

FELIPE ANTONIO CHEGURY VIANA

SURROGATE MODELING TECHNIQUES AND
HEURISTIC OPTIMIZATION METHODS APPLIED TO
DESIGN AND IDENTIFICATION PROBLEMS



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
2008

FELIPE ANTONIO CHEGURY VIANA

**SURROGATE MODELING TECHNIQUES AND HEURISTIC
OPTIMIZATION METHODS APPLIED TO DESIGN AND
IDENTIFICATION PROBLEMS**

Tese apresentada ao Programa de Pós-graduação em Engenharia Mecânica da Universidade Federal de Uberlândia, como parte dos requisitos para a obtenção do título de **DOCTOR EM ENGENHARIA MECÂNICA.**

Área de Concentração: Mecânica dos Sólidos e Vibrações

Orientador: Prof. Dr. Valder Steffen Jr.

UBERLÂNDIA – MG

2008

Dados Internacionais de Catalogação na Publicação (CIP)

V614s Viana, Felipe Antonio Chegury, 1980-
Surrogate modeling techniques and heuristic optimization methods
applied to design and identification problems / Viana, Felipe Antonio
Chegury. - 2008.
156 f. : il.

Orientador: Valder Steffen Jr.

Tese (doutorado) – Universidade Federal de Uberlândia, Programa
de Pós-Graduação em Engenharia Mecânica.
Inclui bibliografia.

1. Otimização matemática - Teses. 2. Vibração - Teses. I. Steffen Jr,
Valder. II. Universidade Federal de Uberlândia. Programa de Pós-Gra-
duação em Engenharia Mecânica. IV. Título.

CDU: 621:51-7

“Nothing shocks me. I’m a scientist.”
Harrison Ford, as Indiana Jones

To my wife Nádia and my daughter Bruna.

ACKNOWLEDGMENTS

I would like to express my gratitude to my parents Celso and Valéria. The good education they gave to me, their unconditional love, appreciation, and incentive were and will always be fundamental in my life.

I thank Nádia and Bruna for dreaming my dreams with me, and for all love they have devoted to me all these years.

I am grateful to have true friends in Anthony, Rômulo and Anderson for lending me a shoulder when I had a bad day and for sharing with me the happy moments.

I also thank my academic advisor Prof. Dr. Valder Steffen for his guidance, patience, and encouragement throughout my graduate school. It has been an amazing experience to work under his supervision.

I would like to especially thank my advisory committee members, Prof. Luiz Góes, Prof. Francisco Soeiro, Prof. Domingos Rade, and Prof. Sezimária Saramago, for their willingness to serve on my committee, for evaluating my thesis, and for offering constructive criticism that has helped improved this work.

I would like to thank Dr. Garret Vanderplaats, Dr. Gerhard Venter, and Dr. Vladimir Balabanov, from Vanderplaats Research and Development Inc.; Dr. Luiz Góes, Mr. Nei Brasil, and Mr. Benedito Maciel from the Technological Institute of Aeronautics in São José dos Campos; Dr. Sérgio Butkewitsch, Dr. Marcus Leal, Mr. Marcelo Zanini and Mr. Sandro Magalhães from EMBRAER; Dr. Raphael Haftka, and Dr. Tushar Goel from the University of Florida in Gainesville; and Dr. Wei Shyy from the University of Michigan in Ann Arbor for the enriching opportunity to work with them along this doctoral research.

My special and sincere thanks to the staff and colleagues at FEMEC-UFU, for making the graduate school a great experience.

Finally, I am deeply thankful to the Brazilian research agencies CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) and CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) for the financial support during my PhD studies.

SURROGATE MODELING TECHNIQUES AND HEURISTIC OPTIMIZATION METHODS APPLIED TO DESIGN AND IDENTIFICATION PROBLEMS

TABLE OF CONTENTS

Abstract	xi
Resumo	xiii
Nomenclature	xv
 CHAPTER 1	
INTRODUCTION	1
1.1 Fundamental Concepts in Surrogate Modeling	3
1.2 Fundamental Concepts in Numerical Optimization	4
1.3 Classification of the Optimization Problems	8
1.4 Literature Review	9
1.5 Scope of Current Research	10
 CHAPTER 2	
SURROGATE MODELING AND NUMERICAL OPTIMIZATION	13
2.1 Introduction	13
2.2 Surrogate Modeling Framework	16
2.2.1 Design of Experiments (DOE) and Latin Hypercube Sampling	16
2.2.2 Surrogate Modeling Techniques	18
2.3 General Optimization Problem	23
2.4 Multi-Objective Optimization Techniques	28
2.4.1 Weighted Sum Method	28
2.4.2 Compromise Programming Method	29
2.4.2 Weighted Min-Max Method	29
2.5 Constrained Optimization Techniques	29
2.5.1 Static Penalties Method	30
2.5.2 Dynamic Penalties Method	31
 CHAPTER 3	
HEURISTIC OPTIMIZATION ALGORITHMS	33
3.1 Introduction	33
3.2 Ant Colony Optimization	34
3.3 Differential Evolution	36

3.3.1 Mutation	38
3.3.2 Selection	40
3.3.3 Remarks about DE	40
3.4 Genetic Algorithm	41
3.5 Particle Swarm Optimization	42
3.5.1 Initial Population	44
3.5.2 Algorithm Parameters	44
3.5.3 Dealing with Violated Constraints	45
3.6 LifeCycle Algorithm	46
3.7 Common Ground of the Basic Algorithms	47
3.7.1 Dispersion Measure of the Population	49
3.7.2 Discrete/Integer Variables	50
3.7.3 Additional Randomness	50
3.7.4 Stopping Criteria	51
3.8 Enhanced Stochastic Evolutionary Algorithm	52
3.8.1 Inner Loop	52
3.8.2 Outer Loop	53

CHAPTER 4

MIXING SURROGATES AND HEURISTIC OPTIMIZATION ALGORITHMS	57
4.1 Introduction	57
4.2 First Approach: Direct Use of Surrogate Models	58
4.3 Second Approach: Use of Different Levels of Fidelity	59
4.4 Third Approach: Coupling Actual High-Fidelity and Surrogate Models	61

CHAPTER 5

APPLICATIONS	65
5.1 Direct Problems	65
5.1.1 Three-dimensional Vehicular Structure Design Optimization	65
5.1.2 Optimization of Aircraft Structural Components by using Heuristic Algorithms and Multi-Fidelity Approximations	73
5.2 Inverse Problems	85
5.2.1 Aircraft Longitudinal Stability and Control Derivatives Identification by using LifeCycle and Levenberg-Marquardt Optimization Algorithms	85
5.2.2 Identification of a Non-Linear Landing Gear Model using Heuristic Optimization ..	93
5.3 Coupling Actual High-Fidelity and Surrogate Models	101

5.3.1 Coupling Heuristic Optimization and High-Fidelity Function Analysis to Improve Surrogate Models	101
5.4 Combinatorial Optimization	113
5.4.1 On How to Implement an Affordable Optimal Latin Hypercube	113
 CHAPTER 6	
SUMMARY, CONCLUSIONS AND FUTURE WORK	123
 APPENDIX A	
SIMPLE OPTIMIZATION TOOLBOX	129
A.1 What is the SIMPLE Optimization Toolbox?	129
A.2 Installation and Uninstallation	130
A.3 Help on Functions	132
A.4 Bug Reports	132
A.5 Simpletoolbox/Surrogatestoolbox User's List	132
 APPENDIX B	
SURROGATES TOOLBOX	133
B.1 What is the SURROGATES Toolbox?	133
B.2 Installation and Uninstallation	135
B.3 Help on Functions	136
B.4 Bug Reports	136
B.5 Simpletoolbox/Surrogatestoolbox User's List	137
 APPENDIX C	
BOXPLOTS	139
 APPENDIX D	
NELDER-MEAD SIMPLEX DIRECT SEARCH AND LEVENBERG-MARQUADT OPTIMIZATION METHODS	141
D.1 Nelder-Mead Simplex Direct Search	141
D.2 Levenberg-Marquadt Algorithm	141
 REFERENCES	143
 RESUMO EXTENDIDO EM PORTUGUÊS	155

VIANA, F. A. C., 2005, **Surrogate Modeling Techniques and Heuristic Optimization Methods Applied to Design and Identification Problems**. 2008. 142 pages. PhD Thesis. Universidade Federal de Uberlândia, Uberlândia, Brazil.

