chances of having an enough “good” design could be marginal, wasting resources with null benefits to the population diversity.

4.5.2.7 Knowledge based variation operators

In classical stochastic evolutionary approach, a candidate design is modified by applying variation operators that heavily rely on randomness, a desired behavior indeed that usually set a trade-off between population size and number of generations, that is, small populations may require several generations to converge (on average) and few generations would necessitate larger populations.

Any of these situations have a strong computational cost impact in the direct simulation-driven approach.

This is, among others, a key motivation to propose domain knowledge exploitation through the evolutionary process.

Definition 4.1. A knowledge based variation operator modifies a design concept by altering its design variables according to knowledge facts and/or inferred decisions, and considering the current design configuration or performance.

Unlike classical evolutionary operators, randomness is not a core aspect but might be present in some degree.

This work proposes two types of knowledge based variation operators: preconfigured and corrective variation operators, they both require domain knowledge in some degree and provide different effects and application opportunities.

4.5.2.7.1 Preconfigured variation operator

Definition 4.2. A preconfigured variation operator is a knowledge based variation operator that modifies a candidate design by applying a structured change in one or more specific design variables in order to shape the system in a predictable form or appearance.

Values assigned to the variables when crafting a preconfigured variation are either fixed, calcu-

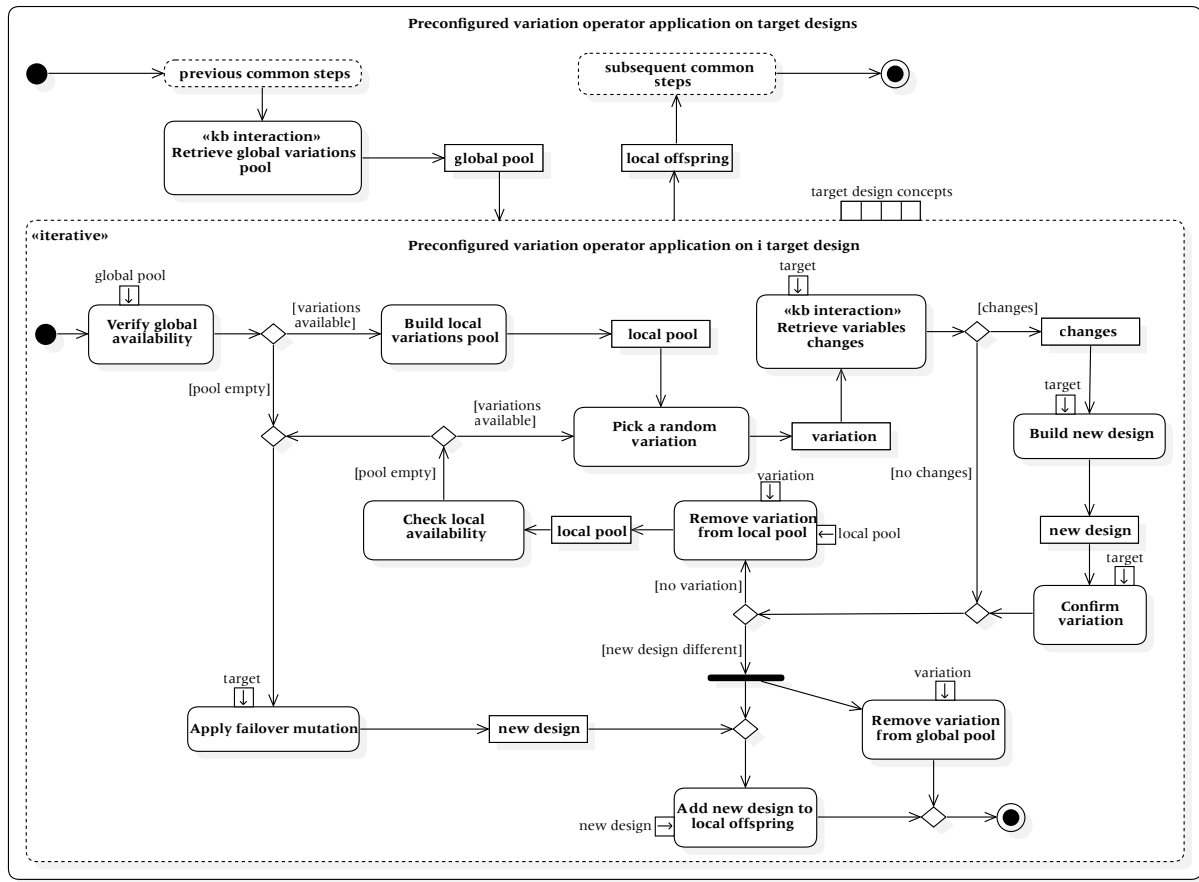


Figure 4.22: Preconfigured variation operator application activity diagram

lated or random in some degree.

Other preconfigured variations can be used as building blocks of more complex variations.

Remark. There might be an approximate foreseeable effect in performance if the variation configuration is well-known in the domain field but it is not its primary purpose.

The (Fig. 4.21) diagram depicts operation procedure of this operator:

A set of preconfigured variations is introduced into the knowledge base either by facts, rules or a combination of both (or their equivalent knowledge representation mechanisms).

Every time the operator is applied to a set of target designs, the set of preconfigured variations is retrieved from the knowledge base as the preconfigured variations global pool.

For each target design the global pool, if not empty, is cloned into a local pool and a new design is created by applying a randomly selected preconfigured variation.

Actual changes are supplied by domain knowledge based on current design and the knowledge

encapsulated in the variation.

If the knowledge base is unable to generate changes from current configuration or if when supplied, their application do not generate a new design that is different from the original one, the new design is discarded, the failed variation is removed from the local pool and a new preconfigured variation is picked randomly from the local pool until changes are detected in the child design or the pool gets exhausted.

When a new design is different from its ancestor it is appended to the operator local offspring and the applied variation is removed from the global pool, preventing the variation from being applied to another target design within the same generation.

Whether the local pool or generation pool gets exhausted, a failover variation operator (e.g., a classical discrete mutation) is applied to generate a child and it is added to the generation offspring.

This operator is a pure exploratory mechanism (although it cannot be named mutation[26]), that basically relies on randomness but at higher levels of abstraction that may speed convergence dramatically compared to full variable by variable randomness, with the drawback of having an exploration space delimited by the quantity and type of the variations pool.

During the development of this strategy and its implementation, some patterns are identified:

Direct type

A variation that involves direct changes to variables without any inference performed by rules.

The impact in structure and performance of this type of variation depends on the number of variables affected.

There could be a variation of just one variable and variations involving several variables (one could even change all of them).

Planned type

A variation that make changes to variables in a more analytical manner, that is, some variable changes are planed based on the current configuration of the system.

Swap type

A variation that swaps over preconfigured variations of the same type, that is, variations that modify the same system structure aspect.

This variation is applied by determining the actual configuration of the system and then picking a variation that makes the design taking the next configuration of a given aspect.

This kind of variation may induce high level of abstraction step changes in specific aspects of the system.

On/off type

A variation that may induce dramatic variations in performance by installing or removing a component or a feature.

Resulting effects depend on the level the affected element occupy in the system structure hierarchy.

Remark. As already mentioned, the main purpose of the preconfigured variation operator is to provide knowledge based exploration capabilities to the strategy; however, this operator can provide to human designers or researchers a mechanism to:

- Implement their own design ideas.
- Introduce existing configurations.
- Test synthetic configurations probably generated by an intelligent agent.
- Evaluate existing configurations identified by automatic clustering.
- Obtain possible optimal designs by considering all these alternatives.

Algorithm 4.3: Preconfigured variation operator

```

Parameters:  $X_g^{IE}$  = cumulative evaluated candidate offspring,
 $\Upsilon_i^{KB} = \{ \text{system } s, \text{ target selection function } \sigma, \text{ failover operator } \eta, \dots, \text{ custom parameter } \Lambda_n^{\Upsilon_i^{KB}} \}$ 

begin
   $Y \leftarrow \emptyset$  // local offspring
   $G \leftarrow \text{variations\_for?}(s)$  // global variations pool
   $Z \leftarrow \sigma(X_g^{IE})$  // target design concepts
  for  $i \leftarrow 1$  to  $|Z|$  do
    if  $|G| > 0$  then // global pool not exhausted
       $L \leftarrow G$  // local variations pool
       $Y_i \leftarrow Z_i$  // new design concept
      repeat
         $l \leftarrow \text{random}(L)$  // variation
         $\text{configured\_variation\_for}(l, s, X_i)$ 
        forall  $v \in Y_i$  do  $\text{variable\_value\_of}(v, s, Y_i)$ 
         $V \leftarrow \text{new\_variables\_values\_for?}(s, Y_i)$  // changes
        if  $|V| > 0$  then forall  $v \in V$  do  $Y_{i,v} \leftarrow v$ 
      if  $Y_i = Z_i$  then  $L \rightarrow \text{remove}(V)$ 
      until  $Y_i \neq Z_i \vee L = \emptyset$  // new design different of target design and
        local pool not exhausted

      if  $Y_i = Z_i$  then
         $Y_i \leftarrow \eta(\{Z_i\})$ 
      else
         $G \rightarrow \text{remove}(V)$  // prevents using same variation
      end
    else
       $Y_i \leftarrow \eta(\{Z_i\})$ 
    end
  end
  return  $Y$ 
end

```

Preconfigured variations*Implementation details 22*

For the fixed-wing aircraft conceptual design 34 preconfigured variations are implemented, the following are some representative examples:

Monoplane variation (direct type)

A preconfigured variation that simply sets the number of wing planes to 1 (Fig. 4.23), that is, if the aircraft had 2 or 3 main wing planes the second and third ones are removed from the design. The single remaining wing plane maintains its current configuration.

Rule 1

```
// Set to 1 the number of wing planes
```

```
If: configured_variation_for?(x) = monoplane Then:
```

```
| new_value_of(1, Nw)
```

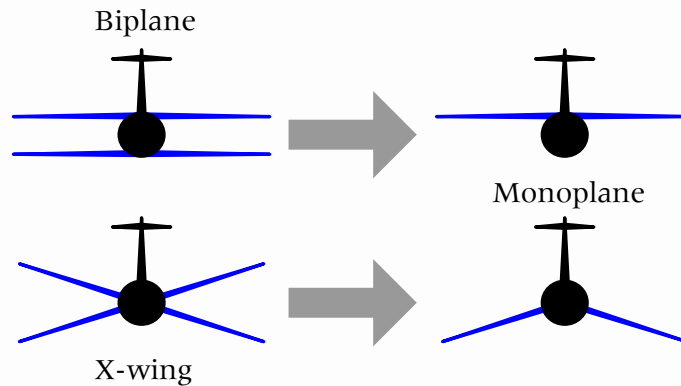


Figure 4.23: Monoplane preconfigured variation example

Uninstall vertical stabilizer (direct, on/off type)

A preconfigured variation that removes the vertical stabilizer (if any) from the aircraft (Fig. 4.24).

Rule 1

```
// Set to 0 the number of vertical stabilizers
```

```
If: configured_variation_for?(x) = uninstall vertical stabilizer Then:
```

```
| new_value_of(0, Nv)
```

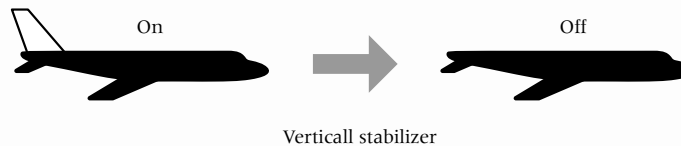


Figure 4.24: Vertical stabilizer on/off preconfigured variation example

Swap swept direction (swap type)

A swap type preconfigured variation that change the swept direction of 0 or more wing planes of the aircraft (Fig. 4.25).

It starts by defining the number of wings being potentially affected by generating a random integer n between 1 and the current number of wing planes installed on the aircraft. Then it traverses the n wings and randomly (50%, 50%) decides if swept direction of that particular wing plane should be reversed or stays as is. If swap should be performed and the wing plane already has a sweep angle (it is not straight) the direction is simply reversed.

```

Rule 1
// Defines the number of wing planes being visited
If: configured_variation_for?(x) = swap swept direction and value_of?(done)
≠ true Then:
| value_of(random({1..value_of?(Nw)}), n)
| value_of(1, w) // visit 1st wing
Rule 2
// swap swept direction in a wing if no sweep angle or forward swept
If: configured_variation_for?(x) = swap swept direction and value_of?(n) > 0
and random([0,1]) ≤ 0.5 and value_of?(Λ[ value_of?(w) ]) ≠ 0.0 and
value_of?(done) ≠ true Then:
| value_of(-value_of?(Λ[ value_of?(w) ]), Λ[ value_of?(w) ])
| value_of(value_of?(n) - 1, n)
| value_of(value_of?(w) + 1, w) // visit next wing
Rule 3
// Stop swaping swept direction
If: configured_variation_for?(x) = swap swept direction and value_of?(n) = 0
and value_of?(done) ≠ true Then:
| value_of(true, done)

```

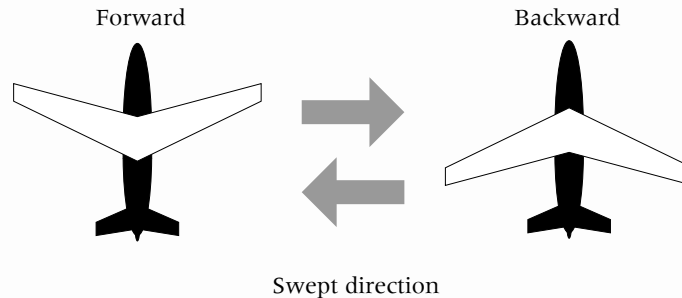


Figure 4.25: Swap swept direction preconfigured variation example

X-wing variation (planned type)

A planned preconfigured variation that arrange 2 main wing planes in a way they resemble a letter X if observed from the front (Fig. 4.26).

It starts by setting the number of wing planes to 2 (if the aircraft happens to have 1 wing the second one is randomly generated by setting values within allowed domains, if the aircraft had 3 wings, the third one is discarded), then, a reference wing is randomly chosen between the already existing 2 wing planes (the reference could then be a priors wing plane or the just generated one.^a).

The next step is to make the 2 wings appear as twins wings by replicating the reference wing plane configuration into the other wing.

Finally, the X pattern is assembled by making one of the wings point down and the other point up and centering the intersection of both wings at the middle of the fuselage.

Longitudinally, the wings are located wherever the reference wing is already positioned.

```

Rule 1
// Set to 2 the number of wing planes
If: configured_variation_for?(x) = x-wing Then:
| new_value_of(2, Nw)
Rule 2
// Randomly determines a wing plane reference a
If: configured_variation_for?(x) = x-wing Then:
| value_of(random({1..2}), a)
| value_of(2- value_of?(a) +1, b)
Rule 3
// Build twin wing planes based on reference a wing plane
If: configured_variation_for?(x) = x-wing Then:
| value_of(value_of?(Zpos[ value_of?(a) ]), Zpos[ value_of?(b) ])
| // relative longitudinal position
| value_of(value_of?(R[ value_of?(a) ]), R[ value_of?(b) ]) // aspect
| ratio
| value_of(value_of?(A[ value_of?(a) ]), A[ value_of?(b) ]) // sweep
| angle
| // sweep angle location
| value_of(value_of?(Al[ value_of?(a) ]), Al[ value_of?(b) ])
| value_of(value_of?(λ[ value_of?(a) ]), λ[ value_of?(b) ]) // taper
| ratio
| value_of(value_of?(i[ value_of?(a) ]), i[ value_of?(b) ]) // incidence
| value_of(value_of?(ε[ value_of?(a) ]), ε[ value_of?(b) ]) // twist
| // airfoil camber
| value_of(value_of?( $\frac{\delta_{max}}{c}$ [ value_of?(a) ]),  $\frac{\delta_{max}}{c}$ [ value_of?(b) ])
| // airfoil camber location
| value_of(value_of?( $\frac{x\delta_{max}}{c}x$ [ value_of?(a) ]),  $\frac{x\delta_{max}}{c}x$ [ value_of?(b) ])
| // airfoil thickness
| value_of(value_of?( $\frac{t_{max}}{c}$ [ value_of?(a) ]),  $\frac{t_{max}}{c}$ [ value_of?(b) ])
Rule 4
// Arrange wing planes in the particular X pattern
If: configured_variation_for?(x) = x-wing Then:
| // anhedral (wing pointing down
| value_of(random({-30..-18}), Γ[ value_of?(a) ])
| // dihedral (wing pointing up in the same magnitude)
| value_of(-value_of?(Γ[ value_of?(a) ]), Γ[ value_of?(b) ])
Rule 5
// Align center of X to the middle of the fuselage
If: configured_variation_for?(x) = x-wing Then:
| value_of(0.5, y_w[ value_of?(a) ]) // relative vertical position
| value_of(value_of?(y_w[ value_of?(a) ]), y_w[ value_of?(b) ])

```

^aNote that despite the structured variation it displays some randomness in some of its components

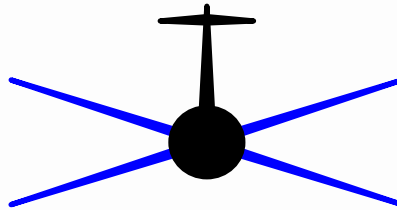


Figure 4.26: X-wing preconfigured variation example

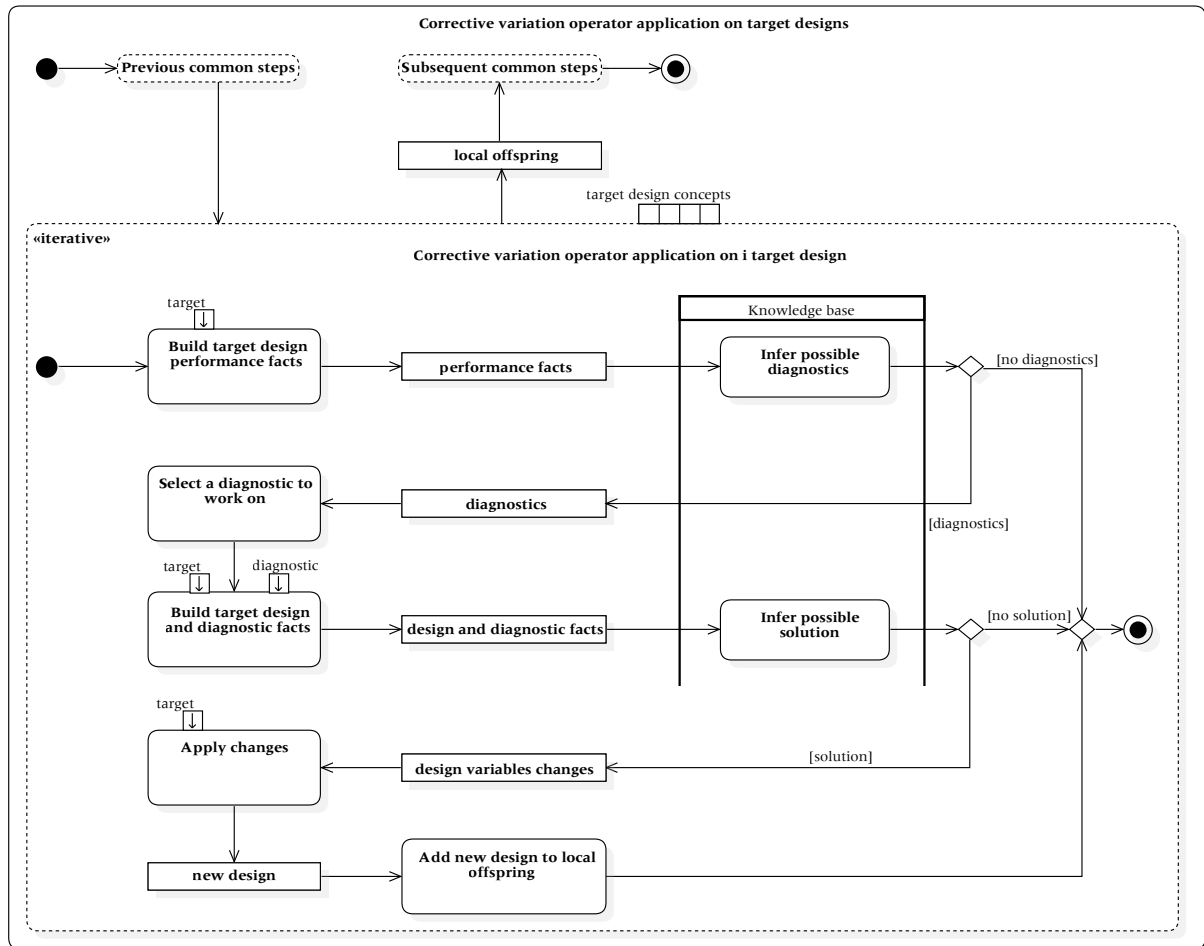


Figure 4.27: Corrective variation operator application activity diagram

Implementation details 22

4.5.2.7.2 Corrective variation operator

Definition 4.3. A corrective variation operator is a knowledge based variation operator that, based on inferred diagnostic, modifies a candidate design by applying a calculated change in one or more specific design variables, providing an approximate, not guaranteed predictable effect in performance.

The (Fig. 4.27) diagram depicts operation procedure of this operator:

A set of corrective variations is introduced into the knowledge base by defining two types of rules: diagnostic and fix rules.

Diagnostic rules contain the knowledge required to analyze a system configuration and its detailed fitness (per objective), and propose a possible diagnostic.

Several rules may lead to the same designation of possible causes of a design failure (i.e., diagnostic) and more than one diagnostic might be inferred for a design.

In order to help the strategy decide which diagnostic work on, a priority is set when entering the rules into the knowledge base.

On the other hand, fix rules encapsulate domain knowledge based solution alternatives required to “heal” the system with a given diagnostic.

These rules also analyze the actual system configuration to properly derive required changes.

During the strategy execution, diagnostic rules are resolved for each target design to get a list of applicable diagnostics. The strategy selects the priority diagnostic and retrieves from the knowledge base a set of changes generated by relevant fix rules.

The set of changes is used to generate a child design that, hopefully would be an improved version of its parent.

A failover variation operator, commonly a uniform mutation operator is used to generate the child in the case of “no diagnostics” are found in the system or if any, there would not be any fix rule applicable.

The primary purpose of this operator is to try to improve a design by “fixing failures”, being a special type of variation operator displaying a kind of exploitation effect[25]; however, if changes do not provide expected results in performance the new design may still help the evolutionary process by introducing diversity.

The effectiveness of this operator directly depends on the size of the set of diagnostic/fix rules, or pieces of knowledge introduced into the design process, and the variety of aspects covered by the set of rules.

Regardless the main application of this type of operator is to improve candidate designs; it can be a useful tool in the design process and help designers and researches to validate hypotheses (in some degree and depending on the simulation tool capabilities) or to induce specific performance in the designs.

Algorithm 4.4: Corrective variation operator

Parameters: X_g^{IE} = cumulative evaluated candidate offspring,
 $\Lambda_i^{KB} = \{ \text{system } s, \text{ target selection function } \sigma, \text{ objectives } O, \dots, \text{ custom parameter } \Lambda_n^{KB} \}$

```

begin
   $Y \leftarrow \emptyset$  // local offspring
   $Z \leftarrow \sigma(X_g^{IE})$  // target design concepts
  for  $i \leftarrow 1$  to  $|Z|$  do
     $Y_i \leftarrow Z_i$  // new design concept
    forall  $o \in O$  do  $\text{fitness\_of}(Y_i \rightarrow \text{fitness}_o, s, o, Y_i)$ 
    forall  $v \in Y_i$  do  $\text{variable\_value\_of}(v, s, Y_i)$ 
     $D \leftarrow \text{diganostics\_of?}(s, Y_i)$  // possible prioritized diagnostics
    if  $|D| > 0$  then
       $d \leftarrow \exists d_i \in D: d_i \rightarrow \text{priority} = \max_{1 \leq i \leq |D|} d_i \rightarrow \text{priority}$  // diagnostic to
      work on
       $\text{diagnostic\_for}(d, s, Y_i)$ 
       $V \leftarrow \text{new\_variables\_values\_for?}(s, Y_i)$ 
      forall  $v \in V$  do  $Y_{i,v} \leftarrow v$ 
    end
  end
  return  $Y$ 
end

```

Corrective variations

Implementation details 23

For the experimental implementation of the strategy, 2 diagnostic types are defined: lateral instability (Fig. 4.28) and directional instability.

For the lateral instability diagnostic, 7 possible solutions are modeled and 2 for the directional instability diagnostic.

The following are some representative examples:

Diagnostic rules**Rule 1**

If: $\text{fitness_of?}(\text{roll variance}, x) \geq 0.05$ **Then:**
 | $\text{diagnostic_for}(\text{lateral instability}, 1, x)$

Rule 2

If: $\text{fitness_of?}(\text{heading variance}, x) \geq 0.05$ **Then:**
 | $\text{diagnostic_for}(\text{directional instability}, 2, x)$

Fix rules

Rule 1

If: `diagnostic_for?(x) = lateral instability` Then:
 | `strategy_for(increase dihedral effect, lateral instability)`

Rule 2

If: `strategy_for?(x) = increase dihedral effect and not has positive dihedral` Then:
 | `action_for(set dihedral angle, increase dihedral effect)`

Rule 3

If: `strategy_for?(x) = increase dihedral effect and has positive dihedral < 15` Then:
 | `action_for(increase dihedral angle, increase dihedral effect)`

Rule 4

If: `strategy_for?(x) = increase dihedral effect and has positive dihedral and has low wing` Then:
 | `action_for(set no dihedral angle and set mid wing, increase dihedral effect)`

Rule 5

If: `strategy_for?(x) = increase dihedral effect and has positive dihedral and has mid wing` Then:
 | `action_for(set no dihedral angle and set high wing, increase dihedral effect)`

Rule 6

If: `strategy_for?(x) = increase dihedral effect and has positive dihedral and has high wing` Then:
 | `strategy_for(swept back, increase dihedral effect)`

Rule 7

If: `strategy_for?(x) = swept back and not wing has sweep` Then:
 | `action_for(set sweep back angle, swept back)`

Rule 8

If: `strategy_for?(x) = swept back and has forward sweep` Then:
 | `action_for(invert sweep angle, swept back)`

Rule 9

If: `strategy_for?(x) = swept back and has swept back < 25` Then:
 | `action_for(increase sweep angle, swept back)`

Rule 10

If: `action_for?(x) = set dihedral angle` Then:
 | `new_value_of(1, $\Gamma[0]$)`

Rule 11

If: `action_for?(x) = increase dihedral angle` Then:
 | `new_value_of(value_of?($\Gamma[0]$) + 1, $\Gamma[0]$)`

Rule 12

If: `action_for?(x) = set sweep back angle` Then:
 | `new_value_of(random({7..25}), $\Lambda l[0]$) and new_value_of(0.25, $\Lambda l[0]$)`

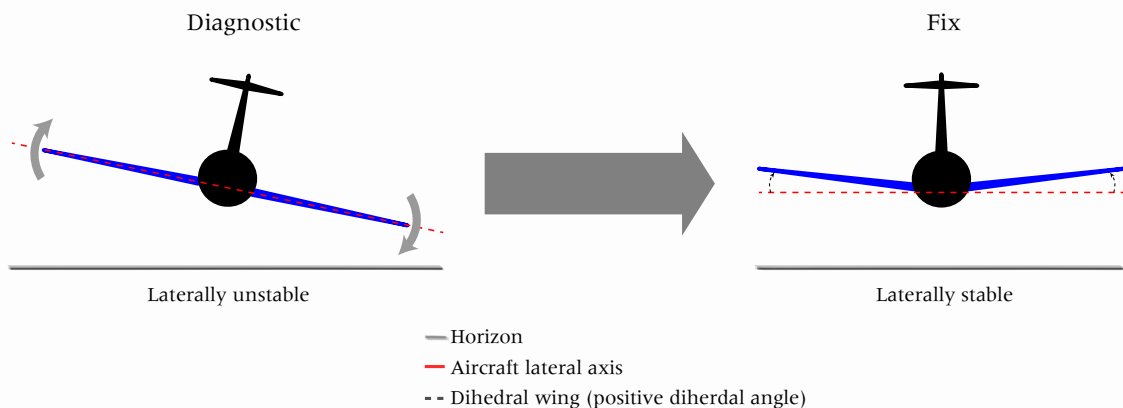


Figure 4.28: Corrective variation example

4.5.2.8 Classical evolutionary variation operators

The strong evolutionary stochastic nature to modify design concepts in the strategy relies on the application of two types of operators: recombination and mutation.