Abstract

Advances in computer throughput have helped to popularize numerical optimization as an engineering tool. However, they also favor an increase in complexity of the state-of-the-art simulation. As a result, the computational cost of complex high-fidelity engineering simulations often makes it difficult to rely exclusively on simulation for optimization. This doctoral research presents an effort in combining global optimization and surrogate modeling techniques as a way to rationally use the computer budget and increase the information level obtained during the optimization task. The above mentioned techniques were used in the solution of the continuous-discrete problems of the optimal design of a vehicular structure and aircraft structural components; identification of aircraft longitudinal stability and control derivatives and non-linear landing gear model and the improvement of surrogate models through extra simulations. Besides, the solution of the combinatorial problem of the optimal Latin Hypercube has been implemented. At the end of the research, the main learning is that, as it also happens with classical optimization algorithms, the success in using heuristic methods is highly dependent on a number of factors, such as the level of fidelity of the simulations, level of previous knowledge of the problem, and, of course, computational resources. This way, the use of variable fidelity and surrogate models together with heuristic optimization methods is a successful approach, since heuristic algorithms do not require gradient information (i.e., resources are directly used for the search, and there is no propagation of the errors due to the computation of the gradients); and they have the trend to find the global or near global solution. In some cases, a cascade-type combination of heuristic and classical optimization methods may be a suitable strategy for taking advantage of the global and local search capabilities of the individual algorithms.

Keywords: Heuristic optimization, surrogate modeling, design optimization, inverse problems, variable-fidelity modeling.

VIANA, F. A. C., 2005, **Técnicas de Meta-Modelagem e Métodos Heurísticos de Otimização Aplicados a Problemas de Projeto e Identificação**. 2008. 142 páginas. Tese de Doutorado. Universidade Federal de Uberlândia, Uberlândia, Brazil.

Resumo

Avanços na capacidade de processamento computacional popularizaram a otimização numérica como uma ferramenta de engenharia. Contudo, eles favoreceram também o aumento na complexidade das simulações. Como resultado, o custo computacional de simulações complexas de alta fidelidade em engenharia dificultam o uso exclusivo de simulações em otimização. Esta pesquisa de doutorado representa um esforço em combinar técnicas de otimização global e meta-modelagem como uma forma de usar racionalmente os recursos computacionais e aumentar o nível de informação obtida durante a tarefa de otimização. As técnicas mencionadas acima foram usadas na resolução dos problemas contínuo-discretos do projeto ótimo de uma estrutura veicular e componentes estruturais aeronáuticos; identificação de derivadas de controle e estabilidade longitudinal de aviões e modelo não linear de trem de pouso; e melhoramento de meta-modelos via adição de simulações. Além disso, a solução do problema combinatorial do hipercubo latino ótimo também foi implementado. Ao final da pesquisa, a principal lição é que, assim como também acontece com algoritmos clássicos de otimização, o sucesso no uso de métodos heurísticos é altamente dependente do problema, nível de fidelidade das simulações, nível das informações já conhecidas do problema, e, obviamente, recursos computacionais. Desta forma, o uso de fidelidade variável e meta-modelagem juntamente com métodos heurísticos de otimização é uma estratégia bem sucedida, uma vez que métodos heurísticos não requerem informação sobre o gradiente (isto é, os recursos são diretamente usados na busca e não há propagação dos erros devido ao cálculo dos gradientes); e eles têm a tendência em encontrar a solução global ou próxima dela. Em alguns casos, uma combinação em cascata de métodos de otimização heurísticos e clássicos pode compor uma estratégia viável para aproveitar as capacidades de busca global e local dos algoritmos individuais.

Palavras-Chave: Métodos heurísticos de otimização, meta-modelagem, otimização de projeto, problemas inversos, modelagem por fidelidade variável.

Nomenclature

Chapter 1

$f(\mathbf{x})$	objective function
$\mathbf{f}(\mathbf{x})$	vector of objective functions
$g(\mathbf{x})$	inequality constraint
$h(\mathbf{x})$	equality constraint
n_{cnstr}	total number of constraints (inequality and equality)
n_{dv}	number of design variables
$n_{ineqcnstr}$	number of inequality constraints
$\mathbf{x}$	vector of design variables
$\mathbf{x}^*$	optimal solution
x_i	i th design variable
$y(\mathbf{x})$	actual function
$\hat{y}(\mathbf{x})$	surrogate model of the actual function

Chapter 2

$\tilde{f}(\mathbf{x})$	scaled version of $f(\mathbf{x})$
$\tilde{g}(\mathbf{x})$	scaled version of $g(\mathbf{x})$
$\tilde{h}(\mathbf{x})$	scaled version of $h(\mathbf{x})$
n_{obj}	number of objective functions
p	number of points in a DOE
$BestPRESS$	surrogate with lowest $PRESS$ value for a given DOE
DOE	design of experiments
CVE	cross-validation error
$J(\mathbf{x})$	functional that combines objective and constraint functions
KRG	kriging
PRS	polynomial response surface
RBNN	radial basis neural networks
$RMSE$	root mean square error
SVR	support vector regression
$PRESS$	prediction sum of squares

Chapter 3

c_1	PSO self trust parameter
c_2	PSO population trust parameter
n_{pop}	population size
ACO	Ant Colony Optimization
CR	DE crossover probability
DE	Differential Evolution
DM	dispersion measure of the population
ESEA	Enhanced Stochastic Evolutionary Algorithm
F	DE weighting factor
LC	LyfeCycle algorithm
P	population matrix
PSO	Particle Swarm Optimization
γ	ACO dissolving rate
σ^{ACO}	aggregation of the ACO population around the current minimum

Chapter 4

$\hat{\beta}(\mathbf{x})$	ratio correction surrogate
$\hat{\delta}(\mathbf{x})$	difference correction surrogate
$y_{HF}(\mathbf{x})$	high fidelity analysis
$y_{LF}(\mathbf{x})$	low fidelity analysis
N_{max}	number of extra actual function evaluations

Chapter 5

$BestRMSE$	most accurate surrogate of the initial DOE (basis of comparison for all other surrogates)
LM	Levenberg-Marquardt algorithm
NMSDS	Nelder-Mead simplex direct search
R_a^2	PRS adjusted correlation coefficients
ϕ_p	criterion which leads to the maximization of the point-to-point distance in a DOE

CHAPTER I

INTRODUCTION

As Venkataraman and Haftka (2004) discussed, the engineering and scientific communities as consumers of computing devices have greatly benefited from the growth in the computer throughput. As a result, the design of engineering systems, ranging from simple products, such as a child's toy, to complex systems, such as an aircraft or automobiles, has become computer centric. This is seen in modern computer-based methodologies of Computer-Aided Design and Computer-Aided Engineering (CAD/CAE), instead of a time consuming process of prototype trial and error. Current software packages allow since three-dimensional drafting to numerical optimization.

However, the truth is that the increases in computer processing power, memory and storage space have alleviated but not eliminated computational cost and time constraints on the use of optimization. In the words of Venkataraman and Haftka (2004): "this is due to the constant increase in the required fidelity (and hence complexity) of analysis models". In short, analysis as well as optimization algorithms have incorporated the hardware advances by increasing the complexity, fidelity, and scope. The bottom line is that the developing in hardware throughput still feeds the advances in both mathematical formulation and numerical implementation of analysis and optimization algorithms, and there is no evidence of changes in years to come. This scenario poses a challenge to optimization, as a discipline: to remain attractive as an engineering tool, while incorporating the technological progress.

This research presents an attempt of combining modern global optimization and surrogate modeling techniques as a way to rationally use the computer budget and increase the information level obtained during the optimization task. The efforts are concentrated in the field of the heuristic optimization methods, where the potential for global optimization, easiness to code, and robustness are encouraging. More precisely, Ant Colony Optimization (ACO), Differential Evolution (DE), Genetic Algorithms (GA), Particle Swarm Optimization (PSO), LifeCycle Optimization (LC), and the Enhanced Stochastic Evolutionary Algorithm (ESEA) form the set of algorithms studied in this thesis. Surrogate modeling are then used to reduce the computational effort intrinsic to these optimization algorithms. Instead of a single and predefined surrogate, this research adopts the use of a large set of different models in order to prevent against badly fitted surrogates. Kriging models (KRG), Polynomial Response

Surfaces (PRS), Radial Basis Neural Networks (RBNN) and Support Vector Regression (SVR) are used in this research. When a large set is used, the selection of a single surrogate follows an estimator of the root mean square error ($RMSE$) based on data points, i.e. the prediction sum of squares ($PRESS$). This surrogate is then called *BestPRESS*.

The main objectives of this work are: (i) studying of the modern optimization techniques; (ii) combining surrogate modeling and heuristic algorithms; (iii) developing general-purpose optimization routines; and (iv) performing applications in direct and inverse problems of engineering. The set of applications, all real world problems, are important to suggest guidelines for the practitioner. These algorithms were employed to solve the following list of continuous/discrete optimization problems:

1. Vehicular three-dimensional structure design optimization.
2. Optimization of aircraft structural components using heuristic algorithms and multi-fidelity approximations.
3. Aircraft longitudinal stability and control derivatives identification by using LifeCycle and Levenberg-Marquardt optimization algorithms.
4. Parameter identification of a non-linear landing gear model.
5. Improvement of surrogate models using heuristic optimization algorithms and a pre-defined number of extra high-fidelity analyses.
6. Optimization of the Latin hypercube design of experiments.

As a result, the contributions of this research can be summarized as:

- The use of non-conventional methods based on heuristic optimization techniques for system identification was explored. Heuristic algorithms presented the required robustness to deal with corrupted experimental data that are typical in this type of application.
- The use of heuristic methods coupled with statistical tools for the solution of design problems was consolidated. This approach presents an alternative to overcome the limitations of classic methods and decrease the computational burden associated with heuristic optimization methods.
- The implementation of both a general-purpose optimization toolbox (SIMPLE Toolbox) and a surrogate modeling toolbox (SURROGATES Toolbox), which are tested in a number of applications.

Next sections introduce the basic concepts on optimization used in this research.

1.1. Fundamental Concepts in Surrogate Modeling

To reduce the computational time of complex engineering simulations, very often surrogate models, also known as meta-models, are employed replacing actual simulation or experimental models. Essentially, surrogate modeling consists in using statistical techniques to build approximations of expensive computer simulation codes. This way, given the vector of variables of interest, $\mathbf{x}$, if the true nature of a model is:

$$y(\mathbf{x}) = F(\mathbf{x}) , \quad (1.1)$$

then a surrogate model is:

$$\hat{y}(\mathbf{x}) = G(\mathbf{x}) , \quad (1.2)$$

and

$$y(x) = \hat{y}(x) + \text{error}(x) , \quad (1.3)$$

where $\text{error}(x)$ represents both the approximation and measurement (random) errors.

Figure 1.1 summarizes the general statistical procedure to generate and use surrogates.

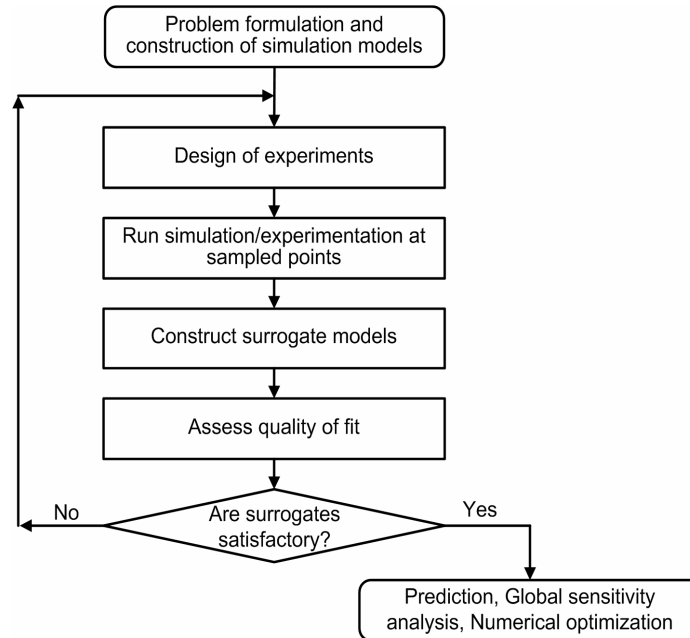


Figure 1.1. Stages of the surrogate-based modeling.

The internal steps of the loop can be detailed as:

1. Design of experiments: the design space is sampled in order to reveal its contents and tendencies (Owen, 1992; Morris and Mitchell 1995; and Montgomery, 1997). At this step, the gain of as much information as possible must be balanced with the cost of simulation/experimentation.
2. Run simulation/experimentation at sampled points: at this phase, the simulation (or experimentation) unfeasible regions of the design space can be revealed.
3. Construct surrogate models: the surrogate model is fitted to the collected data (Myers and Montgomery, 1995; Wasserman, 1993; Martin and Simpson, 2005; and Smola and Scholkopf, 2004). It may imply in the solution of a system of linear equations or even in the solution of an optimization problem.
4. Assess quality of fit: the precedent steps are sufficient to build a first tentative model, which overall quality and usefulness has to be evaluated by adequate sets of metrics (Box et al., 1978; Sacks et al., 1989; and Meckesheimer et al., 2002).

Table 1.1 shows several options for previously discussed techniques used in surrogate modeling.

Table 1.1. Techniques for surrogate modeling.

Technique	Examples
Design of experiments	Factorial (full and fractional), Central Composite Design, Box-Behnken, D-optimal, Orthogonal Array, Latin Hypercube
Surrogate model	Polynomial Response Surface, Radial Basis Neural Networks, Kriging Models, Support Vector Regression
Verification of model accuracy	Root mean square error, Maximum absolute error, Coefficient of correlation (using test points), Prediction sum of squares, Coefficient of determination (R^2), Estimated prediction variance, Estimated process variance

1.2. Fundamental Concepts in Numerical Optimization

According to Vanderplaats (2005), in Mathematics, optimization is the discipline concerned with finding the maxima and minima of functions, possibly subjected to constraints. Despite its name, optimization does not necessarily mean finding the optimum

solution to a problem; instead, it means finding the more suitable solution to a problem (which is the case of compromise solutions and robust solutions). An example of an optimization problem is the following: maximize the profit of a manufacturing operation while ensuring that none of the resources exceed certain limits and also satisfying as much of the demand faced as possible. Thus, extending the previous discussion, the optimization work is a mathematical problem compound by the following elements (Haftka and Gürdal, 1992; Marler and Arora, 2004; and Vanderplaats, 2005):

- **Design space:** where all possible solutions for a problem are considered (also known as search space). Each element of the design space is called design variable. The vector of design variables, $\mathbf{x}$, is composed by the elements x_i , $i = 1, 2, \dots, n_{dv}$, which means that the number of design variables is n_{dv} . The bounds of the design space are given by lower and upper limits for each design variable, $x_i^l \leq x_i \leq x_i^u$, $i = 1, 2, \dots, n_{dv}$. Design variables can be continuous (i.e., real values within a range), or discrete (i.e., certain values defined in a list of permissible values). When optimization is solved numerically, scaling design variables is important to avoid ill conditioning problems. This process consists in mapping the boundaries of each design variable to a new design space with boundaries $[0 \ 1]$ or $[-1 \ 1]$.
- **Objective function:** also known as cost function, fitness function or evaluating function. This is the way to evaluate each point of the design space. The objective function is denoted as $f(\mathbf{x})$ for single objective problems and $\mathbf{f}(\mathbf{x})$ for multi-objective problems.
- **Constraints:** impose restrictions to the system. A design point which satisfies all the constraints is called feasible, while a design point which violates even a single constraint is called infeasible. The collection of all feasible points is called feasible domain, or occasionally the constraint set. Figure 1.2 shows modifications in the original design space that are caused by constraints.
- **Optimization algorithms:** the algorithm that will search for the solution of the optimization problem.