As reviewed in the theoretical framework chapter, it has been demonstrated that recombination and mutation operators tend to display different survival and constructive effects as a result of the level of disruption they introduce to the evolutionary process and the level of their exploratory power.

Regarding mutation and recombination, the strategy key interest is on the overall effect those operators may introduce to the design process, the operator capabilities to work with the particular solution representation used in this problem and the arity of the operators, considering the very small population size.

Recombination operator

The strategy builds a cumulative candidate offspring within a generation. For this reason, the recombination operator is applied once after all knowledge based operators, since a recombination operator may be more efficient and effective when having more parent candidates available to choose from.

Mutation operator

Mutation operator is applied only once within a generation, specifically taking as a target the resulting recombinant design; so basically this is the last operator to be applied within a generation.

The strong exploratory capabilities of this operator makes it also ideal to be used as a failover variation operator for either knowledge based operators or the recombination operator.

Recombination operator

Implementation details 24

A custom —differential mutation/arithmetical recombination/uniform recombination— operator is used as recombination operator.

The operator may generate either a mutant or a recombinant (although for simplicity it is called recombinant).

Recombination is favored over mutation, a resulting design could be a fully arithmetical recombination of three existing designs (might include the target itself), or a uniform binary recombination between the original design and a mutant resulted of a mutation step calculated by the difference between two designs.

The operator is inspired by both, classical differential mutation/recombination operator and “either-or” variant of DE algorithm, taking advantage of the 3 arity of the original operators in order to combine or mutate up to three designs.

The following parameters Λ_i^{rCE} are passed to the recombination operator, most of them are implemented as functions rather than fixed value parameters in order to enable customization:

Parent selection function \dot{r}

Selects the $\dot{r}1$, $\dot{r}2$ and $\dot{r}3$ position indexes of the recombination parents (as defined by the classical differential mutation operator) in the target design concepts set.

As other selection functions used in the strategy this function is able to select parents by order, ranking or a custom method.

Differential mutation scale factor generator function F

$$F(\alpha_F, \beta_F) = \text{random}([\alpha_F, \beta_F]) \quad (4.20)$$

where $\alpha_F, \beta_F \in [0, 1]$ and $\alpha_F < \beta_F$

Uniform recombination probability generator function Cr

$$Cr(\alpha_{Cr}, \beta_{Cr}) = \text{random}([\alpha_{Cr}, \beta_{Cr}]) \quad (4.21)$$

where $\alpha_{Cr}, \beta_{Cr} \in [0, 1]$ and $\alpha_{Cr} < \beta_{Cr}$

Differential mutation probability generator function Pf

$$Pf(c_{Pf}) = c_{Pf} \quad (4.22)$$

where $c_{Pf} \in [0, 1]$

Scale factor sampling frequency flag m_F

Where $m_F \in \{\text{true}, \text{false}\}$. If true, the scale factor is computed for each target design concept (dither method) being modified by the operator, if false (jitter method), the scale factor is computed for each design variable of a target design concept.

Recombination probability sampling frequency flag m_{Cr}

Where $m_{Cr} \in \{\text{true}, \text{false}\}$.

Domain combination method flag d_X

Where $d_X \in \{\text{true}, \text{false}\}$. If true (union), when combining design variables their resulting domains are calculated as the union \cup of the domains of the donor design concepts, if false (full), the resulting domain takes the value of the full domain for that variable in the domain knowledge.

The custom recombination operator being applied on a set of target design concepts is specified as follows:

Algorithm 4.5: Custom recombination operator

Parameters: X_g^{IE} = cumulative evaluated candidate offspring,
 $\Lambda_n^{\Upsilon^{CE}}$ = { differential mutation scale factor function $F(\alpha_F, \beta_F)$,
uniform recombination probability generator function $Cr(\alpha_{Cr}, \beta_{Cr})$,
parent selection function \dot{r} ,
target selection function σ ,
differential mutation probability function $Pf(c_{Pf})$,
differential mutation scale factor sampling frequency m_F ,
uniform recombination probability sampling frequency flag m_{Cr} ,
domain combination method flag d_X ,
..., custom parameter $\Lambda_n^{\Upsilon^{CE}}$ }

```

begin
  Y ← ∅ // local offspring
  Z ← σ(X_g^{IE}) // target design concepts
  for i ← 1 to |Z| do
    Y_i ← Z_i // new design concept (clone)
    mutate = true
    if random([0, 1]) > Pf(c_{Pf}) then mutate = false
    if m_{Cr} = true then Cr' = Cr(α_{Cr}, β_{Cr}) // dither
    if m_F = true then F' = F(α_F, β_F) // dither
    for j ← 1 to |Y_i| do
      if m_{Cr} = false then Cr' = Cr(α_{Cr}, β_{Cr}) // jitter
      if m_F = false then F' = F(α_F, β_F) // jitter
      if random([0, 1]) ≤ Cr' ∨ j = random({1..|Y_i|}) then
        r1, r2, r3 = ṙ(Y_{i,j}) // selection of parents order
        if d^F = true then // union
          | Y_{i,j}[domain] ← Y_{r1,j}[domain] ∪ Y_{r2,j}[domain] ∪ Y_{r3,j}[domain]
        else // full domain
          | Y_{i,j}[domain] ← Y_{i,j}[meta][fulldomain]
        end
        if mutate = true then // differential mutation
          | Y_{i,j} ← Y_{r1,j} + F' * (Y_{r2,j} - X_{r3,j})
        else // differential arithmetical recombination
          | Y_{i,j} ← Y_{r1,j} + (κ * (Y_{r2,j} + Y_{r3,j} - (2 * Y_{r1,j})))
        end
      end
    end
  end
end
return Y
end

```

Implementation details 24

Mutation operator

Implementation details 25

Uniform random mutation operator is implemented for the particular aircraft design case.

In addition to the common parameters passed to variation operators through this strategy (i.e., target selection function σ and its parameters and cumulative candidate offspring X_g^{IE}), the following parameters are defined:

Mutation probability generator function Mp

$$Mp(\alpha_{Mp}, \beta_{Mp}) = \text{random}([\alpha_{Mp}, \beta_{Mp}]) \quad (4.23)$$

where $\alpha_{Mp}, \beta_{Mp} \in [0, 1]$ and $\alpha_{Mp} < \beta_{Mp}$

Mutation probability sampling frequency flag m_{Mp}

Where $m_{Mp} \in \{\text{true}, \text{false}\}$. If true, dither method is applied to compute the mutation probability, if false jitter method is used instead.

Domain scope flag d_M

Where $d_M \in \{\text{true}, \text{false}\}$. If true, the new random value of a design variable is generated within the current variable domain, if false the full variable domain is taken to generate the random value.

The mutation operator being applied on a set of target design concepts is specified as follows:

Algorithm 4.6: Mutation operator

Parameters: X_g^{IE} = cumulative evaluated candidate offspring,
 Λ_i^{rCE} = { mutation probability generation function $Mp(\alpha_{Mp}, \beta_{Mp})$,
mutation probability sampling frequency flag m_{Mp} ,
target selection function σ ,
domain scope flag d_M ,
..., custom parameter Λ_n^{rCE} }

```

begin
  Y ← ∅ // local offspring
  Z ← σ(X_g^{IE}) // target design concepts
  for i ← 1 to |Z| do
    Y_i ← Z_i // new design concept (clone)
    if m_{Mp} = true then Mp' = Mp(α_{Mp}, β_{Mp}) // dither
    for j ← 1 to |Y_i| do
      if m_{Mp} = false then Mp' = Mp(α_{Mp}, β_{Mp}) // jitter
      if random([0, 1]) ≤ Mp' then
        // full domain
        if d_M = false then Y_{i,j}[domain] ← Y_{i,j}[meta][fulldomain]
        Y_{i,j} = random(Y_{i,j}[domain])
      end
    end
  end
end
return Y
end

```

Implementation details 25

4.5.3 Generation selection

After the set of variation operators is applied on the current population and cumulative candidate offspring, a selection operator (provided to the strategy as a parameter) selects from the full candidate offspring μ , designs concepts according to criteria enclosed in the selector.

Selected μ designs become the actual g generation offspring and pass to the next generation as the $g + 1$ generation population.

In the same way as variation operators, the selection operator is able to select design concepts by ordinal order, ranking sort order and elitism to support its specific selection algorithmic logic.

Generation selection operators

Implementation details 26

Two types of selection operators are used in the aircraft conceptual design process:

Elitist ranking

This method of selection simply returns a subset with cardinality μ of the cumulative candidate offspring by using the ranker function ρ that, as explained before, sorts the design concepts by their fitness (quality).

Remark. This method is used as selection method in one of the four implementations provided in this work (Fig. 4.30).

In that implementation the variation operators are distributed across generations, that is, the cumulative candidate offspring size is kept at minimal, therefore, by using this method the selection pressure may be increased in order to compensate the absence of other variation operators in a given generation.

Stochastic universal sampling (SUS)

This method of selection, as previously described in the theoretical framework chapter tends to select the best design concepts more evenly than the fitness proportional selection method.

This method when applied in the strategy is useful since it may help to maintain diversity within population by passing, from time to time, some “not as good” design concepts to the next generation, reducing the selection pressure.

This method is used as selection method in all but one implementation introduced in this work.

Unlike implementation II, the rest of them do not distribute variation operators within generations (Figs. 4.29, 4.31 and 4.32) resulting in larger cumulative candidate offspring that may provide to this method a broader range of selection.

Additionally, elitism with ranker ρ is used to ensure passing the fittest design con-

cept to the next generation.

Implementation details 26

4.5.4 Simulation-driven design integration

Generation of the domain model is triggered through a listener attached to the evolutionary strategy.

Every time the strategy generates a design concept either by initialization or variation operators, the subscribed listener is notified about the creation action and the domain model is generated from the design variables vector of the newly design concept, following by the parametric simulation model, generated from the domain model (additional models and human readable representation text files are also generated in chain).

There is a similar mechanism for evaluation that notifies the simulation controller whenever an evaluation is required by the strategy.

The simulation controller prepares the simulation tools (e.g., load the simulation model, set simulation conditions, etc), launches the simulation and persists simulation data.

4.5.5 Specific implementations

Four different implementations of the strategy are crafted and tested, and their corresponding results are provided in the next chapter.

Different implementations allow achieving contrasting goals in terms of performance, probability of success, computational costs and research specific goals.

All implementations have in common the variation operator order of application (either distributed or not), that is, knowledge based ones are followed by classical evolutionary ones, specifically: corrective variation operator \rightarrow preconfigured variation operator \rightarrow recombination \rightarrow mutation.

A population injection operator is also applied when required at the very beginning of the variation operators sequence.

Target selection for recombination and mutation are also common to all implementations. Recombination target selection is an elitist selection of the fittest design within the candidate offspring and corresponding r_1 , r_2 and r_3 recombination parents are selected by FPS method (i.e., $r = FPS$).

Target selection for mutation is always the recombinant design resulted from recombination (i.e., last design concept generated in the cumulative candidate offspring).

Another common behavior is having the mutation operator as failover operator for corrective variation operator and recombination.

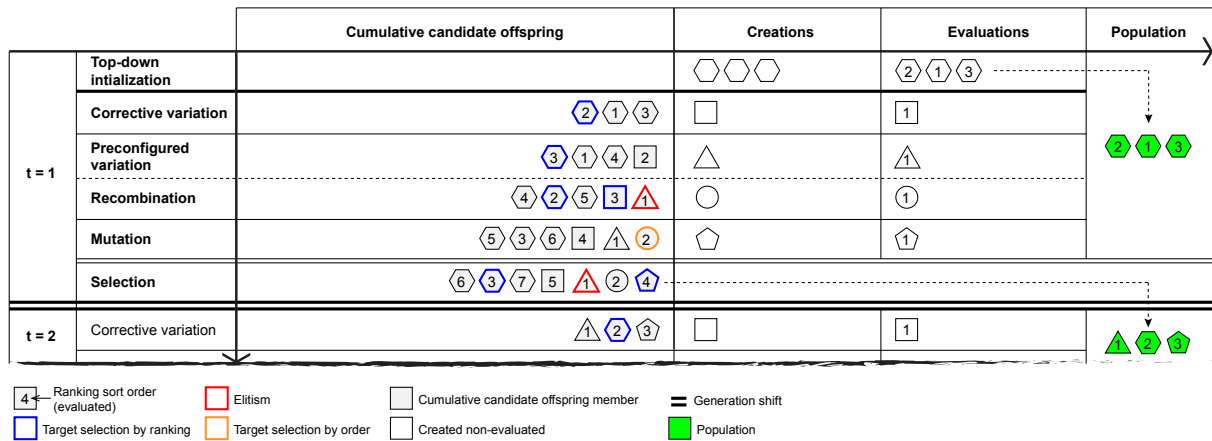


Figure 4.29: Evolutionary strategy implementation I

SUS selection method with elitism for the fittest is used as generation selection operator for all but the II implementation.

4.5.5.1 I: Minimal evaluations per generation

The first implementation of the strategy considers generating and evaluating the minimal amount of design concepts by generation.

Local offspring for every variation operator is composed by only 1 new design.

Corrective variation operator selects the second design concept of the ranked candidate offspring (composed at the time this operator is applied by population of the current generation).

Preconfigured variation operator selects the third design concept of the ranked candidate offspring, that is, the third fittest design of all current population designs and corrective variation operator generated ones combined.

Selecting the fittest for those operators is avoided with the idea of using them as a possible way to improve the next fittest designs since the fittest is selected by recombination anyway.

4.5.5.2 II: Distributed variation operators

The second implementation sets a variation operator distribution across generations.

Corrective variation operator is applied on the whole candidate offspring on the first generation and every 3 generations afterwards.

Preconfigured variation operator is applied on the whole candidate offspring every 3 generations starting at generation 2.

Recombination followed by mutation are both applied every 3 generations after the generation where preconfigured variation is applied.

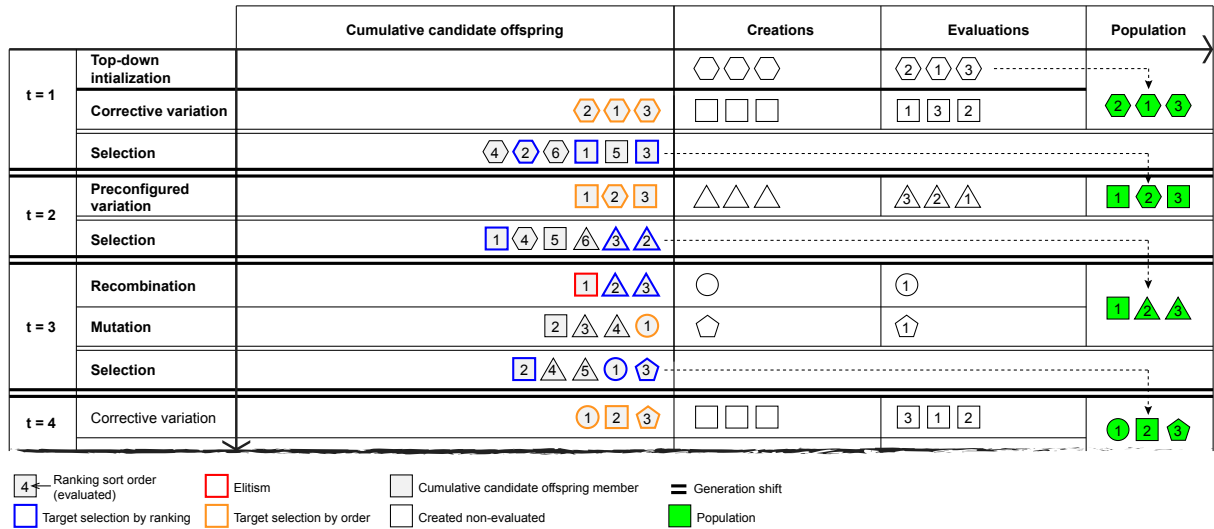


Figure 4.30: Evolutionary strategy implementation II

Population injection operator is configured to be executed if required at the beginning of every generation.

This implementation may help analyze particular behavior and effects induced by specific operators and their components. Selection operator for this particular implementation is changed from SUS to selection of the fittest by ranking to try to increase selection pressure given that the cumulative candidate offspring is kept minimal.

4.5.5.3 III: Maximum probability of success

The third implementation is the one that displays the maximum probability of success by generally achieving the fittest designs with the drawback of being the one with the largest number of evaluations per generation.

It basically applies the knowledge based variation operators on the full cumulative candidate offspring.

4.5.5.4 IV: Medium number of evaluations per generation

The fourth implementation introduces a small variant to the third one that reduce the number of evaluations by bounding the target selection of the preconfigured variation operator (the second operator being applied on the sequence) to the local offspring of the previous operator instead of the full candidate offspring.

4.6 Complementary system representation

Other models can be built from the design variables vector or the domain model of the system.

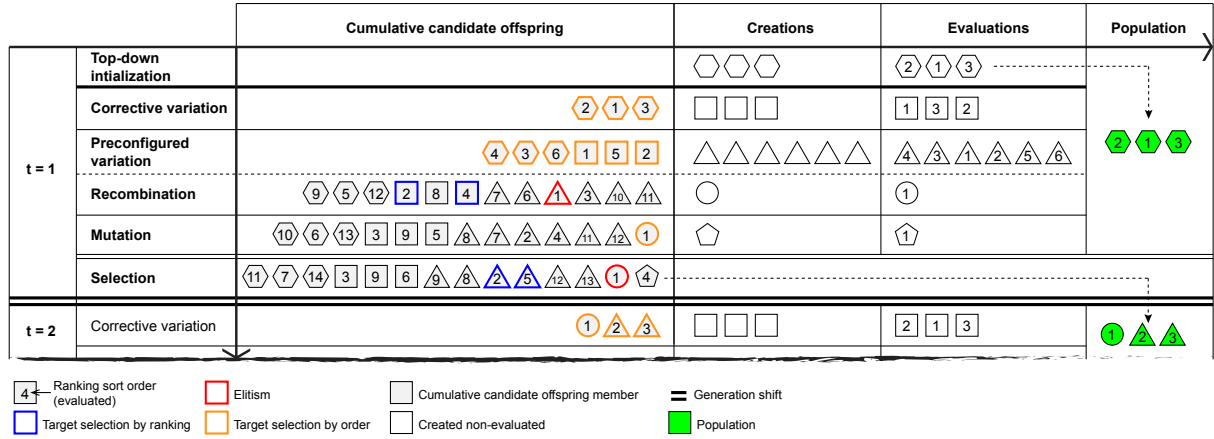


Figure 4.31: Evolutionary strategy implementation III

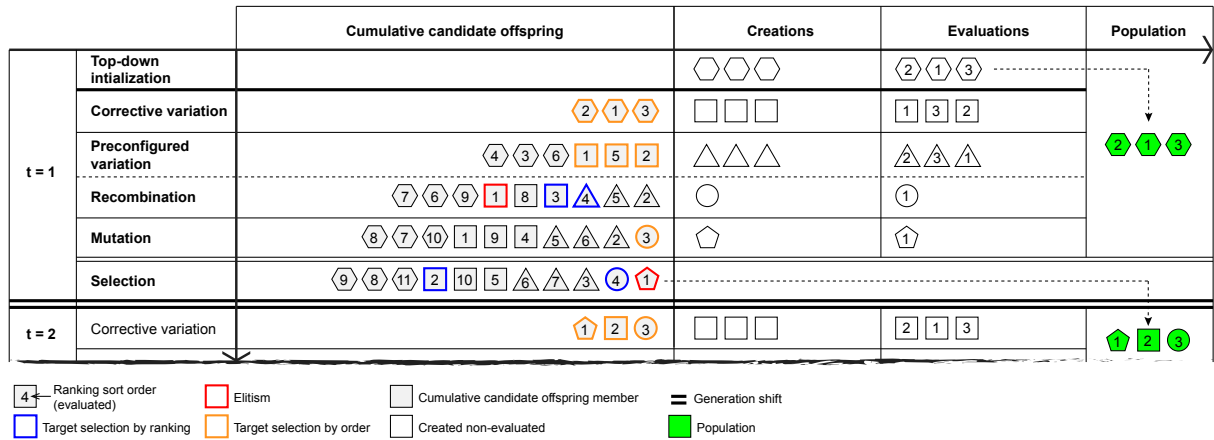


Figure 4.32: Evolutionary strategy implementation IV

These models might be indeed, required by human designers or researches in order to analyze the system from other perspectives.

Visual model

Allows the designers to have a visual perspective of the system.

Visual model

Implementation details 27

For practical purposes, a rendered visual model is not generated from the geometry defined in the domain model, but the simulator companion application for aircraft modeling is used through a custom script to generate visual representations of generated aircrafts in batch.

For each aircraft design concept, 4 views are generated: front, top, side and isometric with 2 versions of each view: wireframe and solid.

The images are post-processed (also in batch) in order to enhance look and contrast.

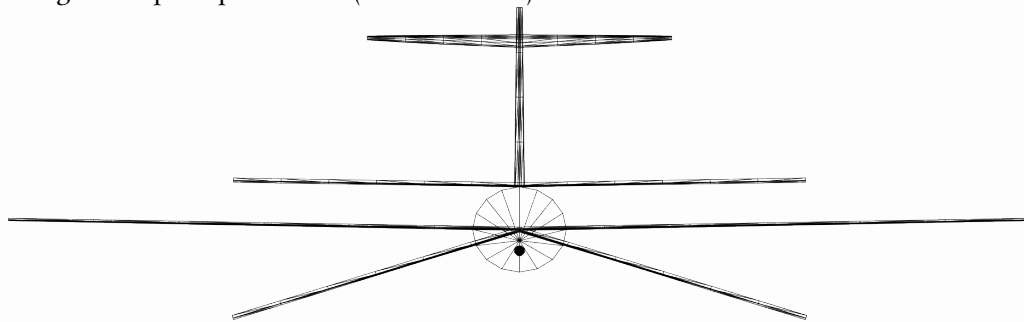


Figure 4.33: Fixed wing aircraft visual model - front wireframe

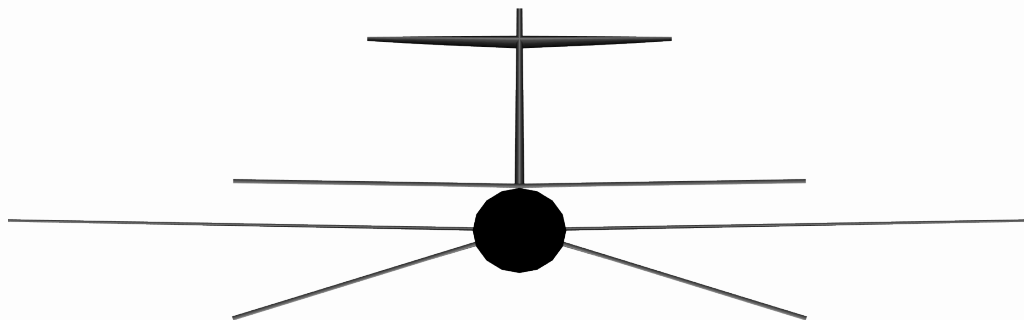


Figure 4.34: Fixed wing aircraft visual model - front solid

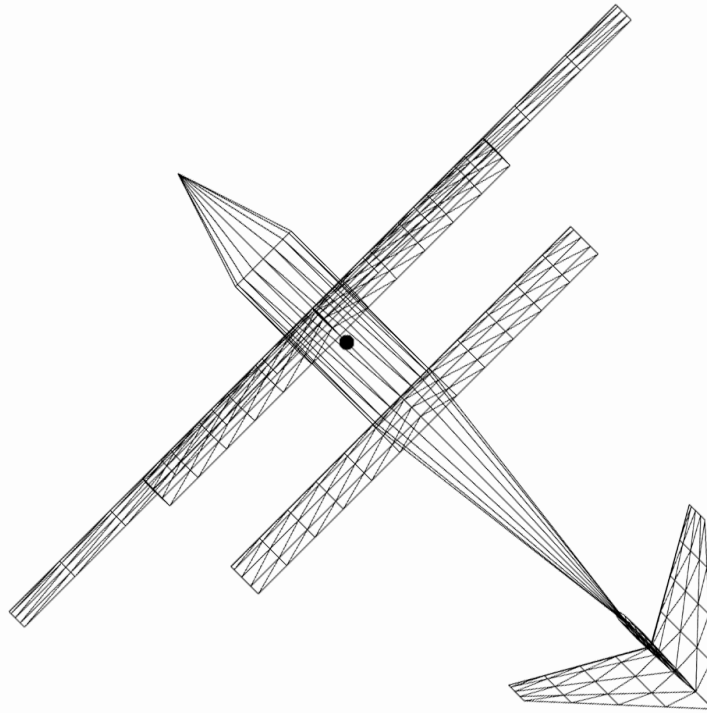


Figure 4.35: Fixed wing aircraft visual model - top wireframe

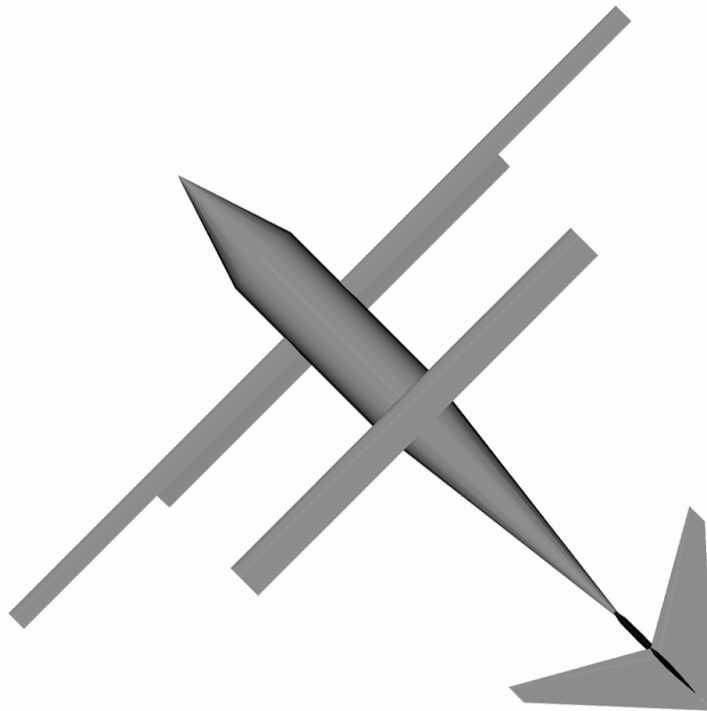


Figure 4.36: Fixed wing aircraft visual model - top solid

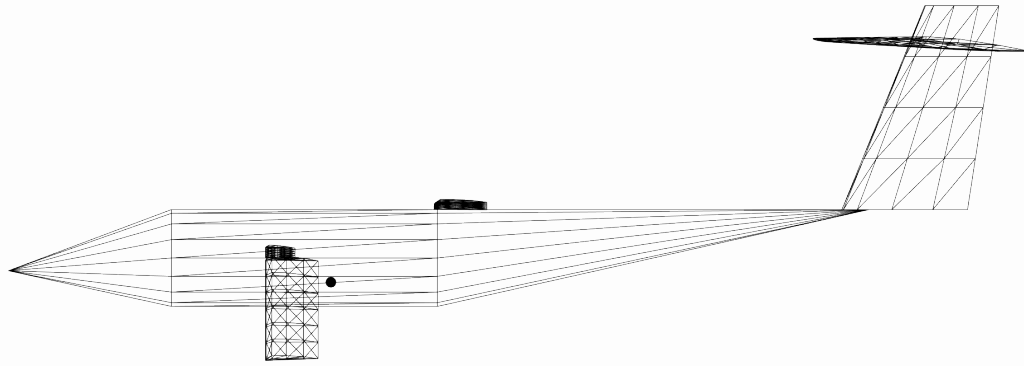


Figure 4.37: Fixed wing aircraft visual model - side wireframe

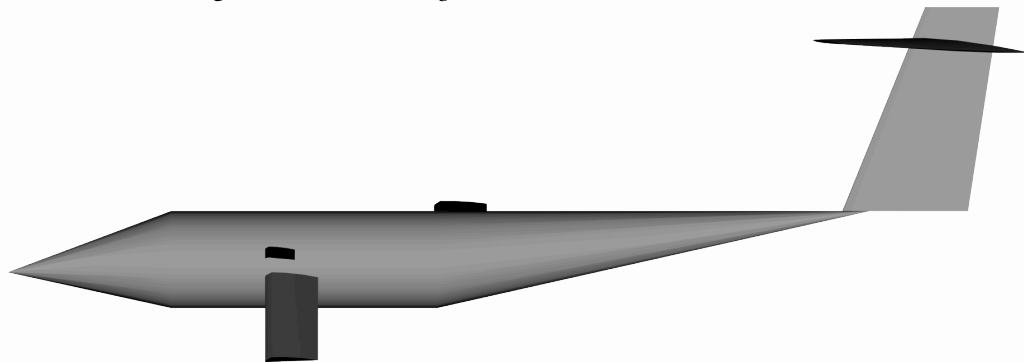


Figure 4.38: Fixed wing aircraft visual model - side solid

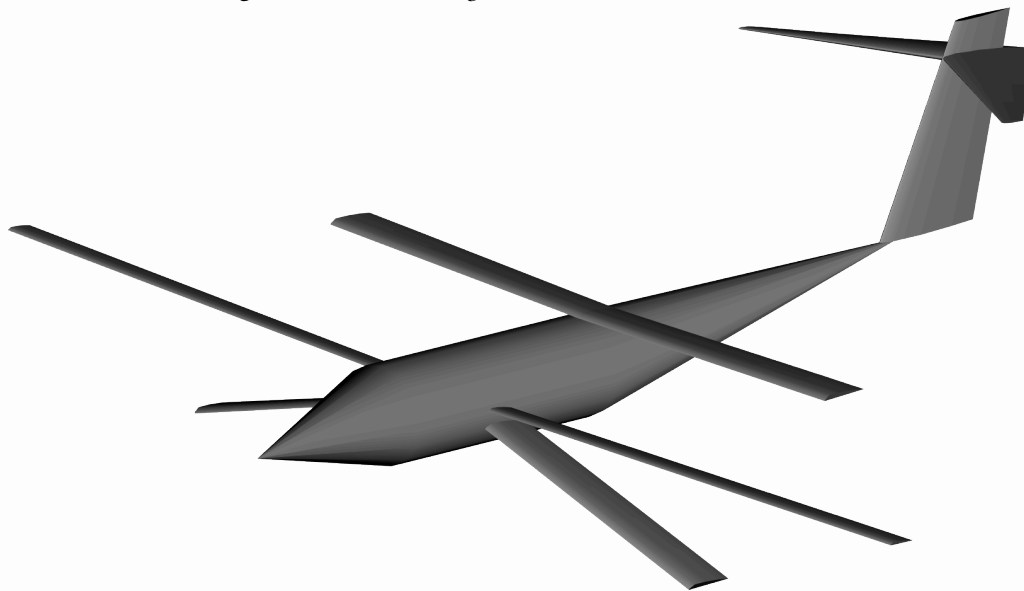


Figure 4.39: Fixed wing aircraft visual model - isometric solid view solid

Implementation details 27

Experiments and results

5.1 Experiment description

The experimental objective of this work is to synthesize an optimal (from the objectives perspective) fixed-wing aircraft design concept flyable inside the simulator according to the simulation script, scalarized fitness function, domain constraints and scenario defined and constructed in the previous chapter through the instrumentation of the strategy also previously developed and implemented as 4 algorithmic variants, namely I, II, III and IV (Figs. 4.29 to 4.32).