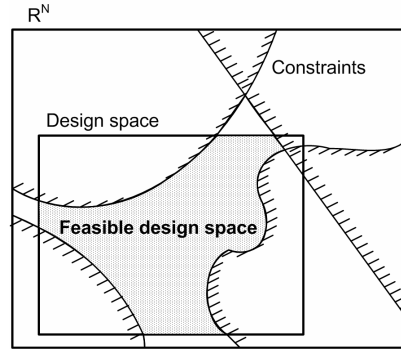


Figure 1.2. Design space definition.

In the single objective case, numerical optimization solves the following nonlinear, constrained problem: find the point, $\mathbf{x}^*$, in the design space that will minimize the objective function for a given set of system parameters, possibly observing a set of constraints. In other words, the standard formulation of an optimization problem is:

$$\text{minimize } f(\mathbf{x}), \quad (1.4)$$

subject to:

$$\begin{cases} x_i^l \leq x_i \leq x_i^u, & i = 1, 2, \dots, n_{dv}, \\ g_j(\mathbf{x}) \leq 0, & j = 1, 2, \dots, n_{ineqcnsrt}, \\ h_j(\mathbf{x}) = 0, & j = n_{ineqcnsrt} + 1, n_{ineqcnsrt} + 2, \dots, n_{cnsrt}, \end{cases} \quad (1.5)$$

where:

- $f(\mathbf{x})$ is the objective function,
- $x_i^l \leq x_i \leq x_i^u$ imposes the side constraints to the design space,
- n_{dv} is the number of design variables,
- $n_{ineqcnsrt}$ is the number of inequality constraints,
- n_{cnsrt} is the total number of constraints (inequality and equality), and
- $g_j(\mathbf{x})$ and $h_j(\mathbf{x})$ are the inequality and equality constraints, respectively.

About the optimization algorithms, they can be classified according to the level of information required to solve the problem. This way, as it can be found in Vanderplaats (2005):

- Zero-order methods: use direct evaluation of the functions (also called direct-search methods). These methods present advantages such as easiness to program, capability of dealing with non-convex and discontinuous functions, and in many cases capability of

working with discrete design variables. The price paid for this generality is that these methods often require a large number of function evaluations.

- First-order methods: require the evaluation of the gradient of the functions. Since more information is used during the search, these methods are expected to be more efficient in terms of the number of function evaluations as compared with zero-order methods. However, the price paid is that gradient information must be supplied. In addition, they have difficulties to deal with local minima and discontinuity on the first derivatives.
- Second-order methods: use function values, gradient and the Hessian matrix (the square matrix of second order partial derivatives of a function). As they increase the amount of used information, they are expected to be more efficient. However, while it does not solve the difficulties with local minima, this fact also implies in the additional hardness of the Hessian computation.

Other than the level of information, the approach used by the algorithm to manipulate this information allows a second classification. In this sense, classical methods are based on Algebra and Differential Calculus. Alternatively, in heuristic methods, the most appropriate solutions of several found are selected at successive stages for use in the next step of the search. They are called heuristic methods since the selection and update of solutions follow a heuristic rather than a formal mathematical approach (Rasmussen, 2002; and Haupt and Haupt, 2004). Figure 1.3 shows a concise representation of both classifications for the optimization algorithms.

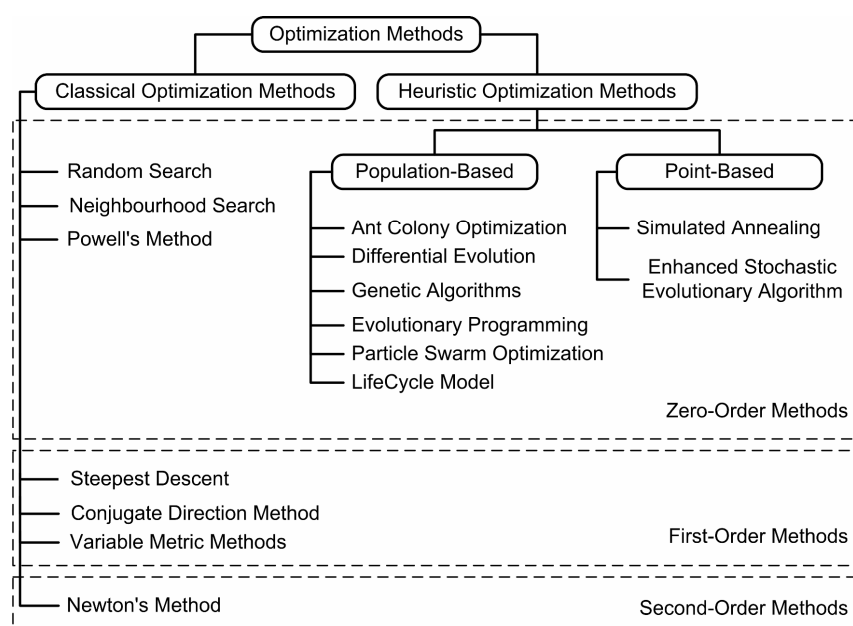


Figure 1.3. Classification of optimization algorithms.

1.3. Classification of the Optimization Problems

In terms of classification, in this research work, the optimization problems with continuous and eventually discrete-continuous variables are divided in two groups, namely direct problems and inverse problems. The distinction between them depends on the causes and effects relationship of the system. Figure 1.4 shows a scheme that illustrates the concepts in discussion. This way, in direct problems the inputs are used to determine either the best configuration of the given system or the response of the system to a pre-defined set of inputs. As an example of direct problems can be cited those in rigid body dynamics (in particular, articulated rigid body dynamics) that often require mathematical programming techniques, since rigid body dynamics attempts to solve an ordinary differential equation subjected to a set of constraints, which are various nonlinear geometric constraints such as “these two points must always coincide,” “this surface must not overlap to any other,” or “this point must always lie somewhere on this curve.” On the other hand, the inverse problem consists in using the results of actual observations to infer about either the values of the parameters characterizing the system under investigation or the inputs that generate a known set of outputs. As an example of inverse problems can be cited the identification of the proper boundary conditions and/or initial conditions such as: a) determination of thermal, stress/strain, electromagnetic, fluid flow boundary conditions on inaccessible boundaries, and b) determination of initial position, velocity, acceleration or chemical composition.

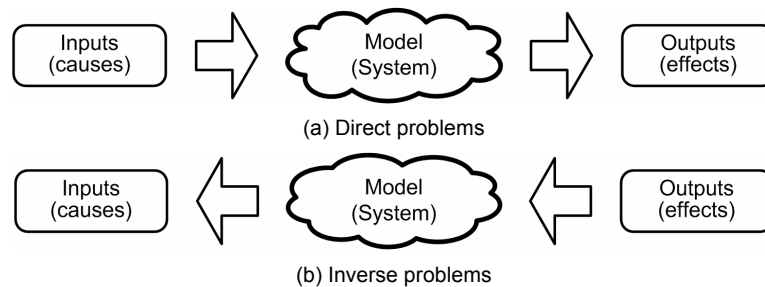


Figure 1.4. Classification of problems according to the causes and effects context

In addition, some difficulties that are intrinsic to inverse problems may arise:

- an accurate model of the system is required, since the results of the identification procedure rely upon the model used;
- identification problems are sensitive to noise found in the experimental data; and

- experimental data are incomplete either in the spatial sense (responses are available only in a limited number of positions along the structure), as in the time sense (responses are obtained in a given time interval and sampling frequency).

Alternatively, it is also possible that instead of finding the optimal set of values for different design variables, the optimization problem consists in finding the optimal combination of elements in a vector. As in the literature (Nemhauser and Wolsey, 1988; Papadimitriou and Steiglitz, 1998), here, these problems are classified as combinatorial optimization problems. In this category, instead of varying the values of each design variable, the optimizer changes the position of the design variables within the vector of design variables. The vehicle routing problem is an example of combinatorial optimization problem. Often the context is that of delivering goods located at a central depot to customers who have placed orders for such goods. The goal is minimizing the cost of distributing the goods, while serving the customers with a fleet of vehicles. Another example is the knapsack problem. It consists in the maximization problem of the best choice of essentials that can fit into one bag to be carried on a trip. Given a set of items, each with a cost and a value, the goal is to determine the number of each item to include in a collection so that the total cost is less than a given limit and the total value is as large as possible.

1.4. Literature Review

There is a vast literature about the basis of surrogate modeling and the application in benchmark and real world problems. Wang and Shan (2007) reviewed the state-of-the-art surrogate-based techniques from a practitioner's perspective according to the role of surrogate modeling in supporting design optimization, including model approximation, design space exploration, problem formulation, and solving various types of optimization problems. Simpson et al. (2001) surveyed the theoretical and practical aspects of several meta-modeling techniques, including Design of Experiments, Response Surface Methodology, Neural Networks, Inductive Learning and Kriging. Simpson et al. (2004) presented a discussion about concepts, state-of-the-art, applications and future directions of approximation methods in multidisciplinary analysis and optimization. Queipo et al. (2005) provided a discussion of the fundamental issues in surrogate-based analysis and optimization, including loss function and regularization criteria for constructing the surrogates, design of experiments, surrogate selection and construction, sensitivity analysis, convergence, and optimization. Zerpa et al. (2005) used multiple surrogates for optimization

of an alkaline–surfactant–polymer flooding processes incorporating a local weighted average model of the individual surrogates. Goel et al. (2007) explored different approaches in which the weights associated with each surrogate model are determined based on the global cross-validation error measure called prediction sum of squares.

The literature about heuristic optimization algorithms is vast as well. Coello (2005) provided a brief introduction to this class of algorithm including some of their applications and current research directions. Van Veldhuizen and Lamont (2000) and Marler and Arora (2004) presented a survey of current continuous non-linear multi-objective optimization concepts and methods. Michalewicz (1995) and Coello (2002) reviewed methods for handling constraints by heuristic methods and tested them on selected benchmark problems; and discussed their strengths and weaknesses. About the algorithms themselves, especially those embraced in this research, Ant Colony Optimization is presented in details in Dorigo et al. (1996) and Pourtakdoust and Nobahari (2004); Differential Evolution is comprehensively discussed in Storn and Price (1997) and Kaelo and Ali (2006); Genetic Algorithms are well explained in Haupt and Haupt (2004) and Michalewicz and Fogel (2000); Particle Swarm Optimization is described in Kennedy and Eberhart (1995) and Venter and Sobieszczanski-Sobieski (2003); and finally, LifeCycle Optimization is introduced in Krink and Løvberg (2002) and recently applied to a real-word problem in Flores et al. (2007). Enhanced Stochastic Evolutionary Algorithm was first introduced by Saab and Rao (1991) and later applied to the optimization of the Latin hypercube design in Jin et al. (2005) and Viana et al. (2007c).

1.5. Scope of Current Research

In short, the goal of the present work is to develop methodologies for applying heuristic optimization techniques to the solution of optimization problems in Engineering. The research, conducted in the context of a doctoral thesis, has the following main objectives:

- Study of the modern optimization techniques, embracing since the problem definition to the solution of the optimization problem.
- Combining surrogate modeling and heuristic algorithms to the solution of optimization problems.
- Developing general-purpose optimization routines for academic use enabling its application to both direct and inverse problems.
- Using the above mentioned aspects in the solution of engineering problems. The applications studied must contain both direct and inverse problems.

The organization of this work is as follows. Chapter II gives more details about surrogate modeling and numerical optimization; it includes an overview about design of experiments as well as the basic surrogate models and the ensemble of surrogates as well as a revision of the formulation of the optimization problem, including constrained and multi-objective techniques as they are used by heuristic algorithms. Chapter III discusses the optimization algorithms used in this work as well as it presents some of the contributions of the present research on the implementation of such methods. Chapter IV discusses the different proposed approaches on how to couple surrogate modeling and heuristic algorithms to optimization. Chapter V presents the applications studied during this doctoral research. Finally, Chapter VI highlights the major conclusions of the present research work and delineates the scope of future work on this topic.

CHAPTER II

SURROGATE MODELING AND NUMERICAL OPTIMIZATION

2.1. Introduction

Chapter I has already introduced the basic concepts on surrogate modeling and numerical optimization. This chapter discusses how the current doctoral research approached these techniques. This is done by presenting:

1. The interaction between the general user of numerical optimization and the set of components involved in the solution of an optimization problem.
2. The proper formulation of optimization algorithms and surrogate modeling techniques addressed by this research.

Figure 2.1 shows a possible scenario on how the user approaches the problem to be solved (at this point there is no distinction between direct, inverse or combinatorial problems). It is worth mentioning that among the different activities illustrated in Figure 2.1, the conception and decision phases (“Concept problem” and “Take decision”, respectively) are not covered by the present study. For the remaining three, the set of different exemplified software makes it easy to see the insertion in the Computer-Aided Engineering, where the computational tools are used together with real world data. In a real world context, these activities may not appear in cascade or there may be others to be considered. System Engineering (Kossiakoff and Sweet, 2002) techniques are usually employed to guide the solution of the problem, by maximizing the acquired amount of information.

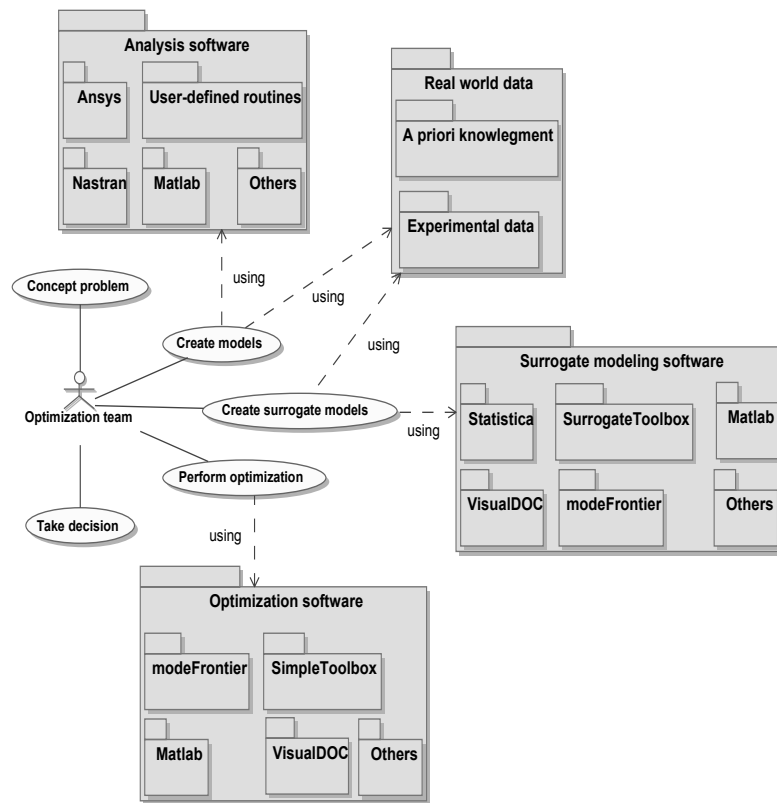


Figure 2.1. Optimization environment.

More details are given below:

- **Concept problem:** this is the process through which goals, scope and resources are defined. The focus is on the needs and required functionality, documenting and then proceeding to design synthesis and system validation while considering the complete problem. This phase often involves contribution from diverse technical disciplines (not only Engineering but also resource management). Figure 2.2-(a) illustrates a possible flowchart for this phase.
- **Create models:** this is the phase in which high-fidelity (possibly computationally or experimentally expensive) models are created. “Models” are intended to represent the physical reality of the systems under investigation. The level of accuracy in representing the reality defines the level of fidelity of the model. In the present work, to avoid misunderstandings, the expression “actual model” is sometimes used to define the high-fidelity models. Finally, the expression “expensive models” stands for difficulties in the model evaluation related to time or resource consumptions. Figure 2.2-(b) shows a possible cycle for this phase.