In addition, two more variants are derived from base III: classical evolutionary only strategy variant (CEO) and knowledge based only strategy variant (KBO) by removing recombination and mutation from the former and corrective and preconfigured variation operators from the latter.

Injection and consequently injection threshold are also removed from these variants with the purpose of isolating particular full randomness and knowledge based behaviors of the strategy, to provide an intuition of the effectiveness of their combination and the separated capabilities to provide a good solution for a common design problem.

For comparison purposes and in the absence of existing known works using the same approach of proposed strategy applied to aircraft conceptual design, and since the recombination operator used by the strategy is based on the differential recombination/mutation operator, 2 versions of Differential evolution (DE) are implemented.

The first algorithm differential evolution algorithm (adjusted to $\mu = 3$) (DE*) reproduces the classical Differential evolution (DE) except for allowing a population size $\mu = 3$ instead of the minimal recommended 4, that is, the differential recombination operator is adjusted to enable degenerated combinations of target and parent design concepts, very similar to the custom operator introduced in the strategy.

The second variant of the differential evolution algorithm custom differential evolution algorithm (adjusted to $\mu = 3$ with differential arithmetic recombination) (CDE*), includes the recombination operator designed for the strategy as is, that is, it enables differential mutation,

differential arithmetical recombination and uniform recombination from original design and differential mutant. This variant is implemented to comparatively test the behavior of classical differential operator and the custom operator of the strategy on the same problem.

The experiment consists in executing a battery of 30 design automation processes per algorithm, sharing a common fixed initial population of 3 randomly generated aircraft design concepts (3 wing shape family representatives) by using the top-down method introduced in this strategy and simulation settings and conditions described in previous chapter (Tables 4.7 and 4.8).

Each algorithm is configured according to (Tables 5.1 to 5.4)¹ including parameters $\{\tau^S, \vec{f}_{MIN}, g_{MAX}\} \in \Lambda^C$ that enable stopping an algorithm execution when no further improvement is detected after τ^S generations, when fitness evaluation goes less than or equal to \vec{f}_{MIN} or when number of generations exceed g_{MAX} [40].

Algorithm	μ	g_{MAX}	Executions	Population injection operator I			τ^S	\vec{f}_{MIN}	Next generation selection ς	
				g_{0I}	g_{eI}	τ^I			Method	Elitism
I		50								
III		20							SUS	$\varsigma(\rho(X_g^{IE}), \{1..1\})$
IV		30		1		5	$\tau^I + 2$			
II	3	90	30					0.0		$\varsigma(\rho(X_g^{IE}), \{1..\mu\})$
KBO		20							SUS	$\varsigma(\rho(X_g^{IE}), \{1..1\})$
CEO						N/A	5			
DE*		90								$\varsigma(\rho(X_g^{IE}), \{1..\mu\})$
CDE*										

Table 5.1: Experiment general parameters and settings

Algorithm	Distribution		Target selection σ	Failover η
	g_0	g_e		
I			$\sigma(\rho(X_g^{IE}), \{2..2\})$	
IV		1	$\sigma(X_g^{IE}, \{ X_g^{IE} - 2.. X_g^{IE} \})$	Mutation
III	1			
II		3	$\sigma(X_g^{IE}, \{1.. X_g^{IE} \})$	
KBO		1		Preconfigured
DE*				
CDE*			N/A	
CEO				

(a) Corrective variation

Algorithm	Distribution		Target selection	Failover
	g_0	g_e		
II	2	3		
III			$\sigma(X_g^{IE}, \{1.. X_g^{IE} \})$	
KBO		1		None
I			$\sigma(\rho(X_g^{IE}), \{3..3\})$	
IV			$\sigma(X_g^{IE}, \{ X_g^{IE} - 2.. X_g^{IE} \})$	
DE*				
CDE*			N/A	
CEO				

(b) Preconfigured variation

Table 5.2: Knowledge based variation operators parameters and settings

¹Maximum number of generations allowed per algorithm is set in relation with the expected number of evaluations in a particular algorithm

Algorithm	Distribution		Target selection σ	Parent selection i	Failover η	F				Cr			Pf	
	g_0	g_e				α_F	β_F	m_F	d_X	α_{Cr}	α_{Cr}	m_{Cr}	c_{Pf}	κ
I														
III	1				Preconfigured									
IV			$\sigma\left(\varrho\left(X_g^{tE}\right),\{1..1\}\right)$			0.4	2.0	jitter	full	0.2	0.5	ditter	0.45	$\frac{F+1}{2}$
II	3			FPS										
CEO					None									
CDE*	1				N/A									
DE*			$\sigma\left(X_g^{tE},\left\{1.. X_g^{tE} \right\}\right)$											N/A
KBO					N/A									

Table 5.3: Recombination parameters and settings

During evolution, variation operators execution order is consistently the same in all algorithms (except on those where some operators are removed or are not applicable):

1. Knowledge based variation operator
 - (a) Corrective variation
 - (b) Preconfigured variation
2. Classical evolutionary operators
 - (a) Recombination
 - (b) Mutation

Algorithm	Distribution		Target selection σ	Failover η	Mp			
	g_0	g_e			α_{Mp}	β_{Mp}	m_{Mp}	d_M
I								
III	1							
IV			$\sigma\left(X_g^{tE},\left\{ X_g^{tE} \dots X_g^{tE} \right\}\right)$	None	0.1	0.45	dither	full
II	3							
CEO	1							
CDE*								
DE*				N/A				
KBO								

Table 5.4: Mutation parameters and settings

5.2 Results

5.2.1 Results analysis

The experiment battery accounts for a total of 240 aircraft conceptual design process executions, half of them belonging to the proposed strategy, 60 for isolated variation operator types analysis and 60 for comparison with differential evolution purposes.

After the execution of the experiment, 33421 design concepts \vec{x} were generated and evaluated (i.e., 33421 simulations performed) accumulating slightly more than 300 simulation hours.

Initial population X_1 shared by all algorithms executions achieved fitness evaluations $\vec{f}(\vec{x})$ of 5960.17, 5860.75 and 4953.98 (Figs. 5.6 to 5.8), that is, considerably non-optimal designs to fulfill the design objectives.

The strategy reached a global optimal fitness (limited to experiment maximum number of generations and other termination criteria) of 18.74, a global optimal mean fitness of 74.41 and a global optimal median fitness of 70.24 for a maximum relative efficacy of 99.66%.

CEO, CDE* and DE* algorithms fall in stagnation before reaching 30 iterations (Fig. 5.1), that is, they all were unable to further induce improvements in their populations within the stagnation threshold.

The other algorithms completed in overall their configured maximum generations displaying continuous improvement until stopping demonstrating the effectiveness of the strategy proposed key aspects.

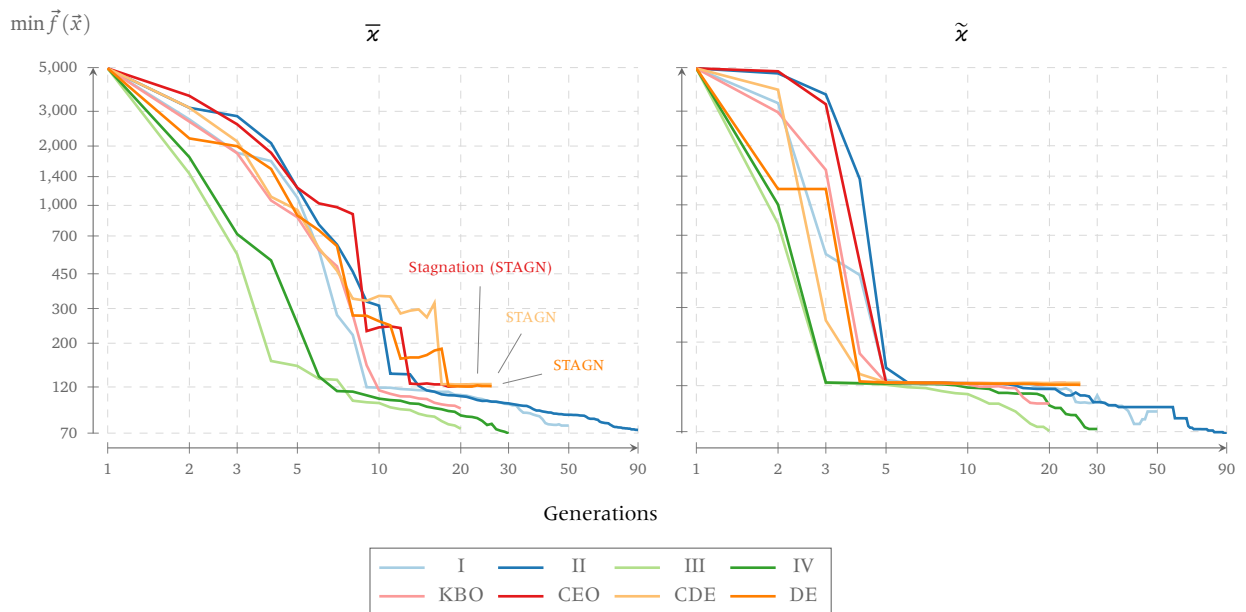


Figure 5.1: Global convergence curve

In general, all strategy variants achieve similar optimal fitness evaluations (Table 5.5), differences become evident when analyzing their average and maximum relative efficacy (im-

provement achieved percentage), relative efficiency (evaluations : efficacy) and relative quality (global optimal fitness : algorithm optimal fitness) metrics.

Algorithm	$\min \vec{f}(\vec{x})$			$ X_g^{E} $				Initial fitness	Improvement	Relative efficacy		Relative efficiency	Relative quality
	\bar{x}	\tilde{x}	min	\bar{x}	\tilde{x}	min	max			\bar{x}	max		
I	95.04	97.34	23.37	148	140	66	218		5496.59	98.30%	99.58%	0.66	78.29%
II	76.43	70.24	23.04	267	286	73	316		5515.20	98.63%	99.59%	0.37	97.36%
III	74.41	71.35	28.59	222	223	180	247		5517.22	98.67%	99.49%	0.44	100.00%
IV	80.08	78.28	18.78	217	238	118	244		5511.55	98.57%	99.66%	0.45	92.92%
KBO	98.60	99.17	46.08	156	183	66	183	5591.63	5493.03	98.24%	99.18%	0.63	75.47%
CEO	525.82	123.05	71.56	28	27	15	47		5065.81	90.60%	98.72%	3.24	14.15%
DE*	208.11	122.90	112.39	53	54	24	81		5383.52	96.28%	97.99%	1.82	35.76%
CDE*	307.59	123.54	119.48	47	45	27	81		5333.78	94.50%	97.86%	2.05	24.19%

Table 5.5: Global results

I (①efficiency, ④average efficacy, ③max efficacy, ④quality)

Variant I is the most efficient variant but has the lowest efficacy and quality in average.

II (④efficiency, ②average efficacy, ②max efficacy, ②quality)

Variant II is the least efficient variant but has the second best efficacy and quality.

III (③efficiency, ①average efficacy, ④max efficacy, ①quality)

Variant III is the second least efficient variant but has the highest average efficacy and quality, although achieve the least maximum relative efficacy.

IV (②efficiency, ③average efficacy, ①max efficacy, ①quality)

Variant IV is the second most efficient variant but has the second lowest average efficacy and quality. It has also the highest maximum relative efficacy.

Relative efficiency of DE*, CDE* and CEO are considerably better than the knowledge based counterparts including KBO, however, as previously mentioned, they fall in stagnation and have the poorest relative quality metrics (bellow 40%). Even the best fitness evaluation of them is $2 + \frac{1}{2}$ times worst the less optimal evaluation of the proposed strategy based algorithms and, although they are not equipped with the injection operator, the differences do not seem to be strongly related to that situation given that KBO did not fall into stagnation even when it does not have injection operator as well.

When comparing DE* and CDE* it is shown that classical differential evolution method for arithmetical recombination slightly outperforms (in this particular case) the custom operator that combines randomly two different methods of arithmetical recombination. This might be intuitively explained because the classical operator is designed to behave like a recombination/-mutation operator and, when crafting the custom operator based on the differential one the desired behavior was more related to recombination effect than mutation effect.

5.2.1.1 Individual objectives

KBO algorithm (designed to analyze the isolated knowledge based operators) results help to demonstrate the effectiveness of the knowledge based variation operators in this implementation case, and specially the preconfigured variation operator (due to the small corrective rules set implemented in this work).

During final generations of its executions the algorithm keeps continuously improving, outperformed by III variant and slightly outperformed by the IV one (Fig. 5.1) but, on the other hand, outperforming variants I and II at least until stopping (explained by the slow convergence speed of I and II variants).

This behavior suggests that, as planned, the domain knowledge and the high level building blocks provided by the preconfigured variation operator heavily accounts for probability of success, but also shows that both, knowledge based and classical evolutionary operators complement each other to achieve the best fitness evaluations (neither CEO nor KBO were able to reach most optimal evaluations although knowledge based by its own reached better fitness evaluations).

The real differences arise when analyzing results by individual objective (Figs. 5.2 and 5.3 and Table 5.6), where clearly the maximization of glide distance objective followed by minimization of the maximum absolute roll are the objectives that make evident the contributions of the application of domain knowledge in the design evolutionary process.

In general all algorithms including differential evolution heavily improves the other objectives but the strategy based algorithms (even the KBO by its own) allow constructing building blocks containing design concepts that fulfill the whole set of objectives.

Algorithm	$\max \check{G}(\vec{x})$			$\min \sigma^2_{\varphi(\vec{x},t)}_{1 \leq t \leq T}$			$\min \max_{1 \leq t \leq T} \varphi(\vec{x},t) $			$\min \sigma^2_{\psi(\vec{x},t)}_{1 \leq t \leq T}$		
	\bar{x}	\tilde{x}	max	\bar{x}	\tilde{x}	min	\bar{x}	\tilde{x}	min	\bar{x}	\tilde{x}	min
I	39.90	36.11	130.25	0.16	0.013	0.0011	1.04	0.73	0.29	0.27	0.036	0.0016
II	63.24	72.09	130.68	0.50	0.014	0.0007	1.70	0.95	0.26	0.13	0.029	0.0030
III	65.62	69.19	123.60	0.14	0.009	0.0003	1.04	0.70	0.22	0.07	0.037	0.0005
IV	58.47	60.56	136.09	0.10	0.005	0.0002	1.04	0.79	0.22	0.12	0.027	0.0005
KBO	35.12	33.77	101.35	2.94	0.045	0.0010	3.05	1.63	0.17	2.78	0.041	0.0012
CEO	11.60	7.38	71.31	52.65	2.756	0.0055	15.07	5.18	0.56	658.24	0.788	0.0533
DE*	8.61	6.07	27.96	21.13	6.810	0.0176	10.85	7.64	0.32	144.93	0.946	0.0240
CDE*	8.26	6.55	20.01	61.33	5.082	0.0160	19.59	7.44	0.79	335.91	0.878	0.0306

Table 5.6: Global results by objective

5.2.1.2 Multivariate analysis

Despite the fact that this strategy and the solution proposed was not designed as a “fully” multi-objective optimization solution (a scalarization technique was used to transform the multi-objective nature into a mono-objective nature), it is interesting to analyze the experiment results by individual objectives.

This analysis actually helped to distinguish the real differences in terms of fitness evaluation achievement between the strategy based algorithms and the ones not supported by domain knowledge.

Part of this analysis consisted in preparing parallels coordinates plots for each algorithm (Figs. 5.4 and 5.5).

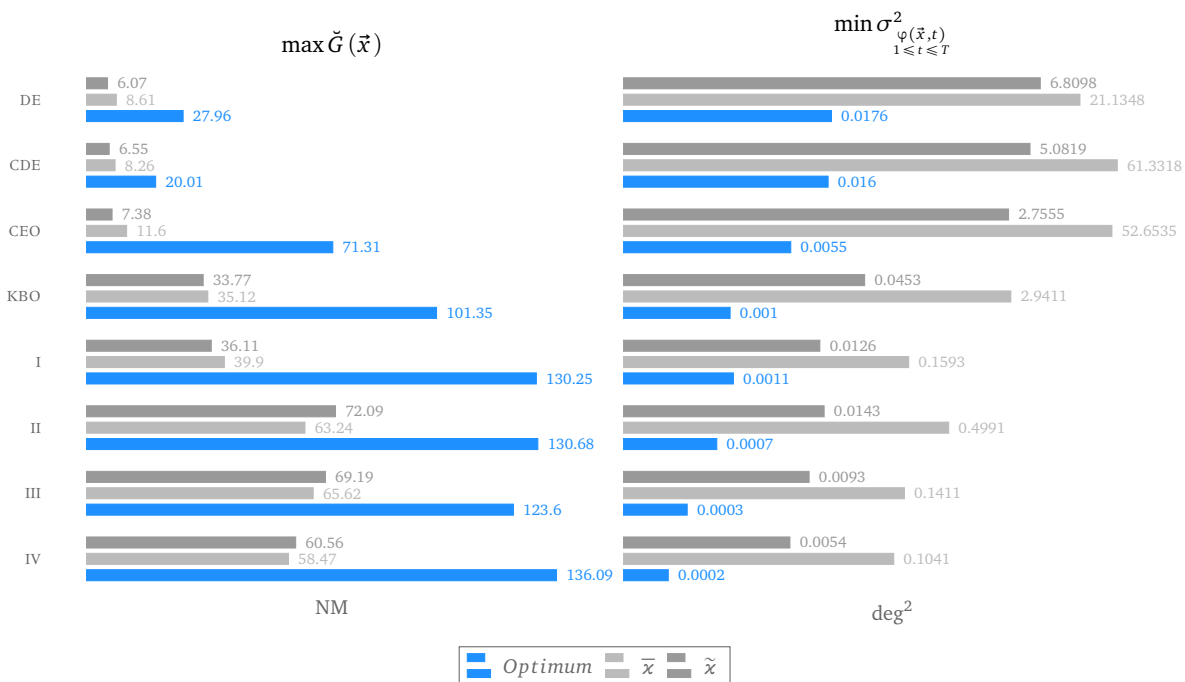


Figure 5.2: Global results by objective I

This kind of plot allows displaying multivariate data in 2 dimensions. The data being plotted are the actual objective functions values for each objective and for each design concept evaluated during all executions of each algorithm.

Each objective is represented with a vertical axis line, a design concept is a straight line plotted from a point in the first objective axis line to a point in the second objective axis line to the third and to the fourth objective.

The intersection point in each objective axis corresponds with the actual objective function value for that objective normalized from the interval $[A, B]$ to the interval $[0, 1]$ where A is the global best (the best of all executions of all algorithms) objective function value and B is the global worst objective function value.

As a result, the plot allows identifying (by objective) better solutions at the bottom and worse solutions at the top, as well as the coverage of the solutions, that is whether a solution is a good or bad solution for 1, 2, 3 or 4 objectives and in what precise amount.

This last visualization advantage was enhanced by grouping the data into 4 clusters through a k -means clustering algorithm, passing the 4 objective functions values as features to build the clusters.

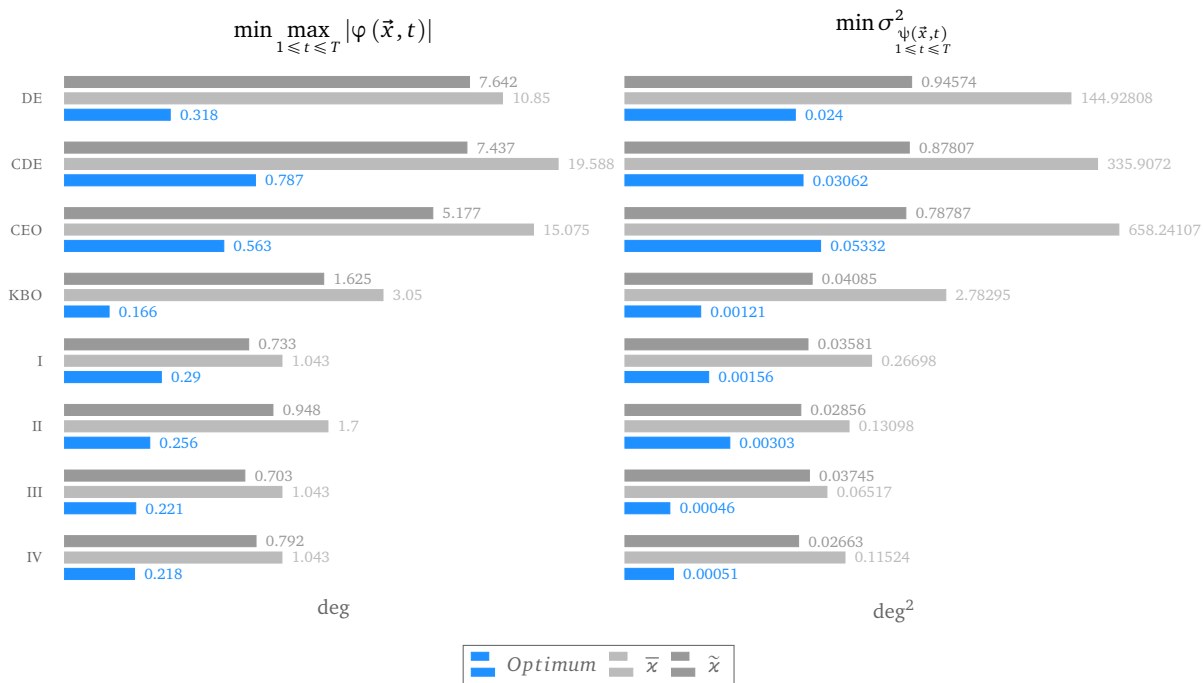


Figure 5.3: Global results by objective II

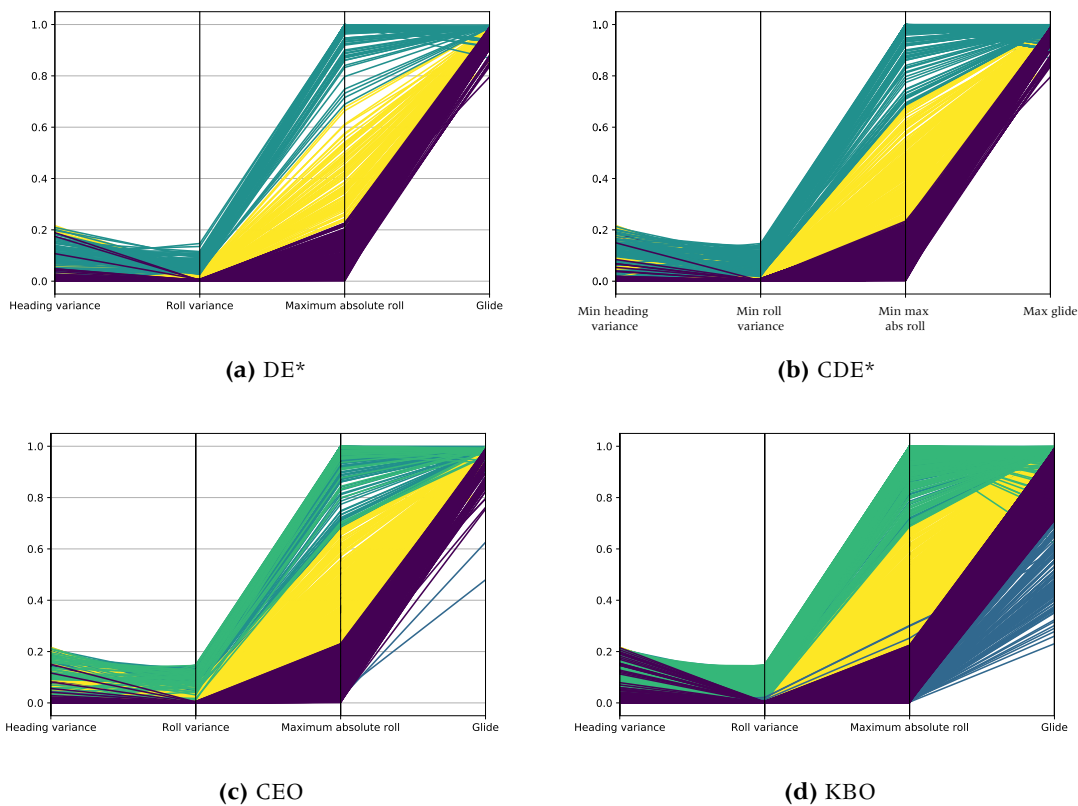


Figure 5.4: Multivariate parallel coordinates plot I

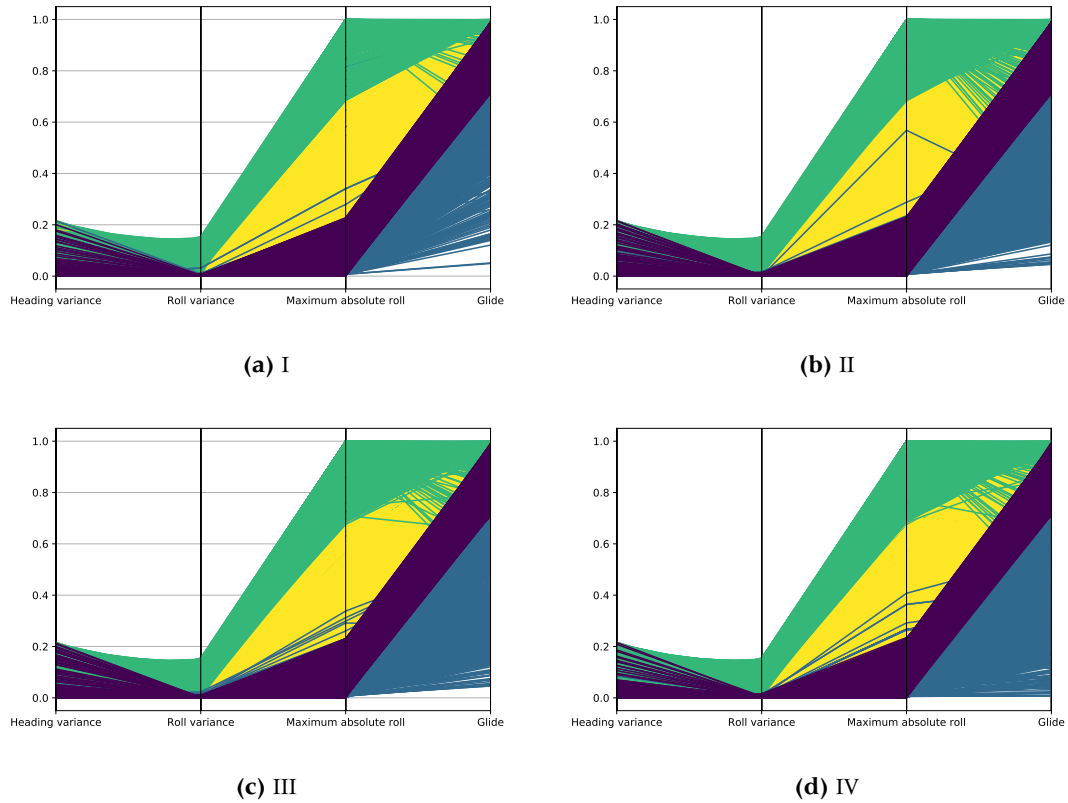


Figure 5.5: Multivariate parallel coordinates plot II

5.2.2 Initial shared population

5.2.2.1 Design variables

Table 5.7: Initial population design concepts design variables

Design variable	Pointed	Constant chord	Trapezoidal
Fixed values			
MLW		42000 lb	
S		860 ft ²	
S_h		217 ft ²	
S_v		170 ft ²	
t_y		4.41 ft	
n_y		-1.16 ft	
$f_{\tilde{n}}$		1.98	
$f_{\tilde{m}}$		1.98	
fw		9.74 ft	
fh		8.83 ft	

Continue on next page

Table 5.7 – continued from previous page

Design variable	Pointed	Constant chord	Trapezoidal
lm		24.33 ft	
lt/d		4.26	
ln/d		1.61	
$\frac{t_{max}}{c} \chi$		30%	
$\frac{th_{max}}{c} \chi$		30%	
$\frac{\delta h_{max}}{c} \chi$		0%	
$\frac{\delta h_{max}}{c}$		0%	
$\frac{\delta v_{max}}{c} \chi$		0%	
$\frac{\delta v_{max}}{c}$		0%	
$\frac{tv_{max}}{c} \chi$		30%	
ϵ_v		0°	
L_v		0°	
Γ_v		90°	
$\frac{\delta vh_{max}}{c} \chi$		0%	
$\frac{\delta vh_{max}}{c}$		0%	
$\frac{tvh_{max}}{c} \chi$		30%	
Non-fixed values			
$\frac{th_{max}}{c} [1]$	7%	8%	6%
$\epsilon_h [1]$	3.83°	0.65°	0.99°
$l_h [1]$	4.87°	3.19°	3.78°
$\Gamma_h [1]$	-3°	20°	12°
$\Lambda l_h [1]$	1.00	0.25	0.00
$\Lambda_h [1]$	29°	0°	1°
$\lambda_h [1]$	0.00	1.00	0.00
$\mathcal{R}_h [1]$	3.22	5.18	1.63
$z_h [1]$	0.71	0.39	0.71
$y_h [1]$	0.40	0.03	0.08
$hpos_h [1]$	2	0	0
Nh	1	1	1
Nvh	0	0	0
Nv	1	1	1
$\frac{x\delta_{max}}{c} \chi [1]$	0%	40%	0%
$\frac{\delta_{max}}{c} [1]$	0%	8%	0%
$\frac{t_{max}}{c} [1]$	7%	9%	13%
$\epsilon [1]$	-1.13°	-2.79°	-3.22°
$\iota [1]$	0.00°	5.63°	0.89°
$\Gamma [1]$	0°	6°	-4°
$\Lambda l [1]$	0.5	0	0.5
$\Lambda [1]$	-19°	0°	-58°
$\lambda [1]$	0.00	1.00	0.23
$\mathcal{R} [1]$	10.49	0.33	15.00
$z_w [1]$	0.49	0.35	0.34
Continue on next page			

Table 5.7 – continued from previous page

Design variable	Pointed	Constant chord	Trapezoidal
y_w [1]	0.82	0.88	0.18
$hpos_w$ [1]	0	1	2
Nw	1	1	1
$\frac{t_{vmax}}{c}$ [1]	7%	8%	10%
Λ_l [1]	1.00	0.25	0.25
Λ_v [1]	40°	11°	24°
λ_v [1]	0.00	1.00	0.06
\mathcal{R}_v [1]	1.61	2.03	2.25
z_v [1]	0.53	0.88	0.51

5.2.2.2 Visual models

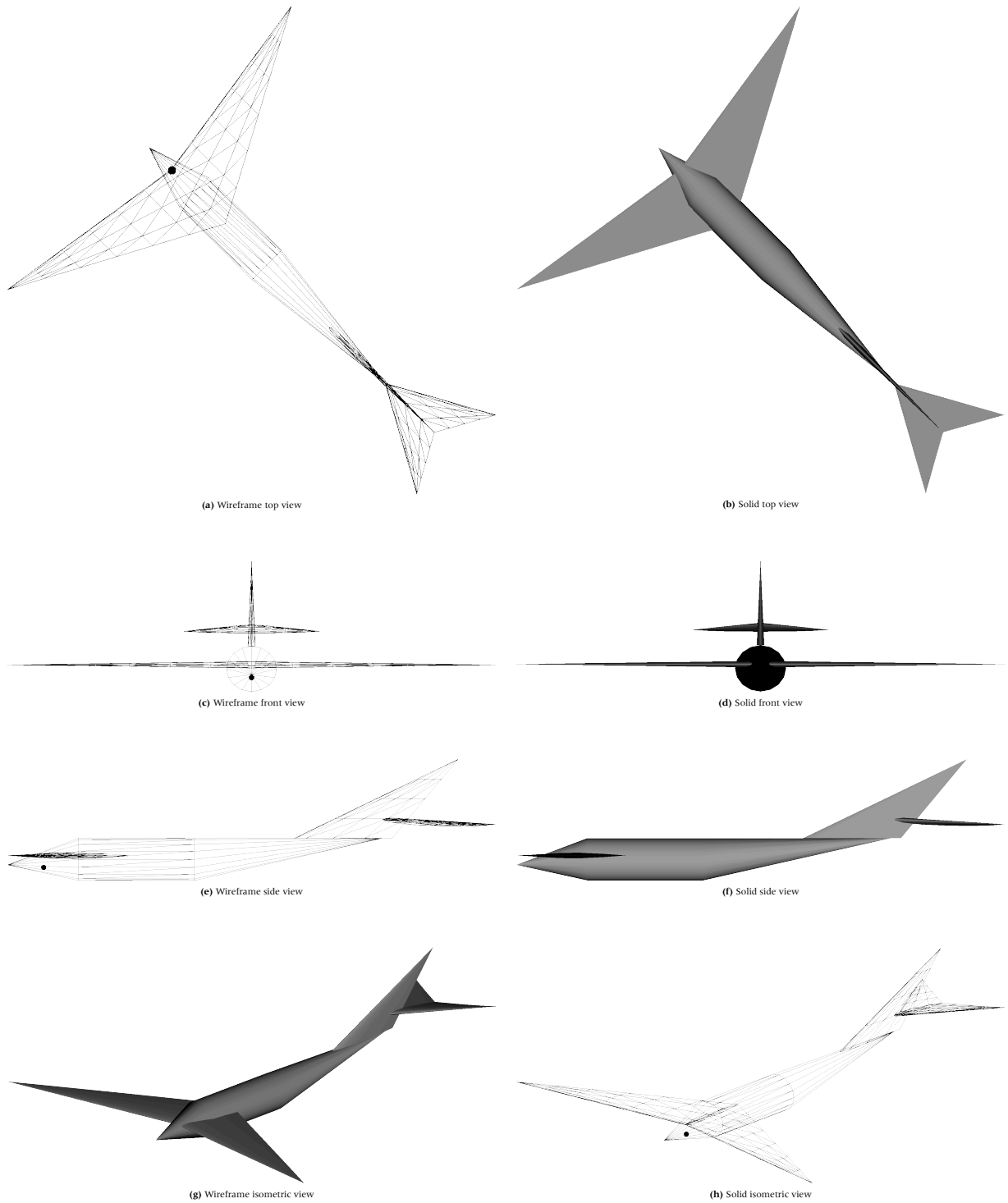


Figure 5.6: Initial shared population aircraft design concept - pointed wing shape family representative

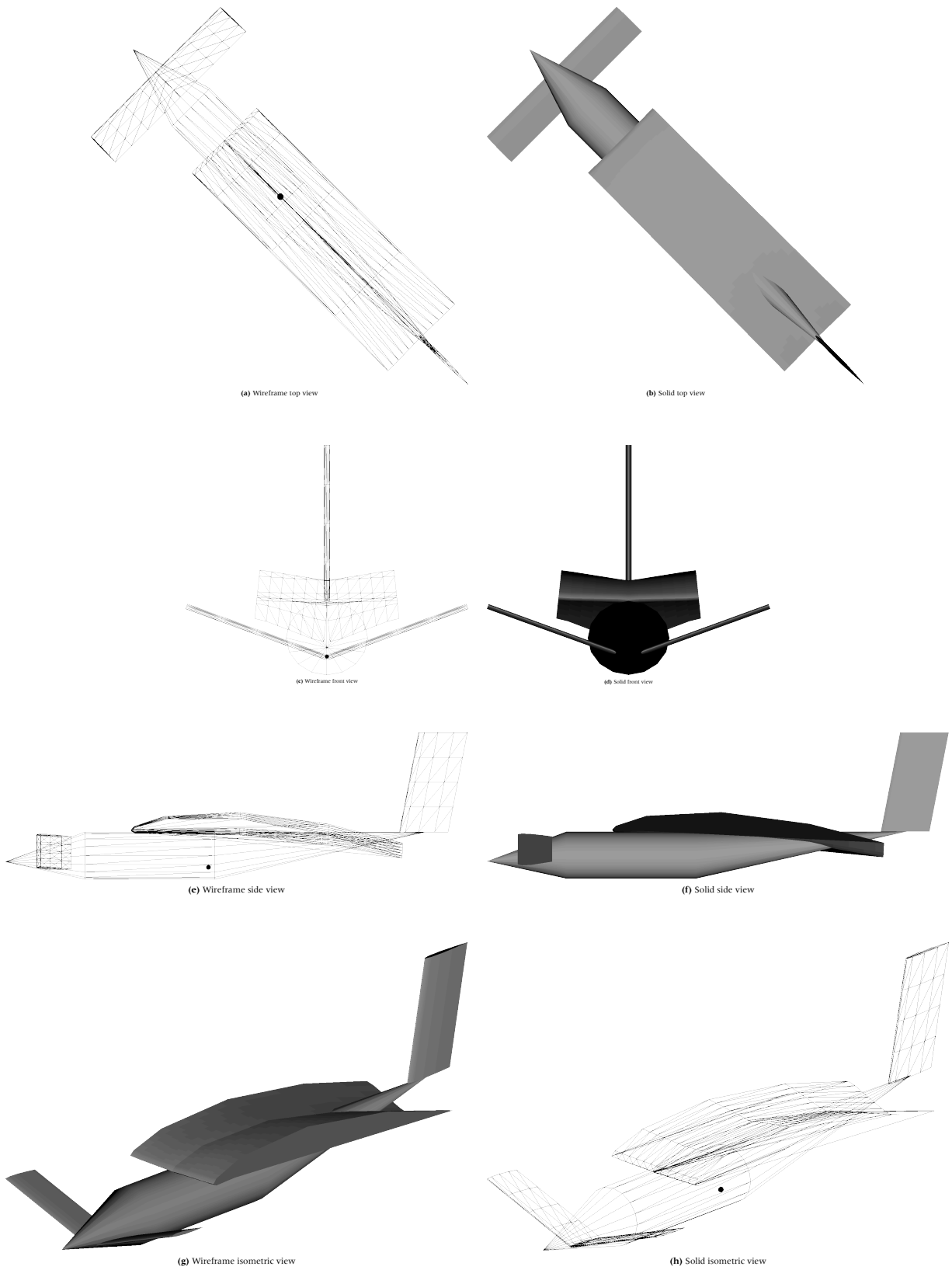


Figure 5.7: Initial shared population aircraft design concept - constant chord wing shape family representative

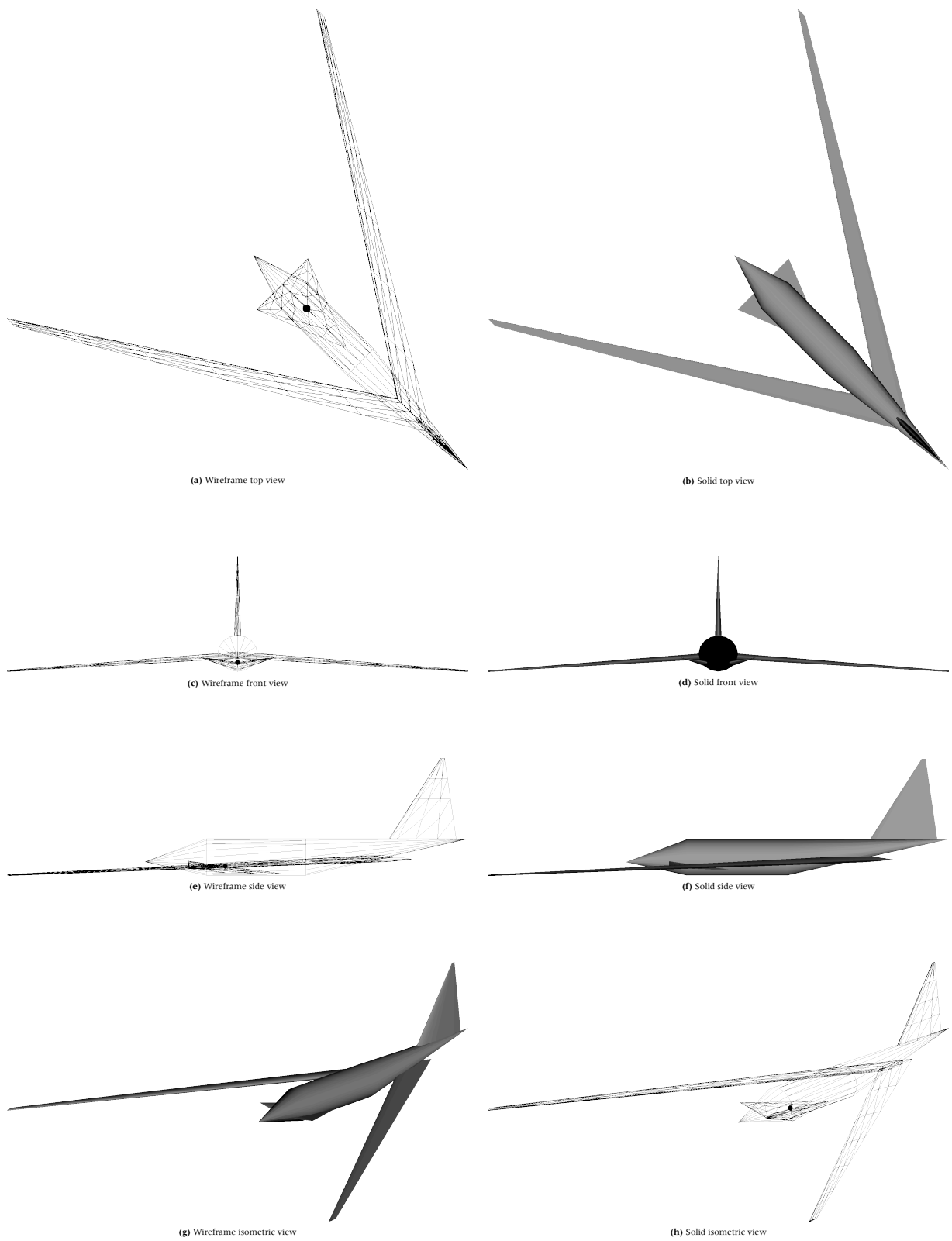


Figure 5.8: Initial shared population aircraft design concept - trapezoidal wing shape family representative

5.2.3 Fittest design concepts

5.2.3.1 Design variables

Table 5.8: Global fittest design concepts design variables

Design variable	I	II	III	IV	KBO	CEO	DE	CDE
Fixed values								
MLW				42000 lb				
S				860 ft ²				
S_h				217 ft ²				
S_v				170 ft ²				
t_y				4.41 ft				
n_y				-1.16 ft				
$f_{\tilde{h}}$				1.98				
$f_{\tilde{m}}$				1.98				
fw				9.74 ft				
fh				8.83 ft				
lm				24.33 ft				
lt/d				4.26				
ln/d				1.61				
$\frac{t_{max}}{c} \chi$				30%				
$\frac{th_{max}}{c} \chi$				30%				
$\frac{\delta h_{max}}{c} \chi$				0%				
$\frac{\delta h_{max}}{c}$				0%				
$\frac{\delta v_{max}}{c} \chi$				0%				
$\frac{\delta v_{max}}{c}$				0%				
$\frac{tv_{max}}{c} \chi$				30%				
ϵ_v				0°				
l_v				0°				
Γ_v				90°				
$\frac{\delta vh_{max}}{c} \chi$				0%				
$\frac{\delta vh_{max}}{c}$				0%				
$\frac{tvh_{max}}{c} \chi$				30%				
Non-fixed values								
$\frac{th_{max}}{c} [1]$	7%	7%	9%	6%	8%	7%	6%	6%
$\epsilon_h [1]$	0.15°	-2.55°	-0.56°	-1.92°	0.65°	0.99°	3.30°	2.01°
$l_h [1]$	3.03°	0.49°	3.57°	2.36°	3.19°	4.54°	3.78°	3.78°
$\Gamma_h [1]$	-20°	-10°	-13°	-10°	20°	11°	15°	2°
$\Lambda l_h [1]$	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
$\Lambda_h [1]$	52°	24°	0°	48°	0°	1°	-55°	6°
$\lambda_h [1]$	0.54	0.87	0.27	1.00	1.00	0.00	0.00	0.00
$\mathcal{R}_h [1]$	2.22	6.14	3.59	3.08	5.18	6.75	3.22	2.17
$z_h [1]$	0.65	0.79	0.93	0.39	0.95	0.48	0.72	0.93

Continue on next page

Table 5.8 – continued from previous page

Design variable	I	II	III	IV	KBO	CEO	DE	CDE
y_h [1]	0.40	0.93	0.05	0.44	0.98	0.08	0.70	0.35
$hpos_h$ [1]	2	2	2	2	2	2	2	2
Nh	1	1	1	1	1	1	1	1
Nvh	0	0	0	0	0	0	0	0
Nv	0	0	0	0	0	0	1	1
$\frac{x\delta_{max}}{c} x$ [1]	10%	10%	10%	10%	10%	0%	10%	10%
$\frac{\delta_{max}}{c}$ [1]	5%	8%	9%	3%	9%	0%	7%	4%
$\frac{t_{max}}{c}$ [1]	7%	9%	10%	9%	9%	16%	13%	13%
ϵ [1]	-3.10°	-0.79°	3.38°	-2.79°	3.41°	-3.22°	-2.18°	3.76°
i [1]	5.88°	1.87°	2.71°	5.63°	3.69°	4.02°	0.89°	0.97°
Γ [1]	7°	17°	14°	4°	16°	6°	-4°	0°
Λl [1]	0.25	0.25	0.25	0	0	0.25	0.25	0.25
Λ [1]	19°	-5°	0°	26°	0°	11°	30°	17°
λ [1]	0.45	0.31	0.43	0.83	0.46	0.33	0.65	0.89
\mathcal{R} [1]	31.99	30.80	39.09	39.90	25.07	15.00	25.76	15.00
z_w [1]	0.53	0.08	0.15	0.41	0.09	0.31	0.49	0.83
y_w [1]	0.93	1.00	0.04	0.93	0.49	0.18	0.16	0.19
$hpos_w$ [1]	1	1	0	0	1	1	1	0
Nw	1	1	1	1	1	1	1	1
$\frac{t_{vmax}}{c}$ [1]	---	---	---	---	---	---	7%	10%
Λl_v [1]	---	---	---	---	---	---	0.25	0.25
Λ_v [1]	---	---	---	---	---	---	14°	24°
λ_v [1]	---	---	---	---	---	---	0.06	0.83
\mathcal{R}_v [1]	---	---	---	---	---	---	2.96	2.25
z_v [1]	---	---	---	---	---	---	0.84	0.51