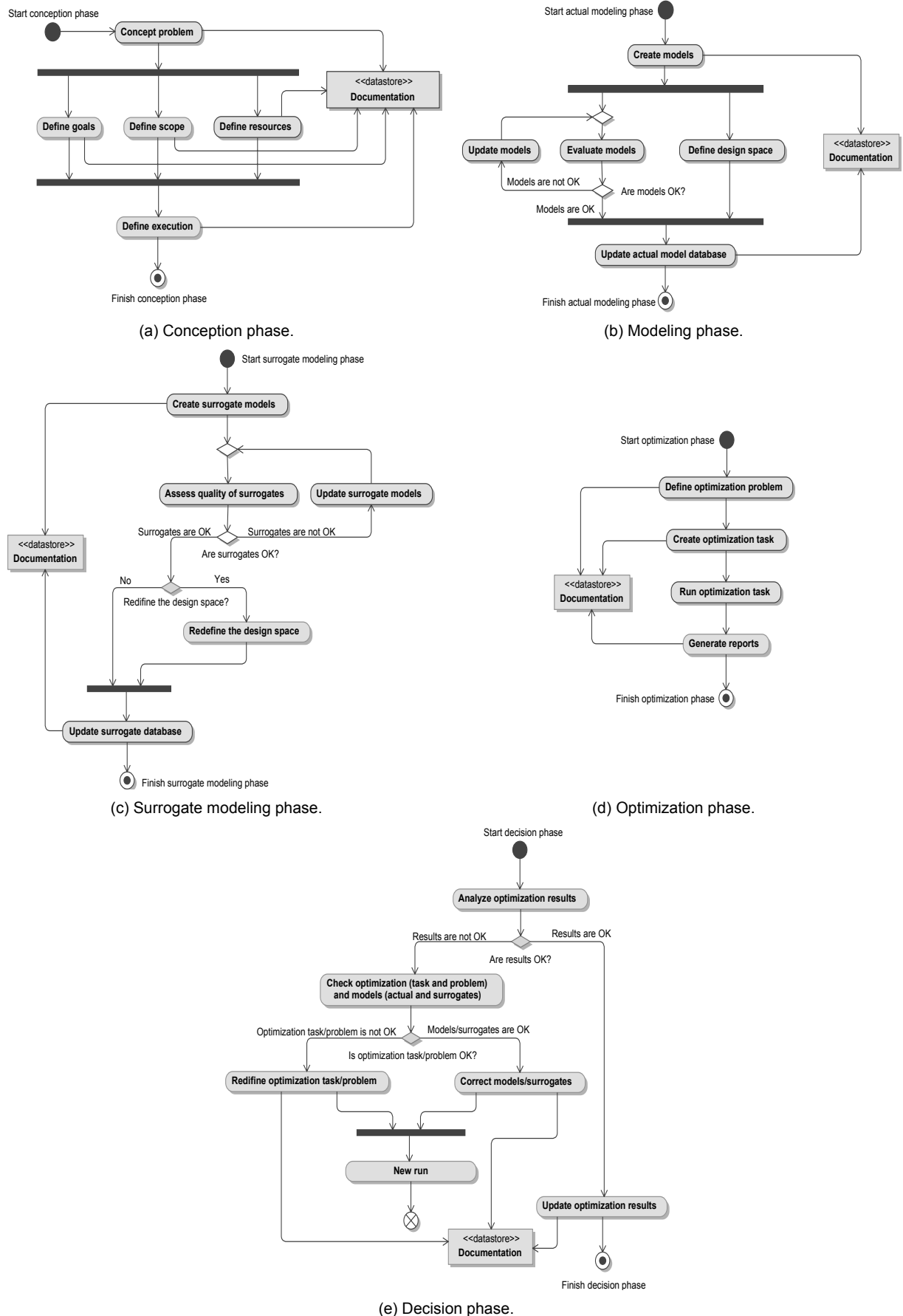


Figure 2.2. Activity diagram of different phases of the Engineering problem solution.

- Create surrogate models: when the actual models are extremely expensive to be used, surrogate models are constructed. This phase is based on statistical procedures to replace the actual models. It frequently encompasses resources for building and evaluating the fidelity level of the resulting surrogate models. Figure 2.2-(c) exemplifies the sequence of activities in this phase.
- Perform optimization: in this phase the optimization problem is defined and solved. Figure 2.2-(d) suggests a possible scheme for this phase.
- Take decision: this procedure is supported by all previous ones. In a more general sense, practical considerations (such as investment, cost-benefit ratio, marketing trends) are combined with the results of the modeling and optimization phases. Here, the focus is given to the optimization process instead, as illustrated in Figure 2.2-(e).

Despite the beauty of each of the phases, this work addresses the aspects of the use of heuristic optimization techniques on the solution of engineering problems. It means that during the conception phase, the optimization team already decided to use this class of optimization algorithms.

2.2. Surrogate Modeling Framework

2.2.1. Design of Experiments (DOE) and Latin Hypercube Sampling

The location of experimental/simulation data points is very important for generating accurate surrogate models, while maintaining a reasonable number of data points. Design of Experiments (Myers and Montgomery, 1995; and Montgomery 1997) aids in the process of point selection, extracting as much information as possible from a limited number of design points. There are many different criteria available for creating a design of experiments. The criterion of space filling design, which aims to cover as much of the design space, seems to be the most suitable for the use with heuristic optimization algorithms. Most of the time in this research, the space filling design called Latin Hypercube design of experiment was used. This sampling scheme, proposed by McKay et al. (1979) and Iman and Conover (1980), presents several advantages, such as: (i) number of points (samples) is not fixed; (ii) orthogonality of the sampling points (different points do not have the same projection in any dimension); (iii) sampling points do not depend on the surrogate model that will be constructed; and (iv) different configurations can be constructed for the same number of variables and number of sampling points.

The Latin Hypercube design is constructed in such a way that each one of the n_{dv} dimensions is divided into p equal levels and that there is only one point for each level (Bates et al., 2004). The final Latin Hypercube design then has p samples. Figure 2.3 shows two possible Latin Hypercube designs for $n_{dv} = 2$ and $p = 5$. Note that the Latin Hypercube design is constructed using a random procedure. This process results in many possible designs, each being equally good in terms of the Latin Hypercube conditions. However, a design that is ill suited for creating a surrogate model is possible, even if all the Latin Hypercube requirements are satisfied, as illustrated in Figure 2.3.

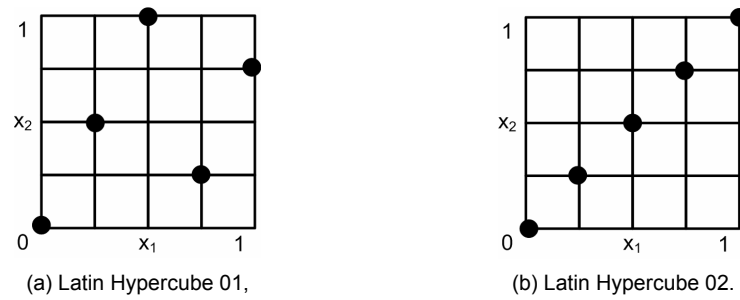


Figure 2.3. Latin Hypercube DOEs for $n_{dv} = 2$ and $p = 5$.

To overcome the above mentioned problem, the Optimal Latin Hypercube design was introduced to improve the space-filling property of the Latin Hypercube design. The Optimal Latin Hypercube design augments the Latin Hypercube design by requiring that the sample points be distributed as uniformly as possible throughout the design space. Unfortunately, the Optimal Latin Hypercube design results in a hard and time consuming optimization problem. For example, to optimize the location of 10 samples in 4 dimensions, the optimizer has to select the best design from more than 1022 possible designs. If the number of design variables is increased to 5, the number of possible designs is more than 6×10^{32} . To solve the Optimal Latin Hypercube design, it is necessary to formulate an optimization problem, the solution of which is the best design. To have an idea about how difficult this task can be, Ye et al. (2000) reported that generating an Optimal Latin Hypercube design with 25 samples in 4 dimensions using a column-wise/pair-wise algorithm could take several hours on a Sun SPARC 20 workstation. The Optimal Latin Hypercube was a case study of combinatorial optimization during this doctoral research. The achieved advances were reported in Viana et al. (2007a), where the generation of an Optimal Latin Hypercube design with 256 samples in 4 dimensions was as fast as 5 minutes in a PC with a 1000 MHz Pentium III Zeon processor. Chapter V gives more details about this implementation.

2.2.2. Surrogate Modeling Techniques

Viana et al. (2008a) have already shown that, for prediction purposes, it pays to generate a large set of surrogates and then pick the best of the set according to an estimator of the root mean square error called *PRESS* (prediction sum of squares). The main features of the four surrogate models used in this study are described in the following sections. For all discussion, consider that the models are fitted with the set of p samples.

1. Kriging (KRG)

Kriging is named after the pioneering work of the South African mining engineer D.G. Krige. It estimates the value of a function as a combination of known functions $f_i(\mathbf{x})$ (e. g., a linear model such as a polynomial trend) and departures (representing low and high frequency variation components, respectively) of the form:

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^p \beta_i f_i(\mathbf{x}) + Z(\mathbf{x}) , \quad (2.1)$$

where $Z(\mathbf{x})$ is assumed to be a realization of a stochastic process with zero mean, process variance σ^2 , and spatial covariance function given by:

$$\text{cov}(Z(\mathbf{x}_i), Z(\mathbf{x}_j)) = \sigma^2 R(\mathbf{x}_i, \mathbf{x}_j) , \quad (2.2)$$

where $R(\mathbf{x}_i, \mathbf{x}_j)$ is the correlation between $\mathbf{x}_i$ and $\mathbf{x}_j$.

The conventional KRG models interpolate training data. This is an important characteristic when dealing with noisy data. In addition, KRG is a flexible technique since different instances can be created, for example, by choosing different pairs of $f_i(\mathbf{x})$ and correlation functions. The complexity of the method and the lack of commercial software may hinder this technique from being popular in the near term (Simpson et al., 1997).

The Matlab code developed by Lophaven et. al (2002) was used to execute the KRG algorithm. More details about KRG are provided in Sacks et al. (1989), Simpson et al. (1997), and Martin and Simpson (2005).

2. Polynomial Response Surface (PRS)

The PRS approximation is one of the most well established meta-modeling techniques. In PRS modeling, a polynomial function is used to approximate the actual function. A second-order polynomial model can be expressed as:

$$\hat{y}(\mathbf{x}) = \beta_0 + \sum_{i=1}^m \beta_i x_i + \sum_{i=1}^m \sum_{j=1}^m \beta_{ij} x_i x_j , \quad (2.3)$$

The set of coefficients β can be obtained by least squares and according to the PRS theory are unbiased and have minimum variance. Another characteristic is that it is possible to identify the significance of different design factors directly from the coefficients in the normalized regression model (in practice, using t-statistics). In spite of the advantages, there is a drawback when applying PRS to model highly nonlinear functions. Even though higher-order polynomials can be used, it may be too difficult to take sufficient sample data to estimate all of the coefficients in the polynomial equation, particularly in large dimensions.

The SURROGATES toolbox of Viana and Goel (2008) was used for PRS modeling. See Box et al. (1978) and Myers and Montgomery (1995) for more details about PRS.

3. Radial Basis Neural Networks (RBNN)

RBNN is an artificial neural network which uses radial basis functions as transfer functions. RBNN consist of two layers: a hidden radial basis layer and an output linear layer, as shown in Figure 2.4.

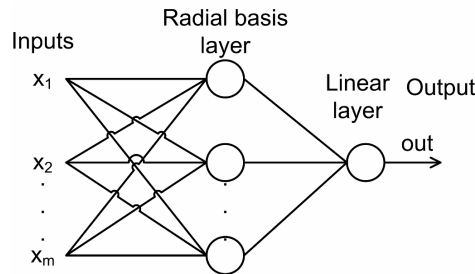


Figure 2.4. Radial basis neural network architecture.

The output of the network is thus:

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^N a_i \rho(\mathbf{x}, \mathbf{c}_i) , \quad (2.4)$$

where N is the number of neurons in the hidden layer, $\mathbf{c}_i$ is the center vector for neuron i , and a_i are the weights of the linear output neuron. The norm is typically taken to be the Euclidean distance and the basis function is taken to be the following:

$$\rho(\mathbf{x}, \mathbf{c}_i) = \exp(-\beta \|\mathbf{x} - \mathbf{c}_i\|^2) , \quad (2.5)$$

where $\exp(\cdot)$ is the exponential function.

RBNN may require more neurons than standard feed-forward/back-propagation networks, but often they can be designed in a fraction of the time it takes to train standard feed-forward networks. They work best when many training points are available, and this can be a crucial drawback in some applications. Another important point to consider is that the training process may lead to totally different configurations and thus different models.

The native neural networks Matlab[®] toolbox (MathWorks Contributors, 2002) was used to execute the RBNN algorithm. RBNN is comprehensively presented in Smith (1993), Wasserman (1993), and Cheng and Titterington (1994).

4. Support Vector Regression (SVR)

SVR is a particular implementation of support vector machines (SVM). In its present form, SVM was developed at AT&T Bell Laboratories by Vapnik and co-workers in the early 1990s (Vapnik, 1995). In SVR, the aim is to find $\hat{y}(\mathbf{x})$ that has at most ε deviations from each of the targets of the training inputs. Mathematically, the SVR model is given by:

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^p (a_i - a_i^*) K(\mathbf{x}_i, \mathbf{x}) + b \quad (2.6)$$

where $K(\mathbf{x}_i, \mathbf{x})$ is the so-called kernel function, $\mathbf{x}_i$ are different points of the original DOE and $\mathbf{x}$ is the point of the design space in which the surrogate is evaluated. Parameters a_i , a_i^* , and b are obtained during the fitting process.

Table 2.1 lists the kernel functions used in this work.

During the fitting process, SVR minimizes an upper bound of the expected risk unlike empirical risk minimization techniques, which minimize the error in the training data (which defines a_i , a_i^* , and b in Eq. (2.6)). This is done by using alternative loss functions. Figure 2.5 shows two of the most common possible loss functions. Figure 2.5-(a) corresponds to the conventional least squares error criterion. Figure 2.5-(b) illustrates the loss function used in this work, which is given by:

$$Loss(\mathbf{x}) = \begin{cases} \varepsilon, & \text{if } |y(\mathbf{x}) - \hat{y}(\mathbf{x})| \leq \varepsilon \\ |y(\mathbf{x}) - \hat{y}(\mathbf{x})|, & \text{otherwise} \end{cases} \quad (2.7)$$

Table 2.1. Kernel functions (consider $\exp(\cdot)$ as the exponential function).

Gaussian Radial Basis Function (GRBF)	$K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\frac{\ \mathbf{x}_1 - \mathbf{x}_2\ ^2}{2\sigma^2}\right)$
Exponential Radial Basis Function (ERBF)	$K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\frac{\ \mathbf{x}_1 - \mathbf{x}_2\ }{2\sigma^2}\right)$
Splines	$K(\mathbf{x}_1, \mathbf{x}_2) = 1 + \langle \mathbf{x}_1, \mathbf{x}_2 \rangle + \frac{1}{2} \langle \mathbf{x}_1, \mathbf{x}_2 \rangle \min(\mathbf{x}_1, \mathbf{x}_2) - \frac{1}{6} (\min(\mathbf{x}_1, \mathbf{x}_2))^3$
Anova-Spline (Anova)	$K(\mathbf{x}_1, \mathbf{x}_2) = \prod_i K_i(x_{1i}, x_{2i})$ $K_i(x_{1i}, x_{2i}) = 1 + x_{1i}x_{2i} + (x_{1i}x_{2i})^2 + (x_{1i}x_{2i})^2 \min(x_{1i}, x_{2i}) - x_{1i}x_{2i}(x_{1i} + x_{2i})(\min(x_{1i}, x_{2i}))^2 + \frac{1}{3}(x_{1i}^2 + 4x_{1i}x_{2i} + x_{2i}^2)(\min(x_{1i}, x_{2i}))^3 + -\frac{1}{2}(x_{1i} + x_{2i})(\min(x_{1i}, x_{2i}))^4 + \frac{1}{5}(\min(x_{1i}, x_{2i}))^5$

The implication is that in SVR the goal is to find a function that has at most ε deviation from the training data. In other words, the errors are considered zero as long as they are inferior to ε .