5.2.3.2 Visual models

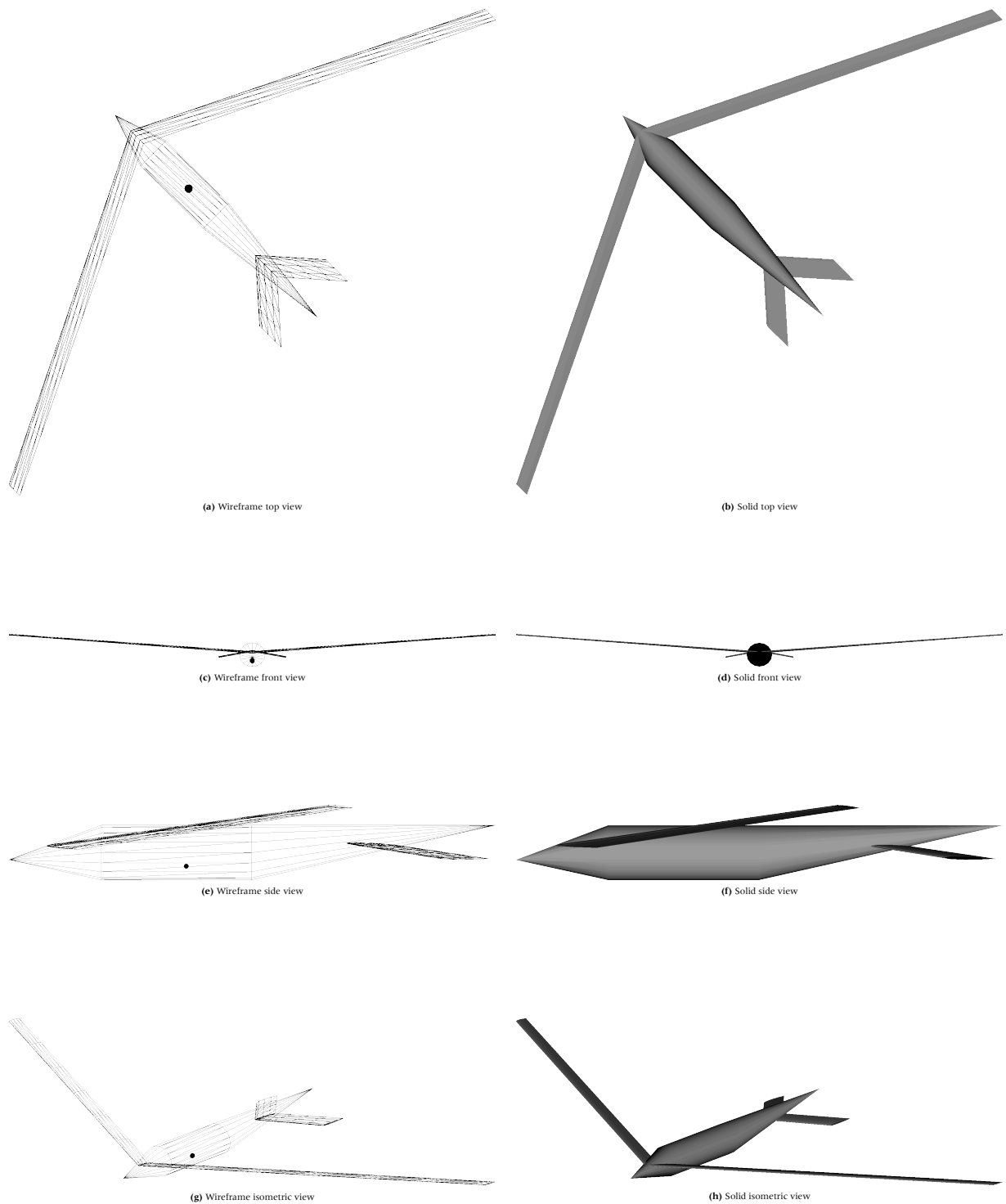


Figure 5.9: Global fittest aircraft design concept generated by algorithm IV

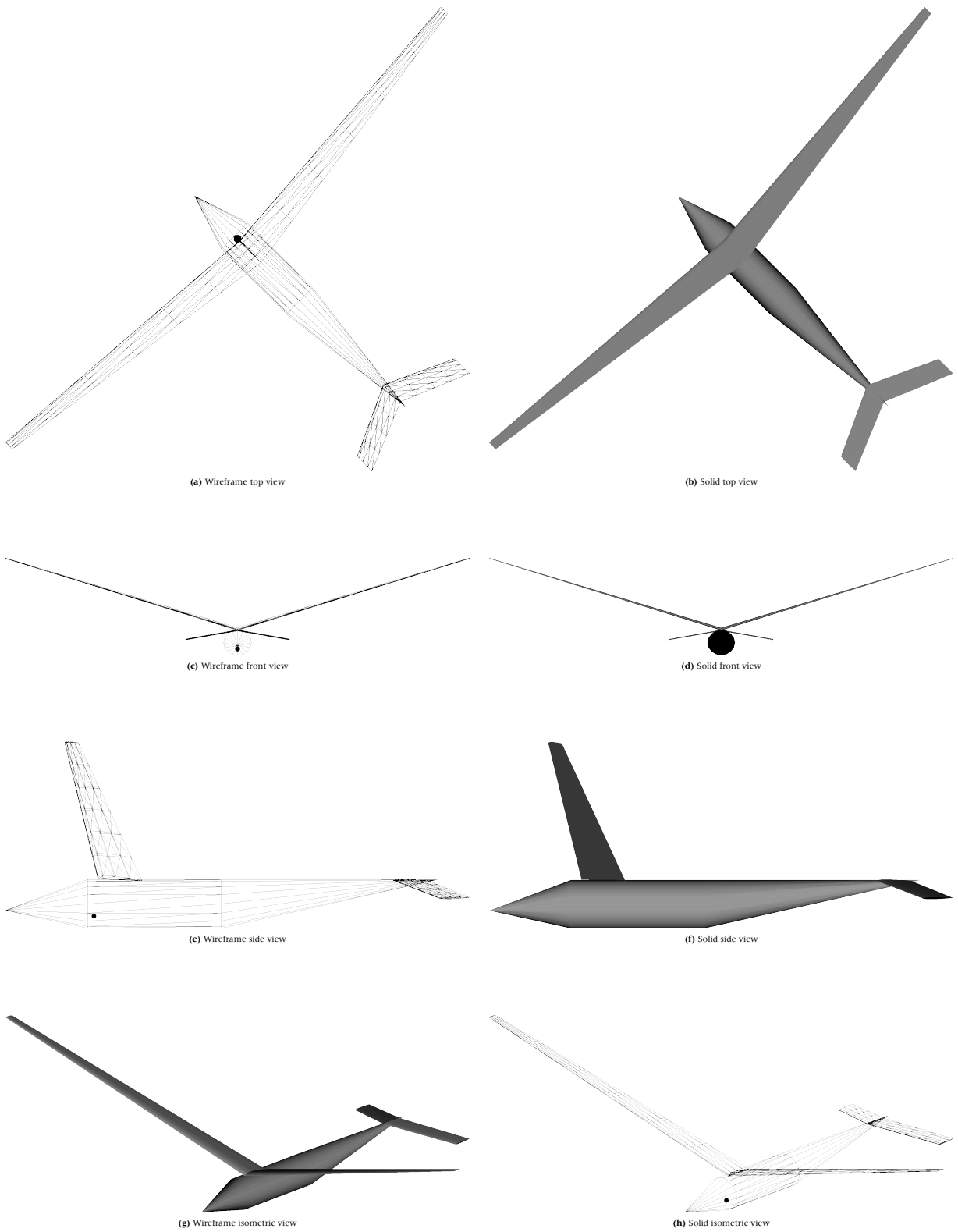


Figure 5.10: Fittest aircraft design concept generated by algorithm II

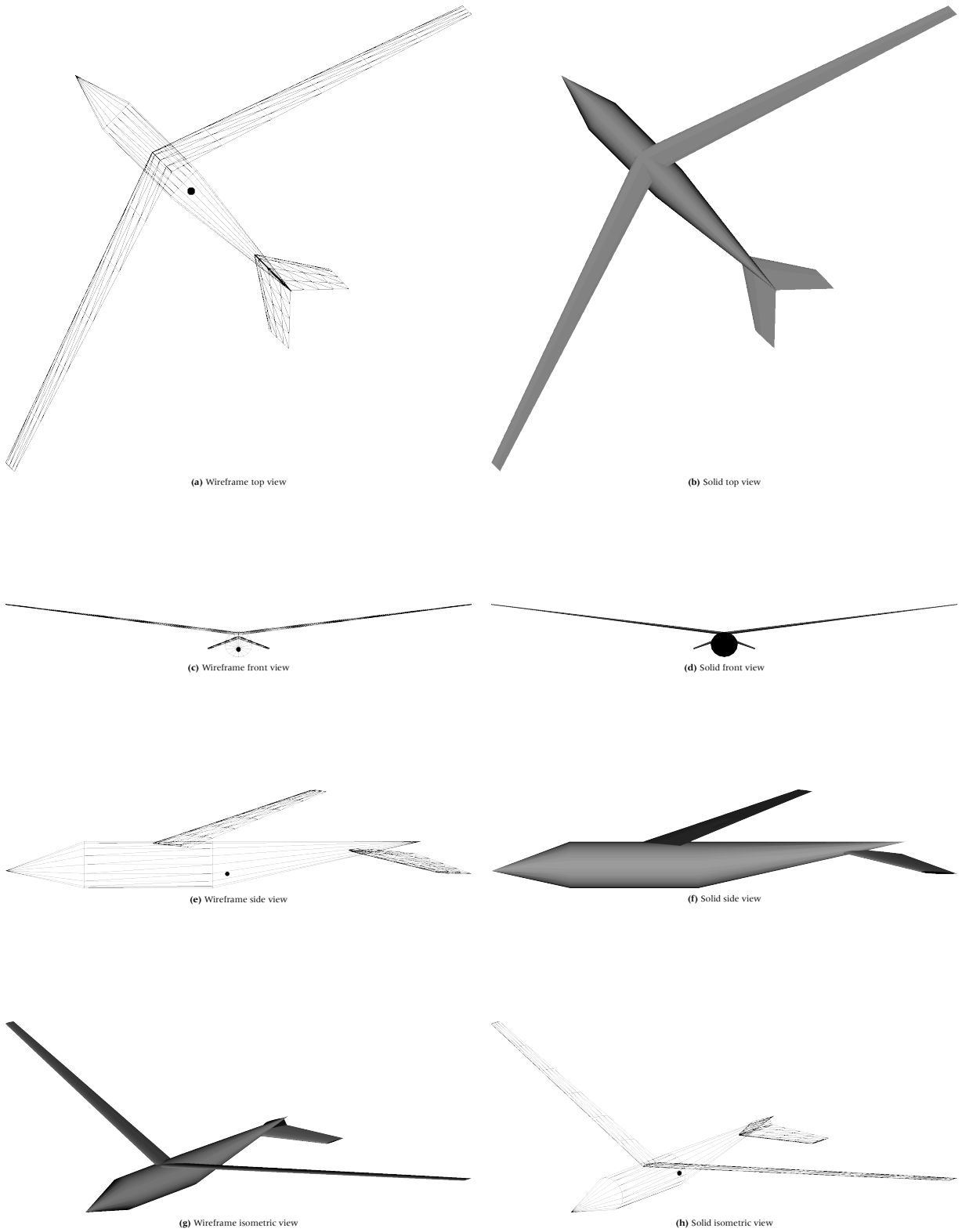


Figure 5.11: Fittest aircraft design concept generated by algorithm I

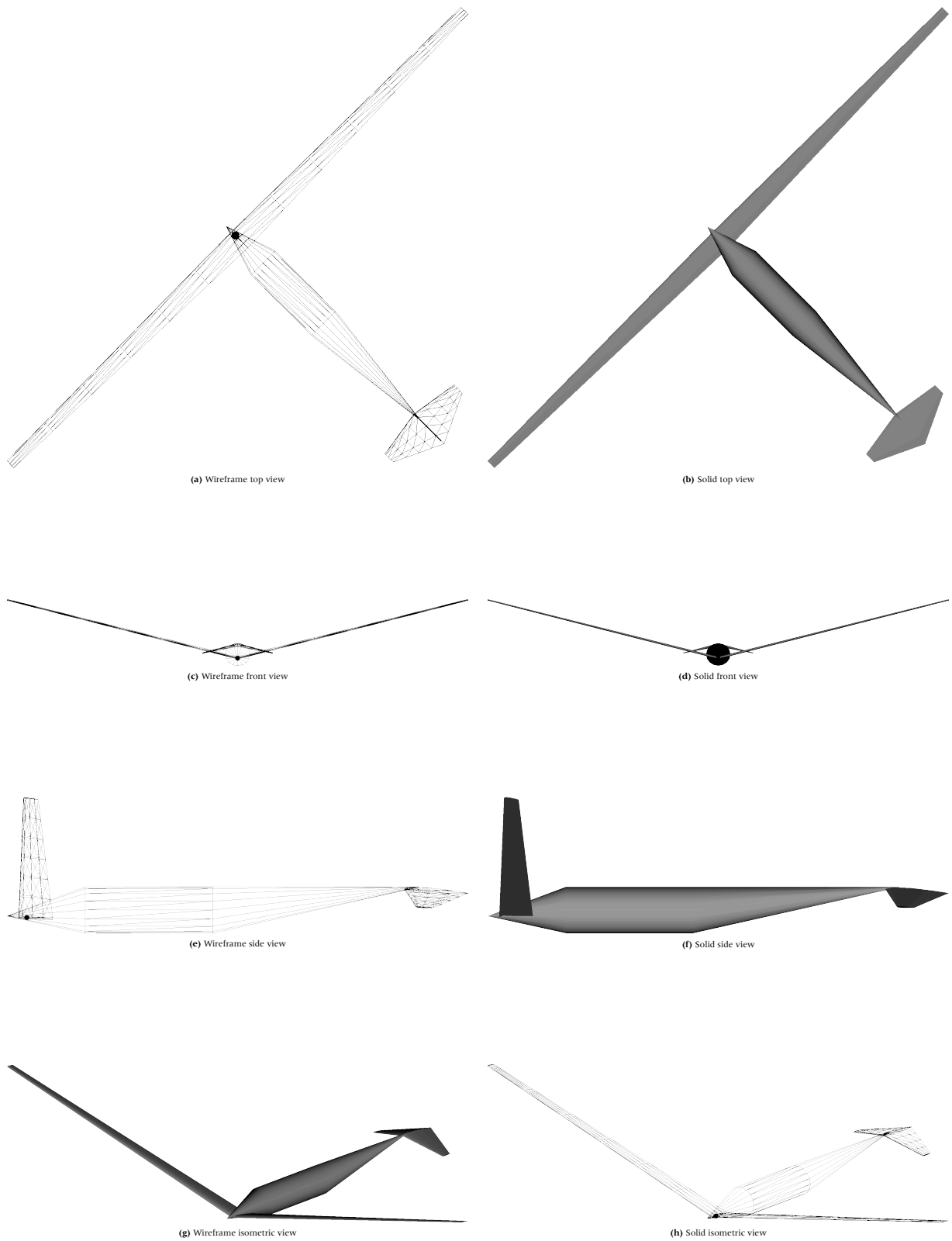


Figure 5.12: Fittest aircraft design concept generated by algorithm III

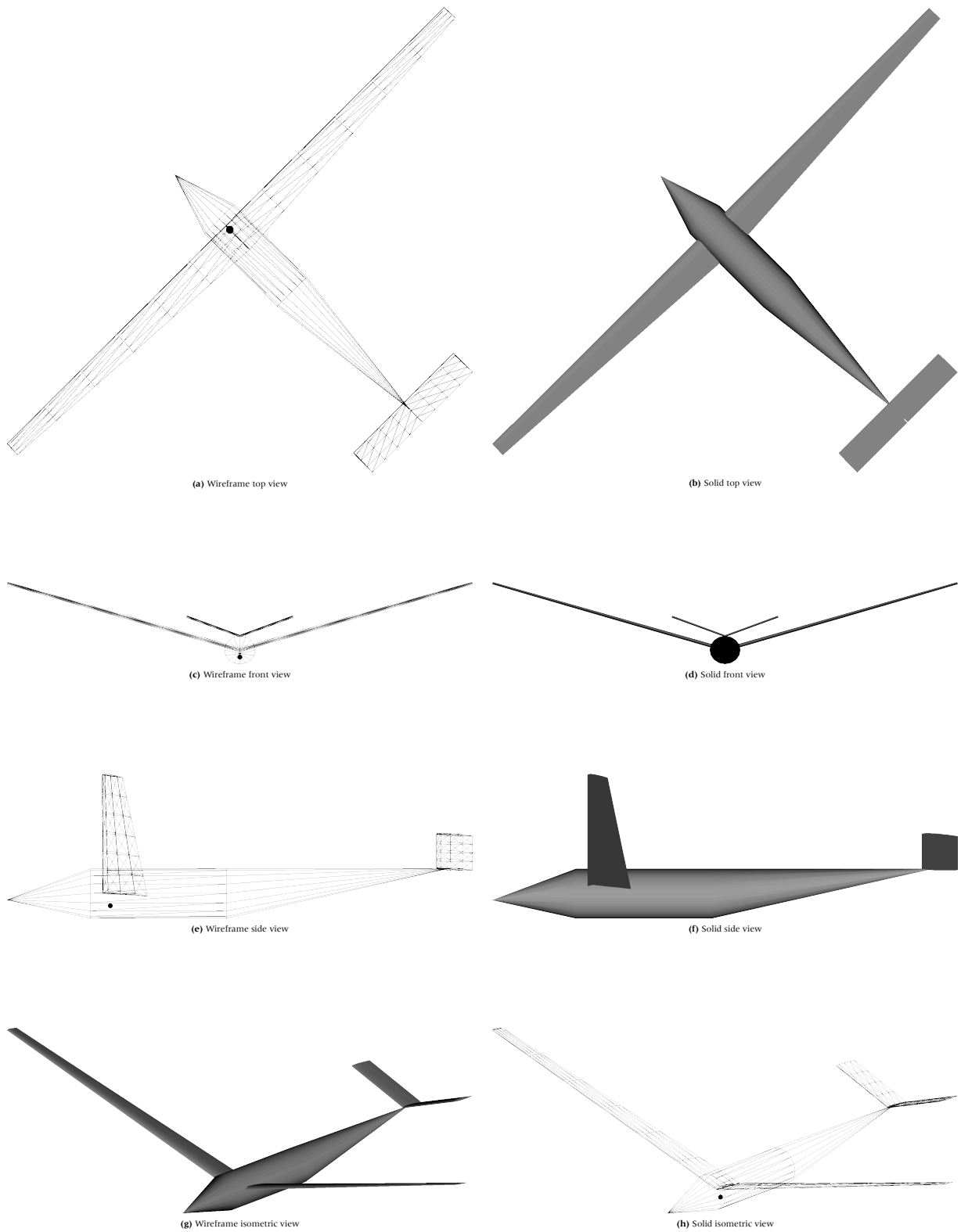


Figure 5.13: Fittest aircraft design concept generated by algorithmic variant KBO

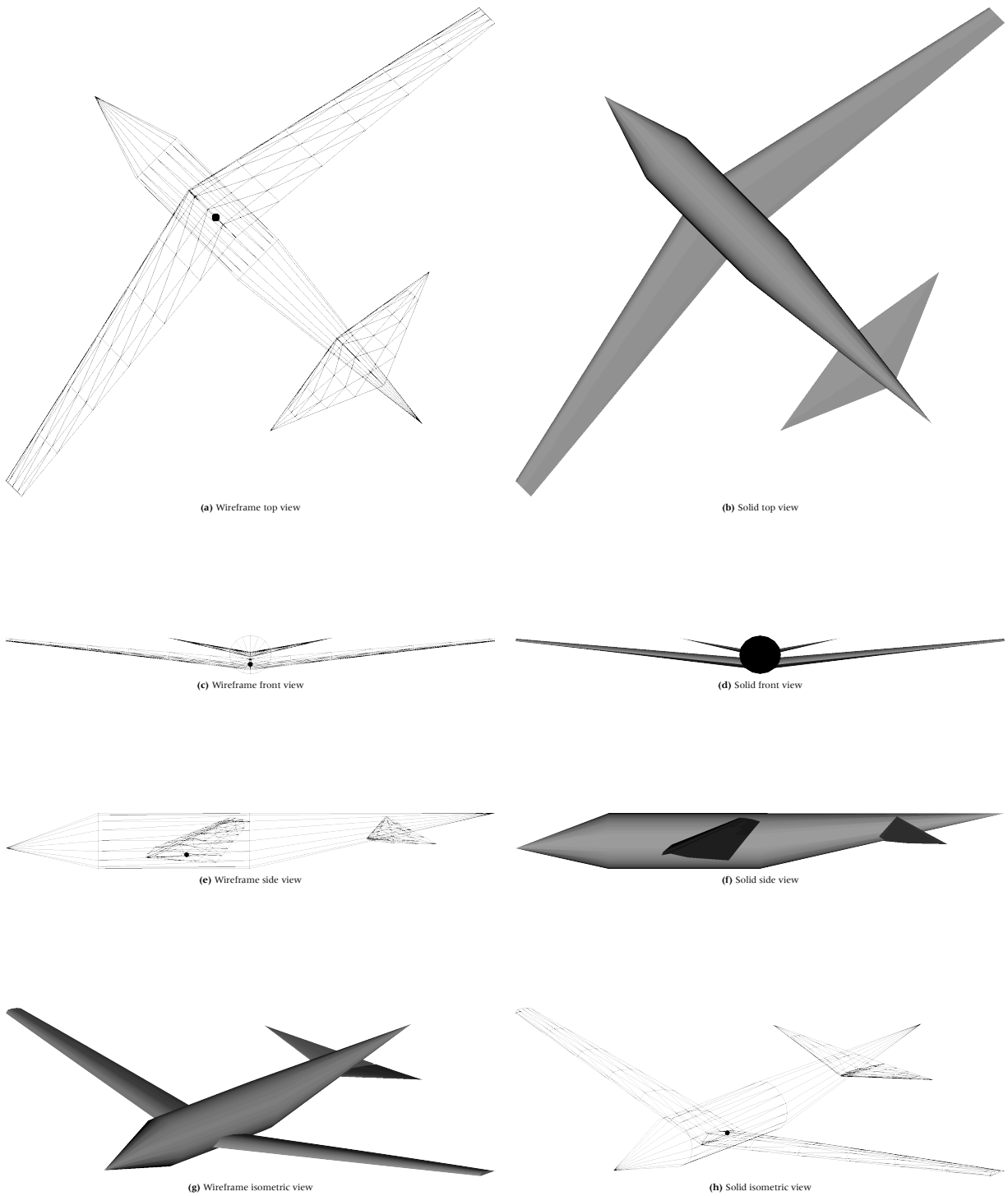


Figure 5.14: Fittest aircraft design concept generated by algorithmic variant CEO

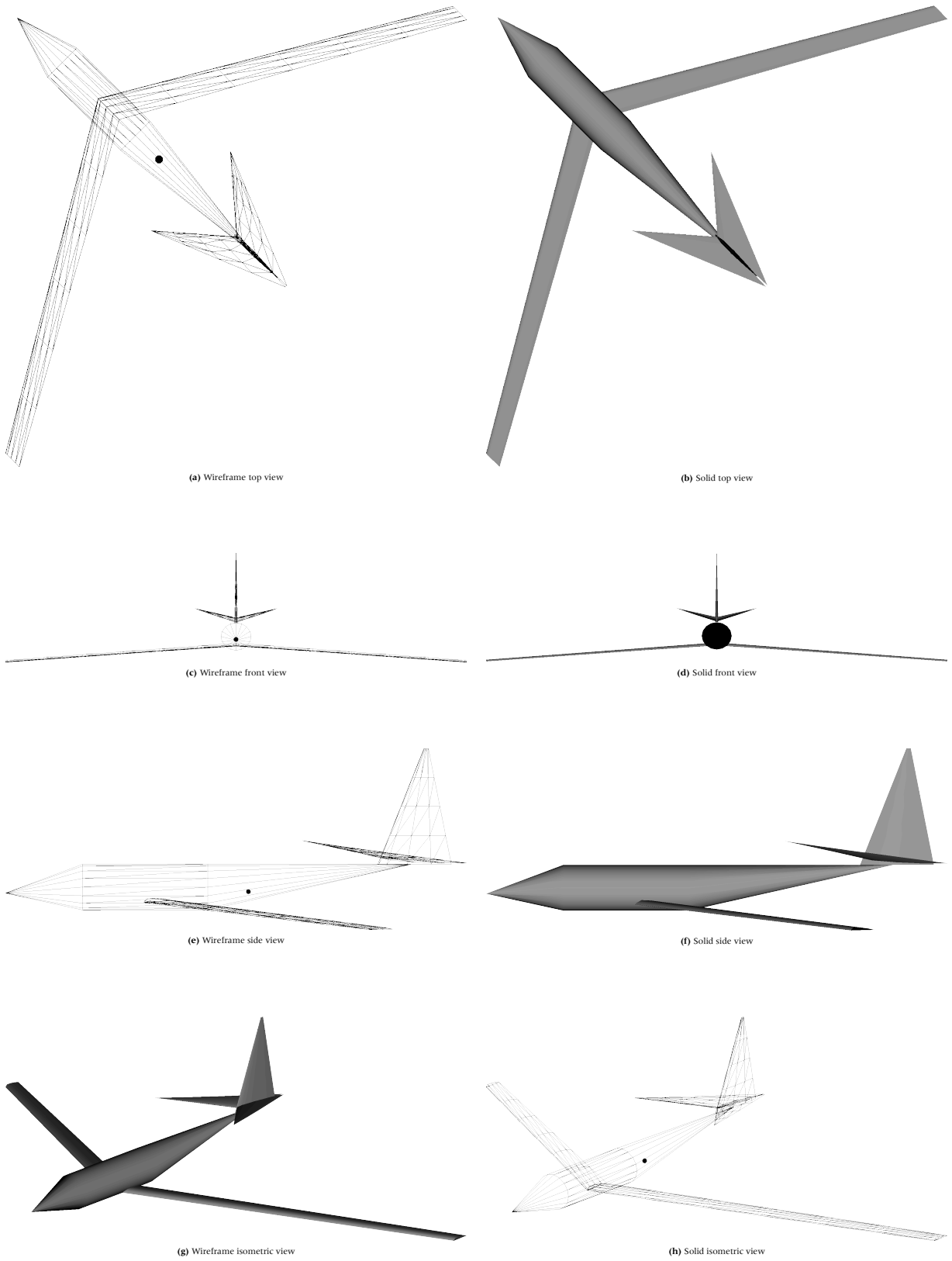


Figure 5.15: Fittest aircraft design concept generated by algorithm DE*

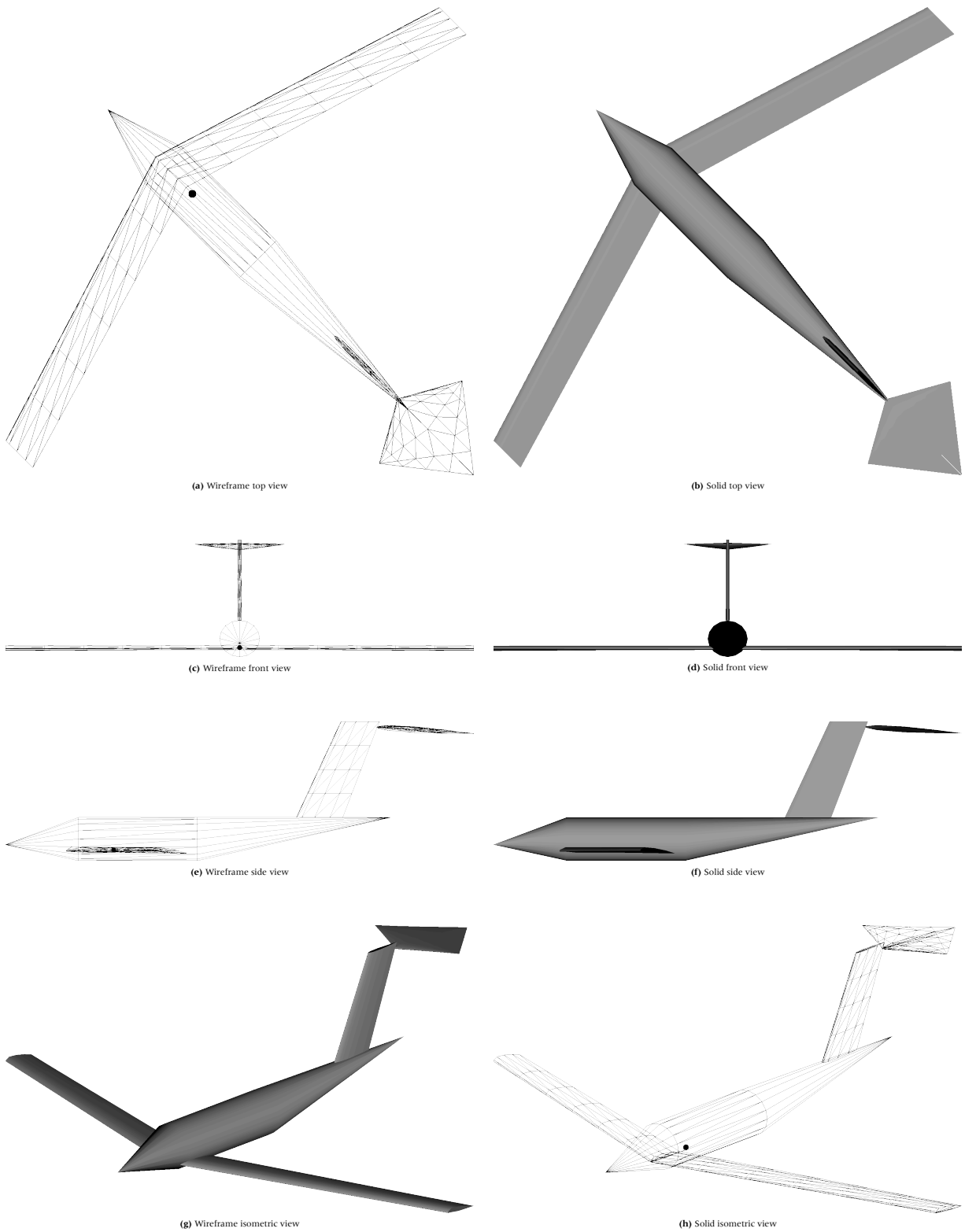


Figure 5.16: Fittest aircraft design concept generated by algorithmic variant CDE*

5.3 Additional experiment: weather enabled

Besides the main experiment, an additional “short” one is conducted by performing 30 executions of the III algorithm, sharing the same initial population of the main experiment as well as simulation settings, objectives and simulation script but introducing different synthetic environment conditions.

Instead of having a constant weather condition consisting of calm wind, no rain and clear sky behavior set on the main experiment, the simulator weather option is enabled in order to introduce variable wind magnitudes, rain, cloud and temperature conditions.

Climatological conditions are randomly generated once (Table 5.9) and then shared across all simulations, the conditions are not, however, completely constant since weather components although they are seeded from the same initial configuration, their actual specific locations and magnitudes are randomly determined at simulation time (within constant ranges).

Clouds			
	Lower layer	Middle (mid) layer	Upper layer
Base	11.222 ft	11.222 ft	16.334 ft
Top	14.334 ft	16.334 ft	...
Type	Scattered cumulus	Cumulus	Clear
Winds			
	Lower layer	Mid layer	Upper layer
Base	2.000 ft	8.000 ft	18.000 ft
Direction	25°	24°	31°
Speed	4 kt	14 kt	11 kt
Gust increase	8 kt	9 kt	7 kt
Shear direction	2°	2°	3°
Precipitation intensity		Severe	
Thermals			
Tops	10.253 ft		
Climb rate	760 ft/min		

Table 5.9: Additional experiment weather conditions

The reason of conducting this additional experiment is to provide additional test data of the strategy performance under semi-uncontrolled simulation conditions, deciding to let weather changes since that environmental aspect may affects drastically the aerodynamic performance of an aircraft, worst if that aircraft has no manual nor automatic control input.

5.3.1 Results

After the execution of the experiment, 6777 design concepts were generated and evaluated Table 5.11 and Fig. 5.19 accounting for almost 93 simulation hours².

Initial population shared by all executions (same population shared by executions of the main experiment) achieved average fitness evaluations of 6429.96, about 15% worse than the main experiment initial fitness evaluations.

The strategy reached an optimal fitness of 96.28, an optimal mean of 117.39 and optimal median of 119.89 (Fig. 5.17 and Table 5.10) for a maximum relative efficacy of 98.50%.

The relative quality with respect of the highest quality achieved in the first experiment was placed at a value of 63.39%, that is, about 30.00% below the average relative quality of the strategy for the first experiment but still considerably higher than the algorithms not based on the strategy.

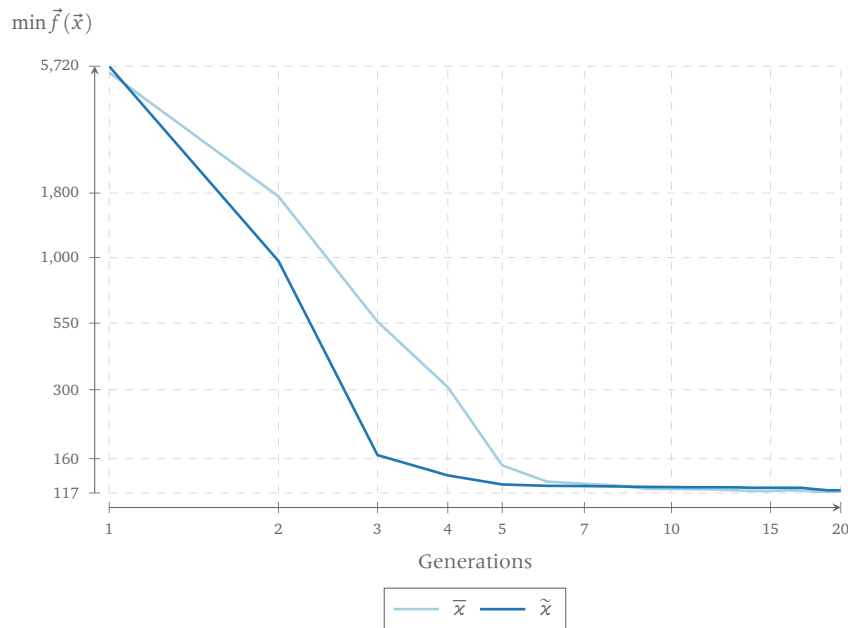


Figure 5.17: Weather enabled experiment convergence curve

$\min \vec{f}(\vec{x})$			$ X_g^E $				Initial fitness	Improvement	Relative efficacy		Relative efficiency	Relative quality
\bar{x}	\tilde{x}	min	\bar{x}	\tilde{x}	min	max	\bar{x}	\bar{x}	max	\bar{x}		
117.39	119.89	96.28	226	235	158	247	6429.96	98.17%	98.50%	0.43	63.39% ³	

Table 5.10: Weather enabled experiment global results

When considering individual objectives Fig. 5.18, even the maximum glide was longer than the average maximum glide achieved by DE* and CDE* algorithms, despite the fact that those

²Simulations take longer in average with respect the main experiment, weather conditions and specially the thermals keep the aircraft longer in air

³With respect of main experiment results

algorithms were conducted under controlled minimal weather conditions. Nevertheless, the

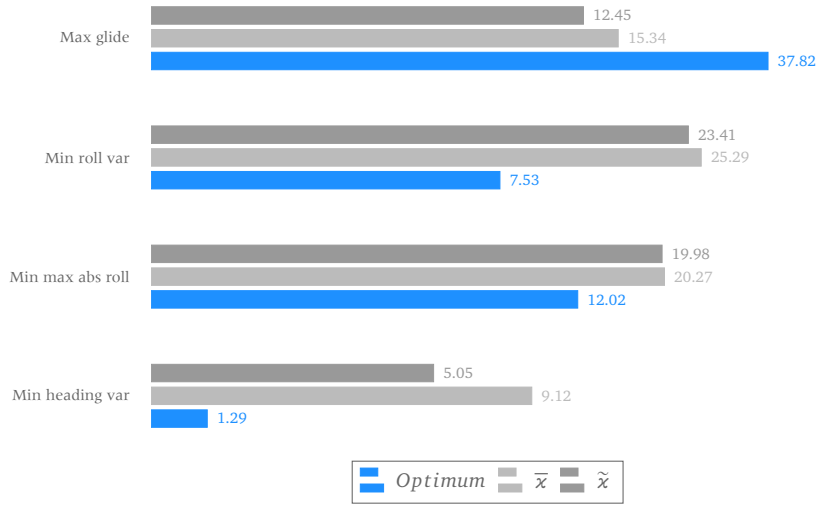


Figure 5.18: Weather enabled experiment objectives

algorithm could hardly produce maximum glides comparable to the first experiment due to the fact that it is virtually impossible that an aircraft without any control input (either manual or automatic) and without propulsion is not perturbed by changing conditions in wind, thermals and rain.

Table 5.11: Additional experiment fittest design concept design variables

Design variable	Value
Fixed values	
MLW	42000lb
S	860 ft ²
S_h	217 ft ²
S_v	170 ft ²
t_y	4.41 ft
n_y	-1.16 ft
$f_{\tilde{n}}$	1.98
$f_{\tilde{m}}$	1.98
fw	9.74 ft
fh	8.83 ft
lm	24.33 ft
lt/d	4.26
ln/d	1.61
$\frac{t_{max}}{c} \mathcal{X}$	30%
$\frac{th_{max}}{c} \mathcal{X}$	30%
$\frac{\delta h_{max}}{c} \mathcal{X}$	0%
$\frac{\delta h_{max}}{c}$	0%
$\frac{\delta v_{max}}{c} \mathcal{X}$	0%

Continue on next page

Table 5.11 – continued from previous page

Design variable	Value
$\frac{\delta v_{max}}{c}$	0%
$\frac{tv_{max}}{c} \chi$	30%
ϵ_v	0°
l_v	0°
Γ_v	90°
$\frac{\delta vh_{max}}{c} \chi$	0%
$\frac{\delta vh_{max}}{c}$	0%
$\frac{tvh_{max}}{c} \chi$	30%
Non-fixed values	
$\frac{th_{max}}{c} [1]$	7
$\epsilon_h [1]$	-3.07°
$l_h [1]$	4.87°
$\Gamma_h [1]$	5°
$\Lambda l_h [1]$	0.25
$\Lambda_h [1]$	-24°
$\lambda_h [1]$	0.48
$\mathcal{R}_h [1]$	3.22
$z_h [1]$	0.75
$y_h [1]$	0.86
$hpos_h [1]$	2
Nh	1
$\frac{tvh_{max}}{c} [1]$	6%
$\epsilon_{vh} [1]$	-3.00°
$l_{vh} [1]$	4.47°
$\Gamma_{vh} [1]$	62°
$\Lambda l_{vh} [1]$	0.00
$\Lambda_{vh} [1]$	70°
$\lambda_{vh} [1]$	0.87
$\mathcal{R}_{vh} [1]$	1.74
$z_{vh} [1]$	0.42
$y_{vh} [1]$	0.17
$x_{vh} [1]$	0.61
$Sw_{vh} [1]$	0.70
$\frac{S_v}{S_{vh}} [1]$	0.34
$\frac{S_h}{S_{vh}} [1]$	0.35
Nvh	1
Nv	0
$\frac{x\delta_{max}}{c} \chi [1]$	0%
$\frac{\delta_{max}}{c} [1]$	0%
$\frac{t_{max}}{c} [1]$	15%
$\epsilon [1]$	-1.13°
$\iota [1]$	0.00°

Continue on next page

Table 5.11 – continued from previous page

Design variable	Value
$\Gamma [1]$	12°
$\Lambda l [1]$	0.25
$\Lambda [1]$	47°
$\lambda [1]$	0.22
$\mathcal{R} [1]$	7.33
$z_w [1]$	0.06
$y_w [1]$	0.82
$hpos_w [1]$	1
Nw	1

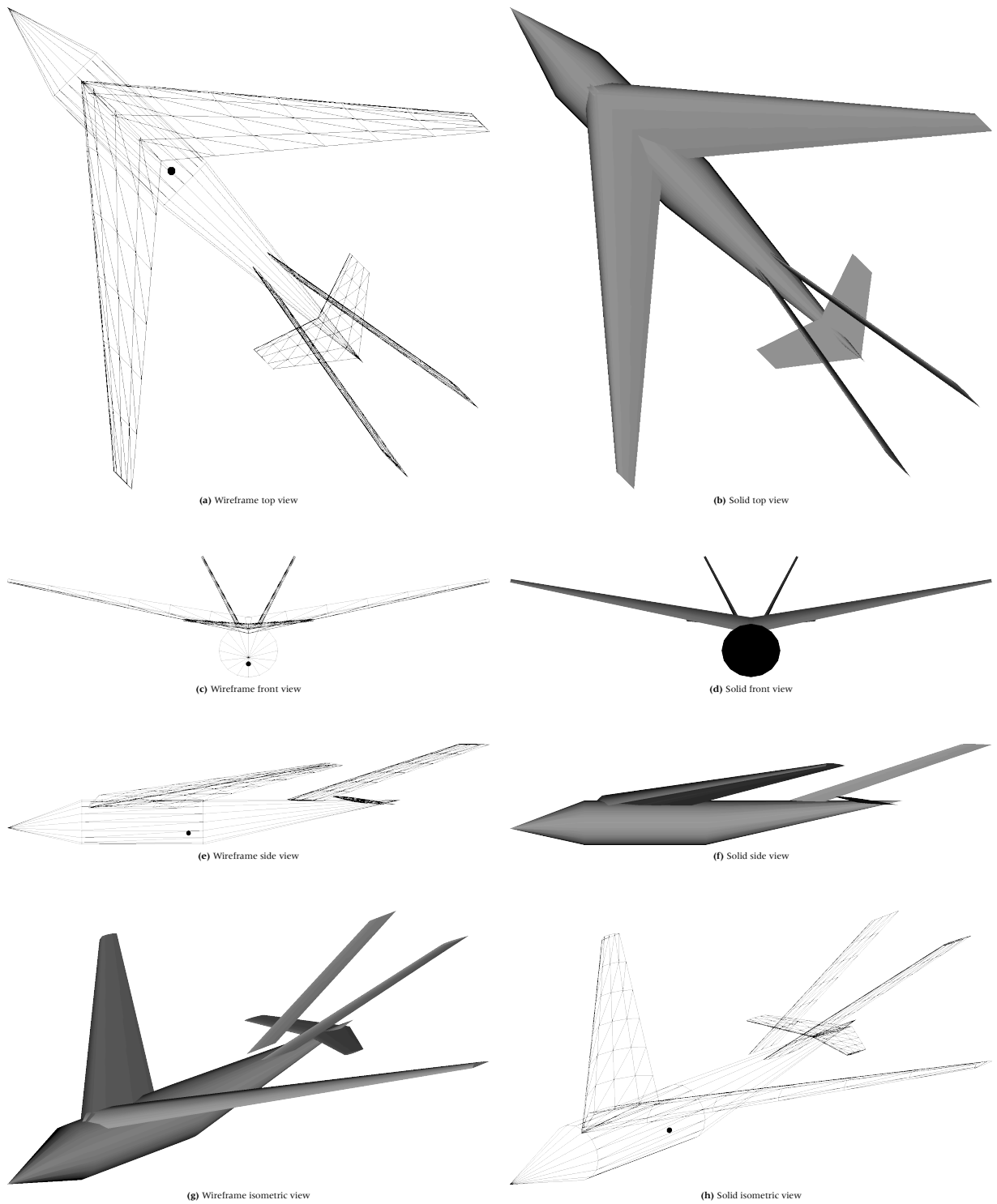


Figure 5.19: Fittest aircraft design concept generated by algorithm III with weather enabled

6

Conclusions

After developing the solution proposed in this research work, it was possible to achieve the general and specific objectives stated at the beginning of the research.

A digital prototype of a fixed-wing aircraft was generated as an optimal solution (with respect of all generated solutions) for a gliding scenario inside a flight simulator. Moreover, an additional short experiment was able to come up with a possible solution inside a less-controlled scenario (weather enabled), again, inside the flight simulator.

The results obtained suggest that domain knowledge can improve the evolutionary design process, probably by the introduction of some kind of exploitation behavior through corrective variations, a behavior generally accepted to be present only on selection operators; yet specially by the introduction of high-level building blocks through preconfigured variations into the design space.

The results also indicate that the mechanisms introduced to deal with the very small population size were effective, preventing the strategy falling in stagnation.

When reviewing the resulting structure of the fittest designs (for each algorithm) in the main experiment, it is clear that none of them retained the vertical stabilizer (installed initially in the shared initial population), except those generated by the differential evolution algorithm based variants.

It is also highlighting that, in almost all cases, the main wing plane was longitudinally placed in the front section of the fuselage and that all design concepts generated by algorithms based on the proposed strategy presented dihedral in both main wing plane and horizontal stabilizer, except those generated by algorithms based on differential evolution, which displayed an anhedral (negative dihedral) and a no-dihedral configurations, an indicator that the dihedral configuration was favored during evaluation possibly by the influence of the corrective variation operator that indeed, contained rules to heal lateral instability by setting or increasing the dihedral effect in wings.

It was an interesting observing through additional experiments how the two types of variation operators, when isolated, did not achieve the best results by their own, neither only evolutionary operators nor knowledge based operators, demonstrating the benefits of their synergy in

this implementation case.

Although desired, it was not possible to define a more elaborated testing scenario, that is, a scenario including flight mission stages like take-off, climbing, cruising, descent and landing; however, the computational model and evolutionary strategy were not precisely the cause to not achieving those scenarios (since additional variables could be kept constants if needed), but the lack of a crucial software component, neither existing nor custom-built, artificial “test pilot” or auto-pilot, that could fly a generated design concept, a component that, for its characteristics could be a full research topic by itself.

During the development of this work, some early decisions had to be made in front of a broad range of possible solution methods and approaches that potentially would cover the stated problem. Starting from accurately placing the problem of aircraft conceptual design automation as an instance of an engineering design optimization problem allowed selecting (by tracing current applicable body of knowledge) a viable and suitable method of solving, as well as the approaches to tackle the problem.

There were some considerations from the beginning that also influenced the path followed, such the idea of using a flight simulator to support evaluation. Even though this work is focused on a specific application case (conceptual aircraft design), it is still a computer science work, therefore, relying on a specialized tool regarding much of domain knowledge specific issues sounded advisable. This first early decision lead the research to take the next important decision supported by existing theory: adopting the evolutionary computing approach and specifically the evolutionary design approach.

This work essentially proposed incorporating pieces of domain knowledge into the evolutionary process. By taking this approach it was expected that the required knowledge to implement the solution would be doubled (the evolutionary computing part of knowledge and the knowledge engineering part of knowledge), presenting a risk regarding the time frame for this research.

For this reason, the focus was set in crafting the evolutionary strategy and the means to represent the solutions, and assume a more practical approach regarding the knowledge engineering part, a decision that resulted in taking an existing rule based framework and integrating it into the solution being developed.

This approach was adequate since that allowed this work to show the effect of incorporating domain knowledge into the evolutionary design process without pretending to build a kind of expert system or being distracted with specific knowledge management algorithms.

Although using the simulator to assist the evaluation of generated design concepts was extremely useful and appropriate to achieve the objectives, it was unavoidable to keep this work too separated (decoupled) from the application case specific domain knowledge to some extent.

Nevertheless, it was a situation that just confirmed what the evolutionary design and meta-heuristic hybridization theory already recommended: to customize and hybridize as much as possible, and to consider specific application domain aspects as part of main principles to apply the evolutionary computing approach into engineering design applications.

It is clear, at least from the perspective of this work experience, that going from the canonical definition of algorithms, methods and approaches into specific applications is far away from

trivial. It requires some understanding of the conceptual and principles of the body of knowledge being applied, mixing and proposing new ideas and mechanisms, avoiding being restricted by original formulations and definitions. This is, in fact an advantage of computer science and computer engineering, and particularly an advantage of artificial intelligence discipline, as they can be (and usually are) inspired by nature, engineering, psychology, science, arts, etc., but are not restricted to that inspiration, nor by available mathematical methods of solution.

Despite the concrete application scenario of this work, there was an interest during the development of the solution in providing as many points for extension as possible (e.g., inheritance for the case of object graphs, rule based system allowing separation between knowledge and inference, generalized evolutionary framework to allow implementing different behaviors, variation operators and algorithms, etc.) that hopefully might help to implement similar solutions into similar problems.

Some ideas had to be postponed to future work such as:

- Incorporating a learning mechanism to the evolutionary process to obtain characterizations of “good” designs.
- A clustering feature that would extract knowledge from the evolutionary process in the form of promising preconfigured variations (synthetically proposed).
- Introducing more variations, perhaps synthetically generated by clustering (from a large set of existing aircraft configurations).
- To append more sophisticated and complete diagnosis and repair rules as corrective variations.
- To implement a non-scalarized multi-objective evolutionary strategy.
- Incorporating constraints other than design variables domain constraints.
- To introduce design variable sensitivity analysis.
- To use aerodynamic coefficients instead of roll and heading variances as method to measure lateral and directional stability.
- ...

From the aircraft conceptual design perspective, some interesting features for further work could be: extending the model to basic aerodynamic analysis for the non-wing type parts of the aircraft (i.e., fuselage), incorporating control surfaces, landing gear and propulsion systems (still from a conceptual perspective), to implement more sophisticated test scenarios assisted by artificial “test pilots”, etc.



Design variables reference

See (Appendix B.3) for visual guidance.

Table A.1: Aircraft design variables reference

Design variable	Context	Domain
Family level		
N_w , Number of wing planes		{1..3}
S_f , Shape family		{'Constant chord', 'Triangular', 'Trapezoidal'}
N_h , Number of horizontal stabilizers		{0..1}
S_{f_h} , Horizontal stabilizer shape family		{'Constant chord', 'Triangular', 'Trapezoidal'}
N_v , Number of vertical stabilizers		{0..1}
S_{f_v} , Vertical stabilizer shape family		{'Constant chord', 'Triangular', 'Trapezoidal'}
N_{vh} , Number of vertical/horizontal stabilizers		{0..1}
$S_{f_{vh}}$, Vertical/horizontal stabilizer shape family		{'Constant chord', 'Triangular', 'Trapezoidal'}
Configuration level		
N_w , Number of wing planes		Inherited
λt , Wing taper ratio type	$S_f = \text{Constant chord}$	{'Untapered'}
	$S_f = \text{Triangular}$	{'Pointed'}
	$S_f = \text{Trapezoidal}$	{'Tapered'}
v_{pos_w} , Wing plane vertical position		{'Low', 'Mid', 'High'}
ARL , Aspect ratio level		{'Very low', 'Low', 'Moderate', 'High', 'Very high', 'Extremely high'}

Continue on next page

Table A.1 – continued from previous page

Design variable	Context	Domain
λl , Wing taper ratio level		{'Low', 'Moderate', 'High'}
Δd , Sweep direction		{'None', 'Forward', 'Back'}
ΔL , Sweep level		{'Slightly', 'Moderately', 'Highly'}
Γt , Wing dihedral angle type		{'No dihedral', 'Dihedral', 'Anhedral'}
ιt , Wing incidence angle type		{'No incidence', 'Low', 'High'}
ϵt , Wing twist type		{'No twist', 'Wash in', 'Wash out'}
$af l t$, Wing airfoil type		{'Symmetrical', 'Cambered'}
$af l T$, Wing airfoil thickness level		{'Thin', 'Medium', 'Thick'}
N_h , Number of horizontal stabilizers		Inherited
λt_h , Horizontal stabilizer taper ratio level	$Sf_h =$ Constant chord	{'Untapered'}
	$Sf_h =$ Triangular	{'Pointed'}
	$Sf_h =$ Trapezoidal	{'Tapered'}
v_{pos_h} , Horizontal stabilizer vertical position		{'Low', 'Mid', 'High'}
$\mathcal{A}l_h$, Horizontal stabilizer aspect ratio level		{'Very low', 'Low', 'Moderate', 'High', 'Very high', 'Extremely high'}
λl_h , Horizontal stabilizer taper ratio level		{'Low', 'Moderate', 'High'}
Δd_h , Horizontal stabilizer sweep direction		{'None', 'Forward', 'Back'}
ΔL_h , Horizontal stabilizer sweep level		{'Slightly', 'Moderately', 'Highly'}
Γt_h , Horizontal stabilizer dihedral angle type		{'No dihedral', 'Dihedral', 'Anhedral'}
ιt_h , Horizontal stabilizer incidence angle type		{'No incidence', 'Low', 'High'}
ϵt_h , Horizontal stabilizer twist type		{'No twist', 'Wash in', 'Wash out'}
$af l t_h$, Horizontal stabilizer airfoil type		{'Symmetrical', 'Cambered'}
N_v , Number of vertical stabilizers		Inherited
λt_v , Vertical stabilizer taper ratio level	$Sf_v =$ Constant chord	{'Untapered'}
	$Sf_v =$ Triangular	{'Pointed'}
	$Sf_v =$ Trapezoidal	{'Tapered'}
λl_v , Vertical stabilizer taper ratio level		{'Low', 'Moderate', 'High'}
ΔL_v , Vertical stabilizer sweep level		{'Slightly', 'Moderately', 'Highly'}

Continue on next page

Table A.1 – continued from previous page

Design variable	Context	Domain
N_{vh} , Number of vertical/horizontal stabilizers		Inherited
λt_{vh} , Vertical/horizontal stabilizer taper ratio level	$Sf_{vh} =$ Constant chord	{‘Untapered’}
	$Sf_{vh} =$ Triangular	{‘Pointed’}
	$Sf_{vh} =$ Trapezoidal	{‘Tapered’}
λl_{vh} , Vertical/horizontal stabilizer taper ratio level		{‘Low’, ‘Moderate’, ‘High’}
Λd_{vh} , Vertical/horizontal stabilizer sweep direction		{‘None’, ‘Forward’, ‘Back’}
ΛL_{vh} , Vertical/horizontal stabilizer sweep level		{‘Slightly’, ‘Moderately’, ‘Highly’}
Γt_{vh} , Vertical/horizontal stabilizer dihedral angle type		{‘Dihedral’, ‘Anhedral’}
tt_{vh} , Vertical/horizontal stabilizer incidence angle type		{‘No incidence’, ‘Low’, ‘High’}
ϵt_{vh} , Vertical/horizontal stabilizer twist type		{‘No twist’, ‘Wash in’, ‘Wash out’}
Prototype level		
MLW , Maximum landing weight in lb		{42000.00}
S , Planform surface: the wing shape projected surface, when viewed from above in ft ²		{860.00}
N_w , Number of wing planes		Inherited
$hpos_w$, Wing plane horizontal position		{‘Nose’, ‘Center’, ‘Tail’}
z_w , Wing plane relative longitudinal arm		[0.0, 1.0]
y_w , Wing plane relative vertical arm	$vpos_w =$ Low	[0.0, 0.25]
	$vpos_w =$ Mid	(0.25, 0.75)
	$vpos_w =$ High	[0.75, 1.0]
\mathcal{R} , Aspect ratio: the ratio of the wing span to the chord length	$\mathcal{R}l =$ Very low	[0.3, 1.0]
	$\mathcal{R}l =$ Low	(1.0, 3.0]
	$\mathcal{R}l =$ Moderate	(3.0, 7.0]
	$\mathcal{R}l =$ High	(7.0, 12.0]
	$\mathcal{R}l =$ Very high	(12.0, 20.0]
λ , Taper ratio: the ratio of the wing chord length at root to the chord length at tip	$\mathcal{R}l =$ Extremely high	(20.0, 40.00]
	$\lambda t =$ Pointed	[0.0, 0.0]
	$\lambda t =$ Tapered	(0.0, 1.0)
	$\lambda t =$ Untapered	[1.0, 1.0]
	$\lambda l =$ Low	(0.0, 0.3)
Λ , Sweep angle: the angle between the lateral axis of the aircraft and a chord-line going from a chordwise point at the wing root to the same chordwise point at the tip	$\lambda l =$ Moderate	[0.3, 0.6)
	$\lambda l =$ High	[0.6, 1.0)
	$\Lambda d =$ None	{0}
	$\Lambda d =$ Back	{1..65}

Continue on next page

Table A.1 – continued from previous page

Design variable	Context	Domain
	$\Lambda d = \text{Forward}$	$\{-65.. -1\}$
	$\Lambda L = \text{Slightly}$	$\{1..10\} \cup \{-10.. -1\}$
	$\Lambda L = \text{Moderately}$	$\{11..29\} \cup \{-29.. -11\}$
	$\Lambda L = \text{Highly}$	$\{30..65\} \cup \{-65.. -30\}$
Λl , Sweep location: a location expressed as a percentage of the wing chord length (starting at leading edge) where a sweep angle is measured		$[0.0, 0.0] \cup [0.25, 0.25] \cup [0.50, 0.50] \cup [1.0, 1.0]$
Γ , Dihedral angle: the angle between the wing and the aircraft lateral axis, when viewed from the front	$\Gamma t = \text{No dihedral}$	$\{0\}$
	$\Gamma t = \text{Dihedral}$	$\{1..20\}$
	$\Gamma t = \text{Anhedral}$	$\{-20.. -1\}$
ι , Incidence: the angle between the the chord line of the wing at root and the aircraft longitudinal axis, when viewed from the side	$\iota t = \text{No incidence}$	$[0.0, 0.0]$
	$\iota t = \text{Low}$	$(0.0, 4.0)$
	$\iota t = \text{High}$	$[4.0, 6.0]$
ϵ , Twist: The absolute difference between the incidence angle at the root and the incidence angle at the tip. Twist is negative if the leading edge of the tip is below the root leading edge, and positive otherwise	$\epsilon t = \text{No twist}$	$[0.0, 0.0]$
	$\epsilon t = \text{Wash in}$	$(0.0, 4.0)$
	$\epsilon t = \text{Wash out}$	$[-4.0, 0.0]$
$\frac{t_{max}}{c}$, Airfoil max thickness	$afLT = \text{Thin}$	$\{6..8\}$
	$afLT = \text{Medium}$	$\{9..12\}$
	$afLT = \text{Thick}$	$\{13..18\}$
$\frac{t_{max}}{c}x$, Airfoil max thickness location		$\{30\}$
$\frac{\delta_{max}}{c}$, Airfoil max camber	$afLT = \text{Symmetrical}$	$\{0\}$
	$afLT = \text{Cambered}$	$\{1..9\}$
$\frac{x\delta_{max}}{c}x$, Airfoil max camber location	$afLT = \text{Symmetrical}$	$\{0\}$
	$afLT = \text{Cambered}$	$\{10, 20, 30, 40, 50, 60\}$
l_n/d , Fuselage nose section (fwd) length to diameter ratio		1.61
l_t/d , Fuselage tail section (aft) length to diameter ratio		4.26
l_m , Fuselage mid section length		24.33
f_h , Fuselage height		8.83
f_w , Fuselage width		9.74
$f_{\tilde{m}}$, Mid fuselage cross section superellipse \tilde{m} parameter		1.98
$f_{\tilde{n}}$, Mid fuselage cross section superellipse \tilde{n} parameter		1.98
f_{Cd} , Fuselage drag coefficient		0.075
n_y , Fuselage nose tip vertical location		-1.16

Continue on next page

Table A.1 – continued from previous page

Design variable	Context	Domain
t_y , Fuselage tail tip vertical location		4.41
S_h , Horizontal tail planform surface		{217.00}
N_h , Number of horizontal stabilizers		Inherited
$hpos_h$, Horizontal stabilizer horizontal position		{'Canard', 'Aft'}
z_h , Horizontal stabilizer relative longitudinal arm		[0.0, 1.0]
y_h , Horizontal stabilizer relative vertical arm	$vpos_h = \text{Low}$	[0.0, 0.25]
	$vpos_h = \text{Mid}$	(0.25, 0.75)
	$vpos_h = \text{High}$	[0.75, 1.0]
$\mathcal{A}l_h$, Horizontal stabilizer aspect ratio	$\mathcal{A}l_h = \text{Very low}$	[0.3, 1.0]
	$\mathcal{A}l_h = \text{Low}$	(1.0, 3.0]
	$\mathcal{A}l_h = \text{Moderate}$	(3.0, 7.0]
	$\mathcal{A}l_h = \text{High}$	(7.0, 12.0]
	$\mathcal{A}l_h = \text{Very high}$	(12.0, 20.0]
λ_h , Horizontal stabilizer taper ratio	$\lambda l_h = \text{Extremely high}$	(20.0, 40.00]
	$\lambda t_h = \text{Pointed}$	[0.0, 0.0]
	$\lambda t_h = \text{Tapered}$	(0.0, 1.0)
	$\lambda t_h = \text{Untapered}$	[1.0, 1.0]
	$\lambda l_h = \text{Low}$	(0.0, 0.3)
Λ_h , Horizontal stabilizer sweep angle	$\lambda l_h = \text{Moderate}$	[0.3, 0.6]
	$\lambda l_h = \text{High}$	[0.6, 1.0]
	$\Lambda d_h = \text{None}$	{0}
	$\Lambda d_h = \text{Back}$	{1..65}
	$\Lambda d_h = \text{Forward}$	{-65.. -1}
ΛL_h , Horizontal stabilizer sweep location	$\Lambda L_h = \text{Slightly}$	{1..10} \cup {-10.. -1}
	$\Lambda L_h = \text{Moderately}$	{11..29} \cup {-29.. -11}
	$\Lambda L_h = \text{Highly}$	{30..65} \cup {-65.. -30}
Γ_h , Horizontal stabilizer dihedral angle	$\Gamma t_h = \text{No dihedral}$	{0}
	$\Gamma t_h = \text{Dihedral}$	{1..20}
	$\Gamma t_h = \text{Anhedral}$	{-20.. -1}
ι , Incidence: the angle between the the chord line of the wing at root and the aircraft longitudinal axis, when viewed from the side	$\iota t_h = \text{No incidence}$	[0.0, 0.0]
	$\iota t_h = \text{Low}$	(0.0, 4.0)
	$\iota t_h = \text{High}$	[4.0, 6.0]
ϵ_h , Horizontal stabilizer twist	$\epsilon t_h = \text{No twist}$	[0.0, 0.0]
	$\epsilon t_h = \text{Wash in}$	(0.0, 4.0)
	$\epsilon t_h = \text{Wash out}$	[-4.0, 0.0)

Continue on next page

Table A.1 – continued from previous page

Design variable	Context	Domain
$\frac{th_{max}}{c}$, Horizontal stabilizer airfoil max thickness		{6..9}
$\frac{th_{max}}{c}x$, Horizontal stabilizer airfoil max thickness location		{30}
$\frac{\delta h_{max}}{c}$, Horizontal stabilizer airfoil max camber	$afl_{t_h} = \text{Symmetrical}$	{0}
	$afl_{t_h} = \text{Cambered}$	{1..9}
$\frac{\delta h_{max}}{c}x$, Horizontal stabilizer airfoil max camber location	$afl_{t_h} = \text{Symmetrical}$	{0}
	$afl_{t_h} = \text{Cambered}$	{10, 20, 30, 40, 50, 60}
S_v , Vertical tail planform surface		{170.00}
N_v , Number of vertical stabilizers		Inherited
z_v , Vertical stabilizer relative longitudinal arm		[0.0, 1.0]
\mathcal{R}_v , Vertical stabilizer aspect ratio: the ratio of the height of the vertical stabilizer to the chord length		(1.0, 3.0]
λ_v , Vertical stabilizer taper ratio	$\lambda t_v = \text{Pointed}$	[0.0, 0.0]
	$\lambda t_v = \text{Tapered}$	(0.0, 1.0)
	$\lambda t_v = \text{Untapered}$	[1.0, 1.0]
	$\lambda l_v = \text{Low}$	(0.0, 0.3)
	$\lambda l_v = \text{Moderate}$	[0.3, 0.6]
	$\lambda l_v = \text{High}$	[0.6, 1.0]
Λ_v , Vertical stabilizer sweep angle		{0..60}
	$\Lambda L_v = \text{Slightly}$	{1..10}
	$\Lambda L_v = \text{Moderately}$	{11..29}
	$\Lambda L_v = \text{Highly}$	{30..60}
ΛL_v , Vertical stabilizer sweep location		[0.0, 0.0] \cup [0.25, 0.25] \cup [0.50, 0.50] \cup [1.0, 1.0]
Γ_v , Vertical stabilizer dihedral angle		{90}
ι , Incidence: the angle between the the chord line of the wing at root and the aircraft longitudinal axis, when viewed from the side		[0.0, 0.0]
ϵ_v , Vertical stabilizer twist		[0.0, 0.0]
$\frac{tv_{max}}{c}$, Vertical stabilizer airfoil max thickness		{6..12}
$\frac{tv_{max}}{c}x$, Vertical stabilizer airfoil max thickness location		{30}
$\frac{\delta v_{max}}{c}$, Vertical stabilizer airfoil max camber		{0}
$\frac{\delta v_{max}}{c}x$, Vertical stabilizer airfoil max camber location		{0}
N_{vh} , Number of vertical/horizontal stabilizers		Inherited
z_{vh} , Vertical/horizontal stabilizer relative longitudinal arm		[0.0, 1.0]
x_{vh} , Vertical/horizontal stabilizer relative lateral arm		[0.0, 1.0]

Continue on next page

Table A.1 – continued from previous page

Design variable	Context	Domain
y_{vh} , Vertical/horizontal stabilizer relative vertical arm		[0.0, 1.0]
\mathcal{R}_{vh} , Vertical/horizontal stabilizer aspect ratio		(1.0, 3.0]
λ_{vh} , Vertical/horizontal stabilizer taper ratio	$\lambda t_{vh} = \text{Pointed}$	[0.0, 0.0]
	$\lambda t_{vh} = \text{Tapered}$	(0.0, 1.0)
	$\lambda t_{vh} = \text{Untapered}$	[1.0, 1.0]
	$\lambda l_{vh} = \text{Low}$	(0.0, 0.3)
	$\lambda l_{vh} = \text{Moderate}$	[0.3, 0.6)
Λ_{vh} , Vertical/horizontal stabilizer sweep angle	$\Lambda d_{vh} = \text{None}$	{0}
	$\Lambda d_{vh} = \text{Back}$	{1..80}
	$\Lambda d_{vh} = \text{Forward}$	{-80.. -1}
	$\Lambda L_{vh} = \text{Slightly}$	{1..10} \cup {-10.. -1}
	$\Lambda L_{vh} = \text{Moderately}$	{11..29} \cup {-29.. -11}
$\Lambda L_{vh} = \text{Highly}$		{30..80} \cup {-80.. -30}
Λl_{vh} , Vertical/horizontal stabilizer sweep location		[0.0, 0.0] \cup [0.25, 0.25] \cup [0.50, 0.50] \cup [1.0, 1.0]
Γ_{vh} , Vertical/horizontal stabilizer dihedral angle	$\Gamma t_{vh} = \text{Dihedral}$	{25..90}
	$\Gamma t_{vh} = \text{Anhedral}$	{-90.. -25}
ι , Incidence: the angle between the the chord line of the wing at root and the aircraft longitudinal axis, when viewed from the side	$\iota t_{vh} = \text{No incidence}$	[0.0, 0.0]
	$\iota t_{vh} = \text{Low}$	(0.0, 4.0)
	$\iota t_{vh} = \text{High}$	[4.0, 6.0]
ϵ_{vh} , Vertical/horizontal stabilizer twist	$\epsilon t_{vh} = \text{No twist}$	[0.0, 0.0]
	$\epsilon t_{vh} = \text{Wash in}$	(0.0, 4.0)
	$\epsilon t_{vh} = \text{Wash out}$	[-4.0, 0.0]
$\frac{t_{vh_{max}}}{c}$, Vertical/horizontal stabilizer airfoil max thickness		{6..12}
$\frac{t_{vh_{max}}}{c} x$, Vertical/horizontal stabilizer airfoil max thickness location		{30}
$\frac{\delta_{vh_{max}}}{c}$, Vertical/horizontal stabilizer airfoil max camber		{0}
$\frac{\delta_{vh_{max}}}{c} x$, Vertical/horizontal stabilizer airfoil max camber location		{0}
$\frac{S_h}{S_{vh}}$, Vertical/horizontal stabilizer - horizontal tail ratio		[0.33, 0.45]
$\frac{S_v}{S_{vh}}$, Vertical/horizontal stabilizer - vertical tail ratio		[0.33, 0.45]
Sw_{vh} , Vertical/horizontal stabilizer planform surface		[0.65, 0.75]

B

Aircraft various reference views

B.1 Aircraft axes

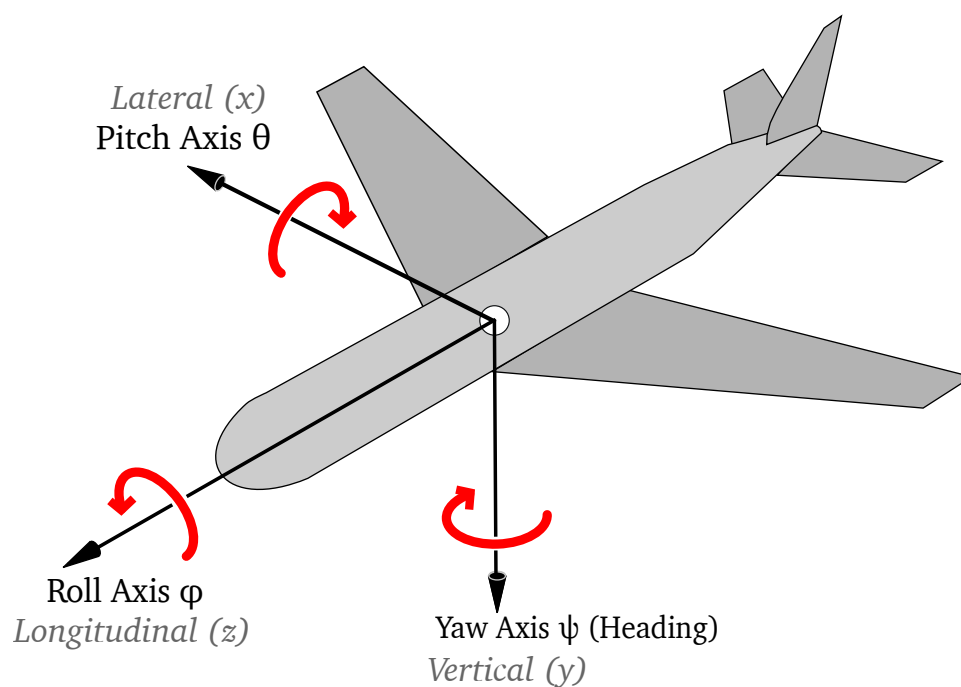


Figure B.1: Aircraft axes

Source: "An image showing all three axes" by Wikimedia Commons user Auawise, modified by Wikimedia Commons user Jrvz, used under CC BY-SA-3.0 / Correct sense of "roll" axis rotation, adjust proportions from original / Labeling and color modifications from derivative[6]

B.2 Aircraft directions

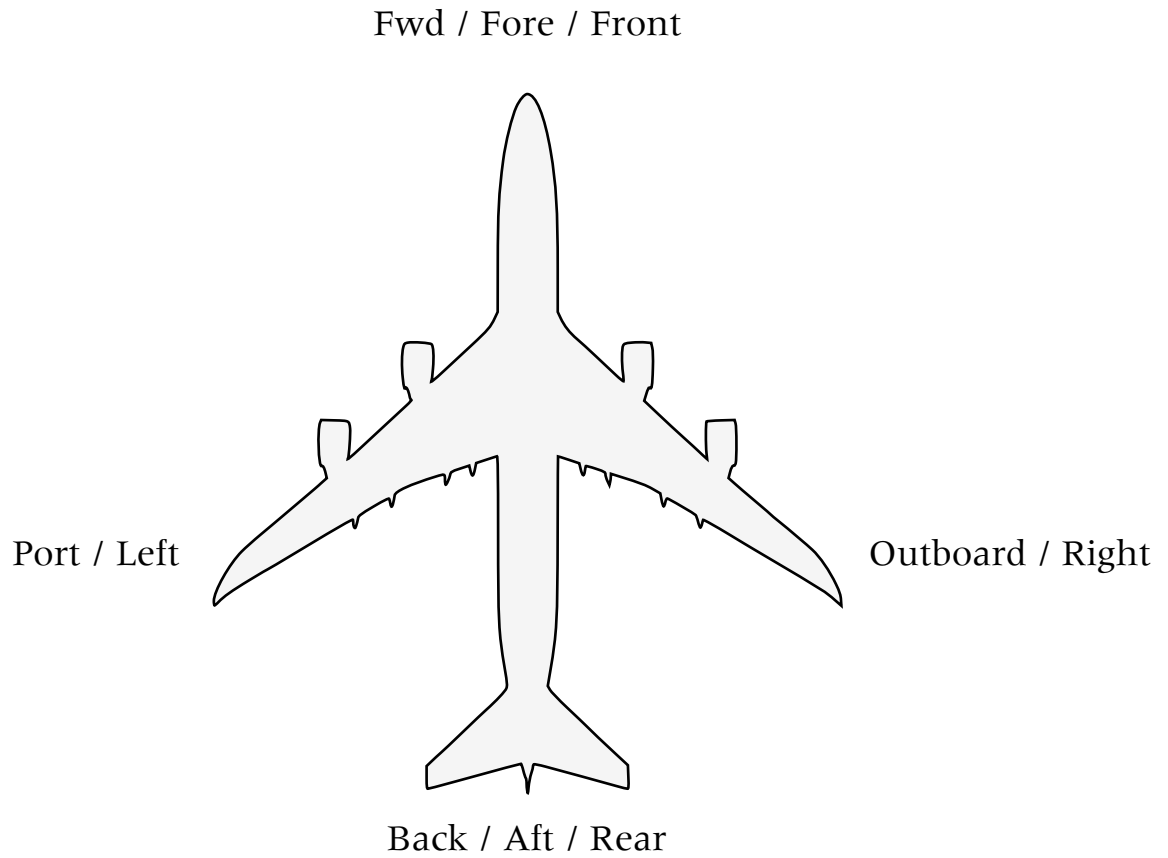


Figure B.2: Aircraft directions

Source: "Giant planes comparison" by Wikimedia Commons user Clem Tillier, used under [CC BY-SA-2.5](#) /
Extracted jet silhouette from original[75]

B.3 Aircraft design variables

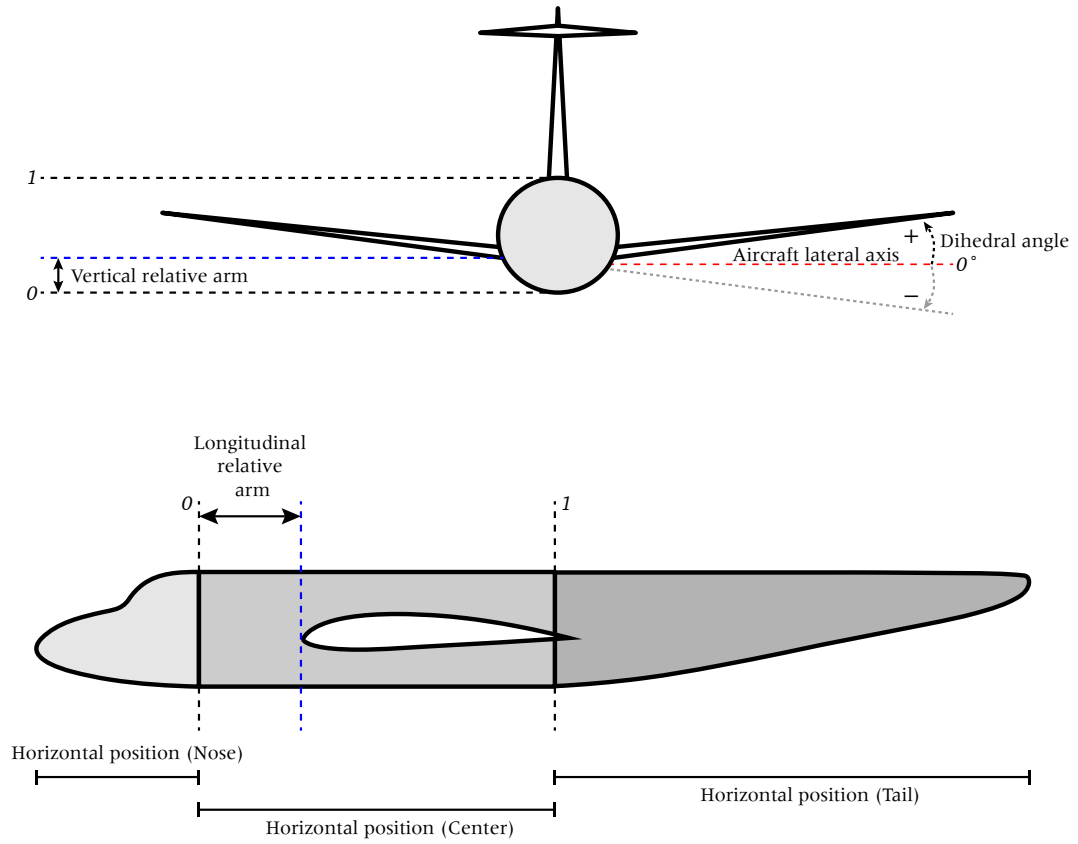


Figure B.3: Wing type part design variables I

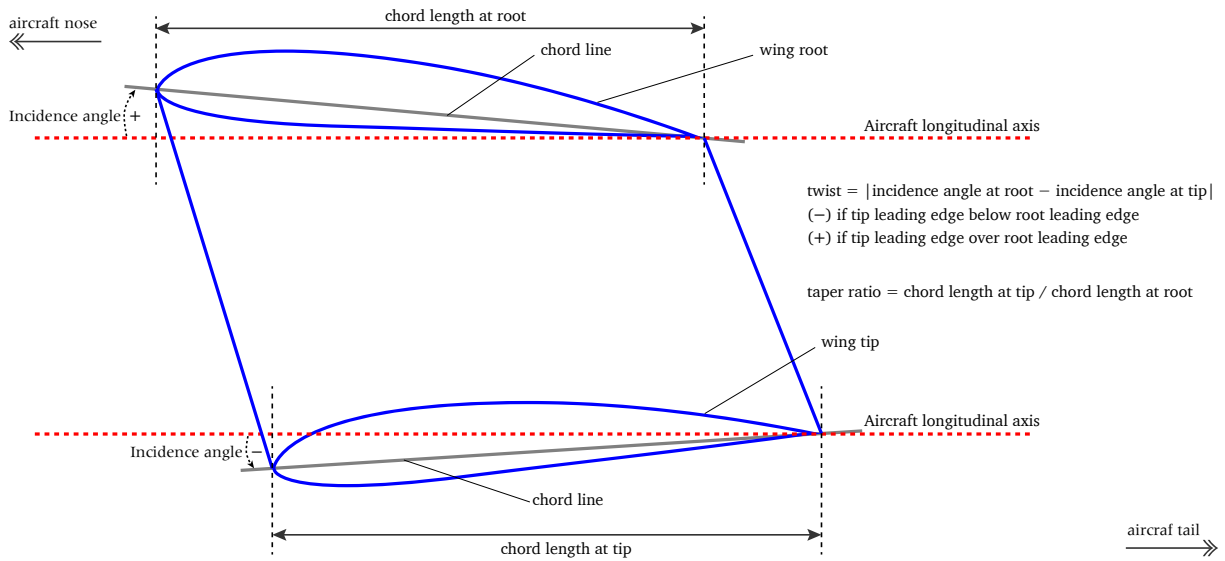


Figure B.4: Wing type part design variables II

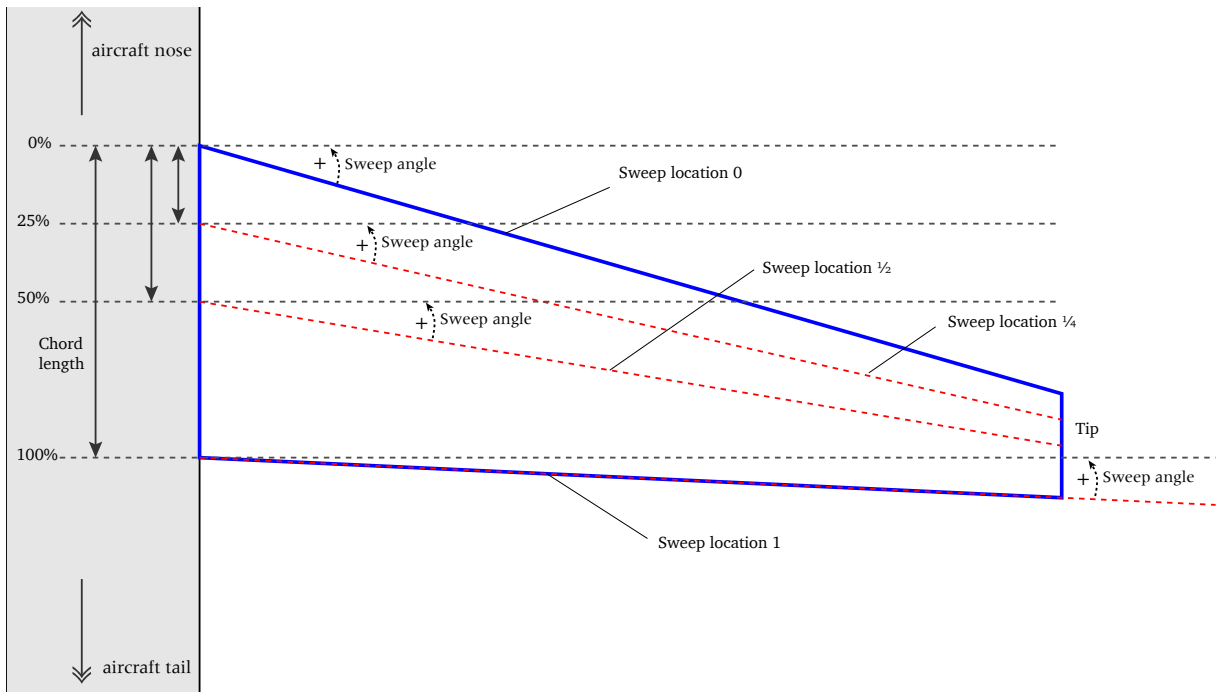


Figure B.5: Wing type part design variables III

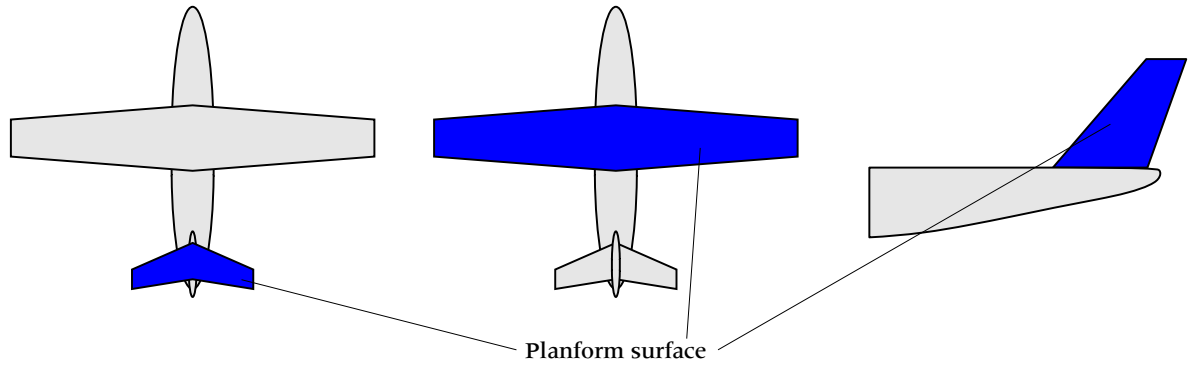


Figure B.6: Wing type part design variables IV

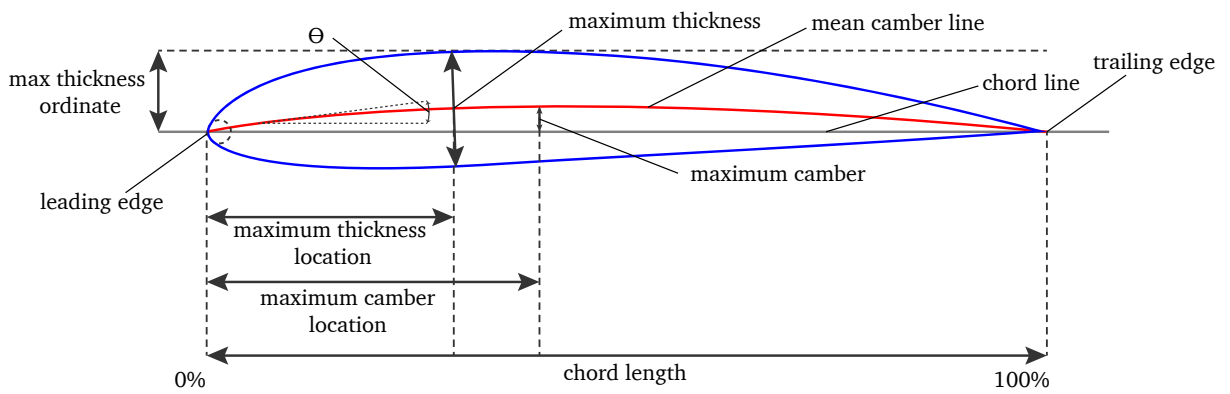


Figure B.7: Wing type part profile design variables

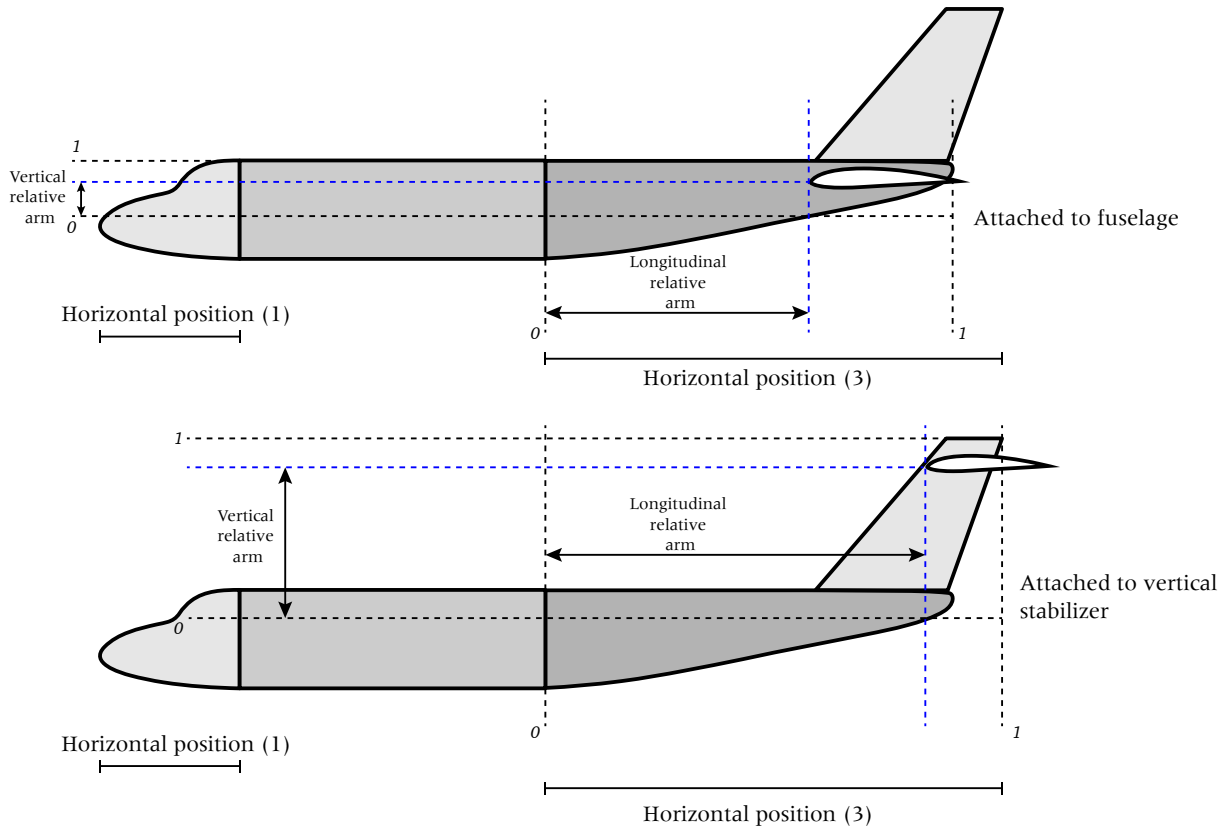


Figure B.8: Horizontal stabilizer design variables (partial)

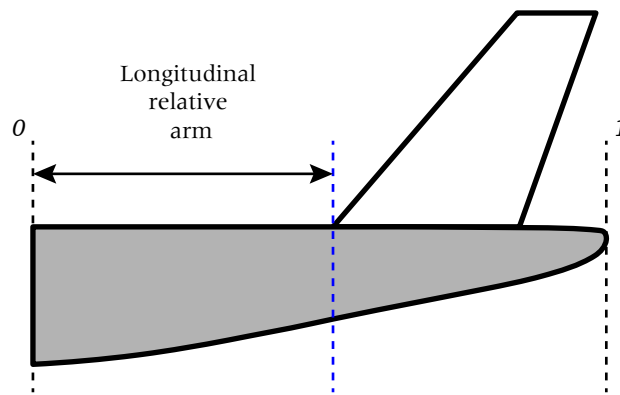


Figure B.9: Vertical stabilizer design variables (partial)

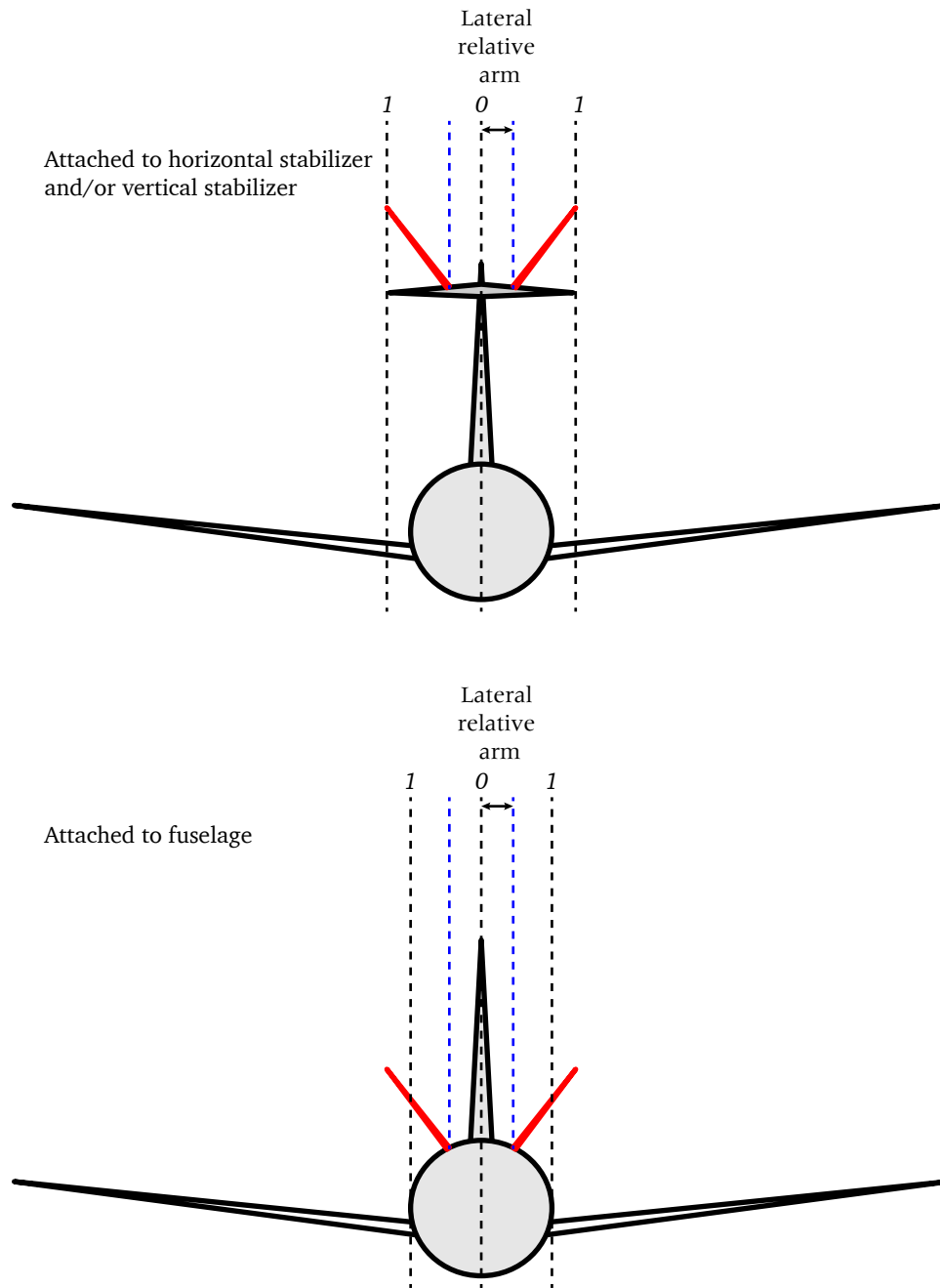


Figure B.10: Vertical/horizontal stabilizer design variables (partial)

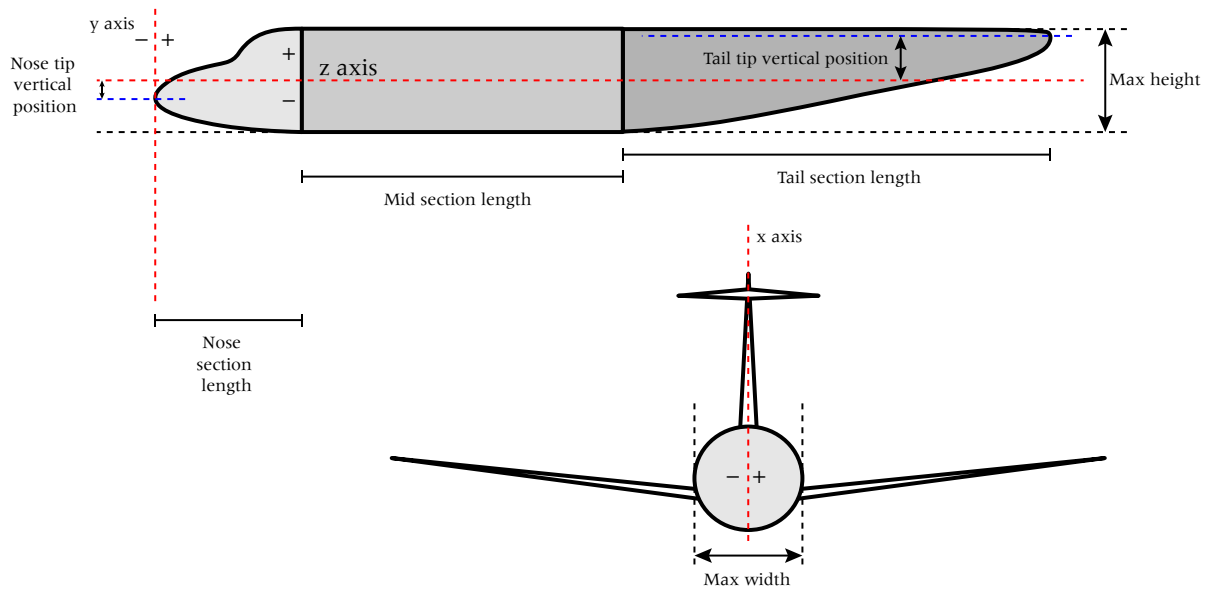



Figure B.11: Fuselage design variables


References

- [1] I. H. Abbott and A. E. Von Doenhoff, *Theory of Wing Sections, Including a Summary of Airfoil Data*, Corrected reprint of the 1st (1949) edition. New York: Dover Publications, 1959, 693 pp., ISBN: 9780486605869.
- [2] J. S. Agte, N. K. Borer, and O. de Weck, "A simulation-based design model for analysis and optimization of multi-state aircraft performance", *51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2010.
- [3] J. Alonso, P. LeGresley, and V. Pereyra, "Aircraft design optimization", *Mathematics and Computers in Simulation*, vol. 79, no. 6, pp. 1948–1958, 2009, ISSN: 0378-4754. DOI: <https://doi.org/10.1016/j.matcom.2007.07.001>.
- [4] M. M. Andreasen, C. T. Hansen, and P. Cash, *Conceptual Design, Interpretations, Mindset and Models*, ser. Studies in Systems, Decision and Control. Springer Switzerland, 2015, 400 pp., ISBN: 9783319198392. DOI: 10.1007/978-3-319-19839-2.
- [5] J. S. Arora, *Introduction to Optimum Design*, 3rd ed. Elsevier, 2012, 919 pp., ISBN: 9780123813756.
- [6] Auawise and Jrvz, Wikimedia Commons user. (Feb. 10, 2010). An image showing all three axes, [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/c/c1/Yaw_Axis_Corrected.svg (visited on 03/18/2017). "An image showing all three axes" by Wikimedia Commons user Auawise, modified by Wikimedia Commons user Jrvz, used under  BY-SA-3.0 / Correct sense of "roll" axis rotation, adjust proportions from original / Labeling and color modifications from derivative.
- [7] Autodesk, Inc. (Dec. 15, 2015). Loft (command), [Online]. Available: <https://knowledge.autodesk.com/support/autocad/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/AutoCAD-Core/files/GUID-0A041818-2E32-4212-A3D8-CE0361C3D229-htm.html> (visited on 03/21/2017).
- [8] T. Bäck, "Introduction", in *Handbook of Evolutionary Computation*, ser. Computational Intelligence Library 2, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., .B - Fundamental Concepts of Evolutionary Computation, Bristol, UK: Institute of Physics Publishing and Oxford University Press, 1997, ch. B1 - Evolutionary Algorithms and their Standard Instances, B1.1:1, B1.1:4, ISBN: 9781420050387. DOI: 10.1201/9781420050387.
- [9] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm", in *Proceedings of the Second International Conference on Genetic Algorithms*, J. J. Grefenstette, Ed., Lawrence Erlbaum Associates, Publishers, 1987, pp. 14–21.
- [10] C. Blum, R. Chiong, M. Clerc, K. De Jong, Z. Michalewicz, F. Neri, and T. Weise, "Evolutionary optimization", in *Variants of Evolutionary Algorithms for Real-World Applications*, R. Chiong, T. Weise, and Z. Michalewicz, Eds. Berlin-Heidelberg: Springer-Verlag, 2012, ISBN: 9783642234248. DOI: 10.1007/978-3-642-23424-8.
- [11] J. Byrne, P. Cardiff, A. Brabazon, and M. O'Neill, "Evolving parametric aircraft models for design exploration and optimisation", *Neurocomputing*, vol. 142, pp. 39–47, 2014, ISSN: 0925-2312. DOI: <https://doi.org/10.1016/j.neucom.2014.04.004>.
- [12] *Cambridge Advanced Learner's Dictionary & Thesaurus*, Cambridge University Press, 2017. [Online]. Available: <https://dictionary.cambridge.org> (visited on 05/25/2017).
- [13] CheCheDaWaff, Wikimedia Commons user. (Apr. 14, 2016). Illustration of great-circle distance, [Online]. Available: <https://upload.wikimedia.org/wikipedia/commons/c/cb/>

- Illustration_of_great-circle_distance.svg (visited on 03/08/2017). “A diagram illustrating great-circle distance” by Wikimedia Commons user CheCheDaWaff, used under © BY-SA-4.0 / Antipodal points removed from original.
- [14] C. A. C. Coello, D. A. Van Veldhuizen, and G. B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, ser. Genetic Algorithms and Evolutionary Computation. New York, NY: Kluwer Academic/Plenum Publishers, 2002, vol. 5, 576 pp., ISBN: 9780306467622.
- [15] M. Crepinšek, S.-H. Liu, and M. Mernik, “Exploration and exploitation in evolutionary algorithms: A survey”, *ACM Comput. Surv.*, vol. 45, no. 3, 35:1–35:33, Jul. 2013, ISSN: 0360-0300. DOI: 10.1145/2480741.2480752. [Online]. Available: <http://doi.acm.org/etechconricyct.idm.oclc.org/10.1145/2480741.2480752>.
- [16] Y. J. Crispin, “Aircraft conceptual optimization using simulated evolution”, in *32nd Aerospace Sciences Meeting and Exhibit*, Reno, NV: American Institute of Aeronautics and Astronautics, 1994.
- [17] R. M. Cummings, S. A. Morton, W. H. Mason, and D. R. McDaniel, “Geometry of aerodynamics, A modern engineering approach”, in *Applied Computational Aerodynamics*. Cambridge University Press, 2015, ch. Appendix A, pp. 731–765, ISBN: 9781107053748.
- [18] “Objective function”, in *International Encyclopedia of the Social Sciences*, W. A. Darity Jr., Ed., 2nd ed. Macmillan Reference USA, 2008, pp. 5–6, ISBN: 9780028659657.
- [19] A. De Gaspari and S. Ricci, “Knowledge-based shape optimization of morphing wing for more efficient aircraft”, *International Journal of Aerospace Engineering*, 2015.
- [20] K. A. De Jong, “An analysis of the behavior of a class of genetic adaptive systems”, PhD thesis, University of Michigan.
- [21] K. A. De Jong, “Generalized evolutionary algorithms”, in *Handbook of Natural Computing*, T. Bäck, G. Rozenberg, and J. N. Kok, Eds., ser. Springer Reference. Springer Berlin, 2012, vol. 4, ch. 20, pp. 625–636, ISBN: 9783540929109. DOI: 10.1007/978-3-540-92910-9.
- [22] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, ser. Wiley Interscience Series in Systems and Optimization. John Wiley & Sons, 2001, vol. 16, 497 pp., ISBN: 9780471873396.
- [23] —, “Evolutionary design in engineering, Advances in evolutionary design”, in *Design by Evolution*, P. F. Hingston, L. C. Barone, and Z. Michalewicz, Eds., ser. Natural Computing Series. Springer Berlin, 2008, pp. 267–271, ISBN: 9783540741114.
- [24] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard, *Metaheuristics for Hard Optimization, Simulated Annealing, Tabu Search, Evolutionary and Genetic Algorithms, Ant Colonies, ... - Methods and Case Studies*. Berlin-Heidelberg: Springer-Verlag, 2006, 372 pp., ISBN: 9783540230229.
- [25] A. E. Eiben and C. A. Schippers, “On evolutionary exploration and exploitation”, *Fundam. Inf.*, vol. 35, no. 1-4, pp. 35–50, Aug. 1998, ISSN: 0169-2968. [Online]. Available: <http://dl.acm.org/citation.cfm?id=297119.297124>.
- [26] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, 2nd ed., ser. Natural Computing Series. Springer Berlin, 2015, 294 pp., ISBN: 9783662448748. DOI: 10.1007/978-3-662-44874-8.
- [27] I. Fister, M. Mernik, and J. Brest, “Hybridization of evolutionary algorithms”, *CoRR*, vol. abs/1301.0929, 2013. [Online]. Available: <http://arxiv.org/abs/1301.0929>.
- [28] B. Frederiksen, “Applying expert system technology to code reuse with pyke”, *PyCon*, 2008, Conference Paper. [Online]. Available: <http://pyke.sourceforge.net/PyCon2008-paper.html> (visited on 11/10/2016).
- [29] —, *PyKE*, version 1.1.1, Chicago, IL, 2008. [Online]. Available: <http://pyke.sourceforge.net>.
- [30] “Preface to first edition”, in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds., with an intro. by F. Glover and G. A. Kochenberger, 2nd ed., ser. International Series in Operations Research & Management Science. Springer Science & Business Media, 2010, vol. 146, pp. ix–xi, ISBN: 9781441916655. DOI: 10.1007/978-1-4419-1665-5.
- [31] D. E. Goldberg, “Simple genetic algorithms and the minimal, deceptive problem”, pp. 74–88.
- [32] A. Gosavi, *Simulation-Based Optimization, Parametric Optimization Techniques and Reinforcement Learning*, 2nd ed., ser. Operations Research/Computer Science Interfaces Series. Springer Science & Business Media, 2015, vol. 55, 530 pp., ISBN: 978148974914. DOI: 10.1007/978-1-4899-7491-4.
- [33] C. Grosan and A. Abraham, *Intelligent Systems, A Modern Approach*, ser. Intelligent Systems Reference Library. Springer Berlin Heidelberg, 2011, vol. 17, 456 pp., ISBN: 9783642210044. DOI: 10.1007/978-3-642-21004-4.

-
- [34] F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, *Building Expert Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1983, ISBN: 0-201-10686-8.
- [35] M. Hepperle, *JAVAFOIL*, version 2.22, Braunschweig, Germany, Apr. 11, 2014. [Online]. Available: <http://www.mh-aerotoools.de/airfoils/javafoil.htm>, About Dr. Martin Hepperle: <http://www.mh-aerotoools.de/company/thecompany.htm>.
- [36] Y.-C. Ho and D. L. Pepyne, "Simple explanation of the no free lunch theorem of optimization", *Cybernetics and Systems Analysis*, vol. 38, pp. 292–298, 2 Mar. 2002, ISSN: 10600396. [Online]. Available: <https://search-proquest-com.etechnicryt.idm.oclc.org/docview/216507815?accountid=163027>.
- [37] J. H. Holland, *Adaptation in Natural and Artificial Systems, An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA: MIT Press, 1992, 183 pp., Originally published by University of Michigan Press, 1975, ISBN: 9780262581110.
- [38] U. Iemma, F. P. Vitagliano, and F. Centracchio, "Multi-objective design optimization of sustainable commercial aircraft, Performance and costs", *International Journal of Sustainable Engineering*, vol. 10, no. 3, pp. 147–157, 2017. DOI: 10.1080/19397038.2016.1222461.
- [39] J. Jahn, *Vector Optimization, Theory, Applications, and Extensions*, 2nd ed. Springer Berlin, 2011, 490 pp., ISBN: 9783642170058. DOI: 10.1007/978-3-642-17005-8.
- [40] B. J. Jain, H. Pohlheim, and J. Wegener, "On termination criteria of evolutionary algorithms", in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO'01, San Francisco, California: Morgan Kaufmann Publishers Inc., 2001, pp. 768–768, ISBN: 1-55860-774-9.
- [41] A. Kaveh, "Introduction", in *Advances in Metaheuristic Algorithms for Optimal Design of Structures*. Springer Switzerland, 2014, ch. 1, pp. 1–8, ISBN: 9783319055497. DOI: 10.1007/978-3-319-05549-7.
- [42] S. L. Kendal and M. Creen, *An Introduction to Knowledge Engineering*. Springer Berlin Heidelberg, 2007, 294 pp., ISBN: 9781846284755.
- [43] S. Koziel and L. Leifsson, *Simulation-Driven Design by Knowledge-Based Response Correction Techniques*. Springer International Publishing Switzerland, 2016, 266 pp., ISBN: 9783319301150. DOI: 10.1007/978-3-319-30115-0.
- [44] O. Kramer, *A Brief Introduction to Continuous Evolutionary Optimization*, ser. Springer Briefs in Applied Sciences and Technology - Studies in Computational Intelligence. Springer Science & Business Media, 2014, 100 pp., ISBN: 9783319034225. DOI: 10.1007/978-3-319-03422-5.
- [45] C. L. Ladson, C. W. Brooks Jr., A. S. Hill, and D. W. Sproles, "Computer program to obtain ordinates for naca airfoils", National Aeronautics and Space Administration (NASA), Langley Research Center, Hampton, VA, Technical Memorandum NASA TM-4741, 1996, 23 pp. [Online]. Available: <ftp://techreports.larc.nasa.gov/pub/techreports/larc/96/NASA-96-tm4741.ps.Z>.
- [46] Laminar Research, *X-Plane*, version 10.50r3, Columbia, SC, 2016. [Online]. Available: <http://www.laminarresearch.com>.
- [47] R. Landa-Becerra, L. V. Santana-Quintero, and C. A. C. Coello, "Knowledge incorporation in multi-objective evolutionary algorithms", in *Multi-Objective Evolutionary Algorithms for Knowledge Discovery from Databases*, A. Ghosh, S. Dehuri, and S. Ghosh, Eds., ser. Studies in Computational Intelligence. Berlin: Springer-Verlag, 2008, vol. 98, ch. 2, pp. 23–46, ISBN: 9783540774679.
- [48] C. Lee and E. K. Antonsson, "Variable length genomes for evolutionary algorithms", in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '00), Las Vegas, Nevada, USA, July 8-12, 2000.*, Morgan Kaufmann, 2000, p. 806, ISBN: 1-55860-708-0.
- [49] M. Li, H. Zhang, B. Tang, H. Mao, G. Peng, and J. Zhao, "A novel rapid conceptual design based on knowledge reuse and evolution", *International Journal of Computer Theory and Engineering*, vol. 6, no. 5, pp. 367–371, Oct. 2014.
- [50] H. Liu, C. Sun, Z.-D. Bai, and Q.-P. Zhao, "Automated implementation of a design principle during the optimization of conceptual aircraft", *Knowledge-Based Systems*, vol. 20, no. 3, pp. 277–282, 2007, ISSN: 0950-7051. DOI: <https://doi.org/10.1016/j.knosys.2006.04.020>.
- [51] D. Long and Z. Scott, *A Primer For Model-Based Systems Engineering*, 2nd ed. Vitech Corporation, 2011, 122 pp., ISBN: 9781105588105.
- [52] P. Lucas and L. Van Der Gaag, *Principles of Expert Systems*, ser. International Computer Science Series. Addison-Wesley, 1991, 518 pp., ISBN: 9780201416404.
-

- [53] A. B. Md. Sultan, R. Mahmud, M. N. Sulaiman, and M. R. Abu Bakar, "Maintaining diversity for genetic algorithm, A case of timetabling problem", *Journal Teknologi*, vol. 44, pp. 123–130, D 2006.
- [54] K. Miettinen, "Introduction to multiobjective optimization: Noninteractive approaches, Interactive and evolutionary approaches", in *Multiobjective Optimization*, J. Branke, K. Deb, M. Kaisa, and R. Słowiński, Eds., ser. Lecture Notes in Computer Science. Springer Berlin, 2008, vol. 5252, pp. 1–26, ISBN: 9783540889076.
- [55] "Engineering design: A systems approach", in *Handbook of Performability Engineering*, K. B. Misra, Ed. London, UK: Springer London, 2008, ch. 2, pp. 13–24, ISBN: 9781848001312. DOI: 10.1007/978-1-84800-131-2.
- [56] M. Mitchel, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1998, 209 pp., ISBN: 9780262631853.
- [57] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm – i. continuous parameter optimization", *EVOLUTIONARY COMPUTATION*, vol. 1, pp. 25–49, 1993.
- [58] G. Pahl, W. Beitz, J. Feldhusen, and K.-H. Grote, *Engineering Design, A Systematic Approach*, 3rd ed., K. Wallance and L. Blessing, Eds., trans. by K. Wallance and L. Blessing. London, UK: Springer-Verlag, 2007, 629 pp., ISBN: 9781846283192.
- [59] G.-J. Park, *Analytic Methods for Design Practice*. London, UK: Springer Science & Business Media, 2007, 636 pp., ISBN: 9781846284731.
- [60] K. V. Price, "Eliminating drift bias from the differential evolution algorithm", in *Advances in Differential Evolution*, U. K. Chakraborty, Ed., ser. Studies in Computational Intelligence. Springer Berlin, 2008, vol. 143, pp. 33–88, ISBN: 9783540688303. DOI: 10.1007/978-3-540-68830-3.
- [61] K. V. Price, M. S. Rainer, and J. A. Lampinen, *Differential Evolution*, ser. Natural Computing Series. Berlin-Heidelberg: Springer Science & Business Media, 2005, 544 pp., ISBN: 9783540209508.
- [62] G. R. Raidl, J. Puchinger, and C. Blum, "Metaheuristic hybrids", in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds., 2nd ed., ser. International Series in Operations Research & Management Science. Springer Science & Business Media, 2010, vol. 146, ch. 16, pp. 469–496, ISBN: 9781441916655. DOI: 10.1007/978-1-4419-1665-5.
- [63] D. P. Raymer, *Aircraft Design*: 2nd ed., ser. AIAA Education Series. Washington, DC: American Institute of Aeronautics and Astronautics (AIAA), 1992, 729 pp., ISBN: 9780930403515.
- [64] H. Richter and S. Yang, "Dynamic optimization using analytic and evolutionary approaches: A comparative review, From classical to modern approach", in *Handbook of Optimization*, I. Zelinka, V. Snasel, and A. Abraham, Eds., ser. Intelligent Systems Reference Library. Berlin - Heidelberg: Springer Science & Business Media, 2013, vol. 38. Classical Methods - Theory, pp. 1–28, ISBN: 9783642305047. DOI: 10.1007/978-3-642-30504-7.
- [65] F. Rothlauf, *Representations for Genetic and Evolutionary Algorithms*, 2nd ed. Springer Berlin, 2006, 334 pp., ISBN: 9783540250593.
- [66] M. Rouse, Ed. (Feb. 2015). Framework definition, [Online]. Available: <http://whatis.techtarget.com/definition/framework> (visited on 05/15/2017).
- [67] M. H. Sadraey, *Aircraft Design, A Systems Engineering Approach*, ser. Aerospace Series. Chichester, West Sussex: John Wiley & Sons, 808 pp., ISBN: 9781119953401.
- [68] B. S. Shylaja, "From navigation to star hopping: Forgotten formulae", *Resonance*, vol. 20, pp. 352–359, 4 May 3, 2015, ISSN: 0973-712X. DOI: 10.1007/s12045-015-0190-7. [Online]. Available: <http://link.springer.com/10.1007/s12045-015-0190-7>.
- [69] P. Siarry, *Metaheuristics*. Springer Switzerland, 2016, 501 pp., ISBN: 9783319454030. DOI: 10.1007/978-3-319-45403-0_1.
- [70] M. J. Simpson and J. J. Simpson, "Formal, theoretical aspects of systems engineering, Comments on "principles of complex systems for systems engineering"", *Systems Engineering*, vol. 13, pp. 109–207, 2 2010, ISSN: 1520-6858. DOI: 10.1002/sys.v13:2. [Online]. Available: <http://onlinelibrary.wiley.com/doi/10.1002/sys.20163/full>.
- [71] V. Singh, S. K. Sharma, and S. Vaibhav, "Transport aircraft conceptual design optimization using real coded genetic algorithm", *International Journal of Aerospace Engineering*, vol. 2016, 2016.
- [72] W. M. Spears, *Evolutionary Algorithms, The Role of Mutation and Recombination*, G. Rozenberg, T. Bäck, A. Eiben, J. Kok, and H. Spaink, Eds., ser. Natural Computing Series. Berlin-Heidelberg: Springer-Verlag, 2000, 222 pp., ISBN: 9783540669500. DOI: 10.1007/978-3-662-04199-4.

-
- [73] S. Sunnersjö, *Intelligent Computer Systems in Engineering Design, Principles and Applications*, ser. Studies in Systems, Decision and Control. Springer Switzerland, 2016, vol. 51, 165 pp., ISBN: 9783319281254.
- [74] G. Syswerda, “Uniform crossover in genetic algorithms”, in *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer, Ed., Morgan Kaufmann, Jun. 1989, pp. 2–9.
- [75] C. Tillier, Wikimedia Commons user. (Aug. 26, 2006). Giant planes comparison, [Online]. Available: https://commons.wikimedia.org/wiki/File%5C%3AGiant_planes_comparison.svg (visited on 05/31/2017). “Giant planes comparison” by Wikimedia Commons user Clem Tillier, used under  BY-SA-2.5 / Extracted jet silhouette from original.
- [76] G. Van Brummelen, *Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*. Princeton University Press, Jan. 11, 2013, ch. 9, pp. 159, 160, 192 pp., ISBN: 9780691148922. DOI: 10.1515/9781400844807.
- [77] D. Whitley, “A genetic algorithm tutorial”, *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994, ISSN: 1573-1375. DOI: 10.1007/BF00175354. [Online]. Available: <http://dx.doi.org/10.1007/BF00175354>.
- [78] Wikipedia. (2017). Cruise (aeronautics) — Wikipedia, the free encyclopedia, [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Cruise_\(aeronautics\)&oldid=782401274](https://en.wikipedia.org/w/index.php?title=Cruise_(aeronautics)&oldid=782401274) (visited on 05/25/2017).
- [79] —, (2017). K-means clustering — Wikipedia, the free encyclopedia, [Online]. Available: https://en.wikipedia.org/w/index.php?title=K-means_clustering&oldid=783012507 (visited on 05/25/2017).
- [80] —, (2017). Reynolds number — Wikipedia, the free encyclopedia, [Online]. Available: https://en.wikipedia.org/w/index.php?title=Reynolds_number&oldid=782435108 (visited on 05/25/2017).
- [81] —, (2017). True airspeed — Wikipedia, the free encyclopedia, [Online]. Available: https://en.wikipedia.org/w/index.php?title=True_airspeed&oldid=767471379 (visited on 05/25/2017).
- [82] —, (2017). X-plane (simulator) — Wikipedia, the free encyclopedia, [Online]. Available: [https://en.wikipedia.org/w/index.php?title=X-Plane_\(simulator\)&oldid=781668079](https://en.wikipedia.org/w/index.php?title=X-Plane_(simulator)&oldid=781668079) (visited on 04/17/2017).
- [83] C. Yang, Z. Liu, H. Wang, and J. Shen, “Reusing design knowledge based on design cases and knowledge map”, *International Journal of Technology and Design Education*, vol. 23, no. 4, pp. 1063–1077, 2013, ISSN: 1573-1804. DOI: 10.1007/s10798-013-9239-7.
- [84] D. Zahaire, “Critical values for the control parameters of differential evolution algorithms”, in *Proceedings of Mendel*, Morgan Kaufmann, 2002, pp. 62–67.
- [85] B.-T. Zhang and J.-J. Kim, “Comparison of selection methods for evolutionary optimization”, *Evolutionary Optimization: An International Journal on the Internet*, vol. 2, no. 1, pp. 55–70, 2000, ISSN: 10600396.
- [86] E. Zitzler, “Evolutionary algorithms for multiobjective optimization, Methods and applications”, PhD thesis, Swiss Federal Institute of Technology Zurich, 2008.