According to Smola and Scholkopf (2004): “after that the algorithmic development seems to have found a more stable stage, one of the most important ones seems to be to find tight error bounds derived from the specific properties of kernel functions.” Another open issue in SVR is the choice of the values of parameters for both kernel and loss functions.

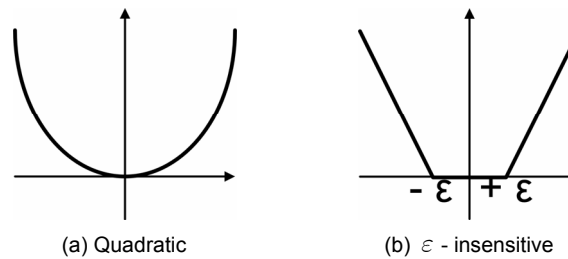


Figure 2.5. Loss functions.

The Matlab code developed by Gunn (1998) was used to execute the SVR algorithm. To learn more about SVR see Vapnik (1995), Smola and Scholkopf (2004), and Clarke et al. (2005).

5. Using a Large Set of Surrogates

Before start the discussion about the use of an ensemble of surrogates, it is important to clarify that when a set of surrogates is generated a common measure of quality must be employed in order to rank the surrogates. Since the set may be constituted by surrogates based on different statistical assumptions, this measure must be model-independent. One commonly used measure is the root mean square error ($RMSE$), which in the design domain with volume V is given by:

$$RMSE = \sqrt{\frac{1}{V} \int_V [y(\mathbf{x}) - \hat{y}(\mathbf{x})]^2 d\mathbf{x}}, \quad (2.8)$$

$RMSE$ is computed by Monte-Carlo integration at a large number of p_{test} test points:

$$RMSE = \sqrt{\frac{1}{p_{test}} \sum_{i=1}^{p_{test}} (y_i - \hat{y}_i)^2} \quad (2.9)$$

However, the use of an extra large set of test points is frequently prohibitive in most of the real world applications. This way, $RMSE$ is estimated by using cross-validation errors. A cross-validation error is the error at a data point when the surrogate is fitted to a subset of the data points not including that point. When the surrogate is fitted to all the other $p - 1$ points, (so-called leave-one-out strategy), it is obtained the vector of cross-validation errors, $\tilde{\epsilon}$. This vector is also known as the $PRESS$ vector ($PRESS$ stands for prediction sum of squares). Figure 2.6 illustrates the cross-validation errors at the third point of the DOE by fitting a PRS and a KRG model to the remaining four of the five data points of the function $\sin(x)$.

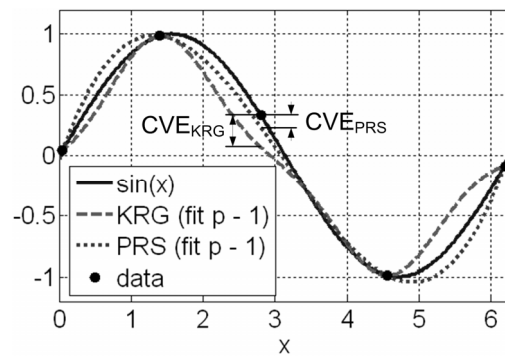


Figure 2.6. Cross-validation errors (CVE) for a polynomial response surface and Kriging surrogates.

However, the leave-one-out strategy is computationally expensive for large number of points. We then use a variation of the k -fold strategy (Kohavi, 1995). According to the classical k -fold strategy, after dividing the available data (p points) into p/k clusters, each fold is constructed using a point randomly selected (without replacement) from each of the clusters. Of the k folds, a single fold is retained as the validation data for testing the model, and the remaining $k - 1$ folds are used as training data. The cross-validation process is then repeated k times with each of the k folds used exactly once as validation data. Note that k -fold turns out to be the leave-one-out when $k = p$.

The $RMSE$ is estimated from the $PRESS$ vector:

$$PRESS = \sum_{i=1}^p \tilde{e}_i^2, \quad PRESS_{RMS} = \sqrt{\frac{1}{p} PRESS}, \quad (2.10)$$

where $\tilde{e}_i$ is the cross-validation error obtained at the i -th point of the DOE.

As shown in Viana et al. (2008b), since $PRESS_{RMS}$ is a good estimator of the $RMSE$, one possible way of using multiple surrogates is to select the model with best $PRESS$ value (called in the literature as *BestPRESS* surrogate). Because the quality of fit depends on the data points the *BestPRESS* surrogate may vary from DOE to DOE. This strategy may include surrogates based on the same methodology, such as different instances of Kriging (e.g., Kriging models with different regression and/or correlation functions). The main benefit from a diverse and large set is the increasing chance of avoiding (i) poorly fitted surrogates and (ii) DOE dependence of the performance of individual surrogates. Obviously, the key for the success when using *BestPRESS* is the quality of the $PRESS$ estimator.

The SURROGATES toolbox of Viana and Goel (2008) is used for an easy manipulation of all different codes previously presented.

2.3. General Optimization Problem

The basic version of the heuristic optimization algorithms is defined for unconstrained problems with a single objective. Since most engineering problems are multi-objective and/or constrained problems in one way or another, it is important to add the capability of dealing with constraint functions in a multi-criteria context. For the sake of clarity, the optimization problem is restated in its general form as:

$$\text{minimize } \mathbf{f}(\mathbf{x}), \quad (2.11)$$

subject to:

$$\begin{cases} x_i^l \leq x_i \leq x_i^u, & i = 1, 2, \dots, n_{dv}, \\ g_j(\mathbf{x}) \leq 0, & j = 1, 2, \dots, n_{ineqcnsrt}, \\ h_j(\mathbf{x}) = 0, & j = n_{ineqcnsrt} + 1, n_{ineqcnsrt} + 2, \dots, n_{cnsrt}, \end{cases} \quad (2.12)$$

where:

- $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}) \ f_2(\mathbf{x}) \ \dots \ f_{n_{obj}}(\mathbf{x})]^T$ is the vector of objective functions. This vector is composed by objective functions that can sometimes support but more commonly conflict with each other.
- $x_i^l \leq x_i \leq x_i^u$ imposes the side constraints to the design space,
- n_{dv} is the number of design variables,
- $n_{ineqcnsrt}$ is the number of inequality constraints,
- n_{cnsrt} is the total number of constraints (inequality and equality), and
- $g_j(\mathbf{x})$ and $h_j(\mathbf{x})$ are the inequality and equality constraints, respectively.

Figure 2.7 gives a graphical representation of the general optimization problem in terms of design and function space.

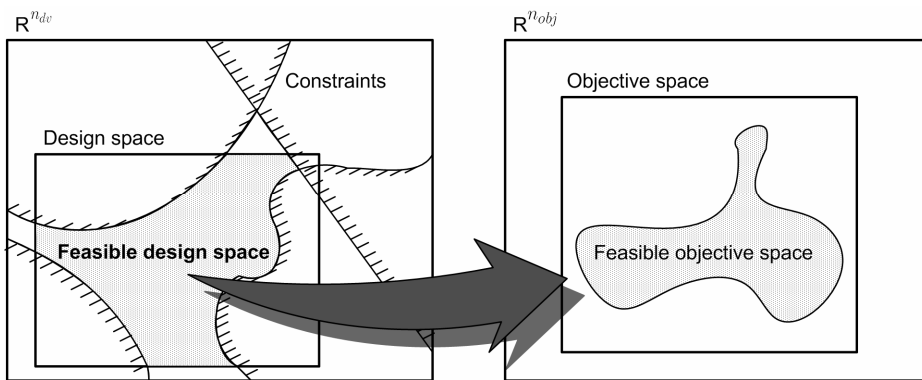


Figure 2.7. General optimization problem.

The addition of multiple objectives asks for the introduction of new concepts. While the solution of a single objective problem is a single design point, the solution of a multi-objective problem is a subspace of designs. The subspace is characterized by the condition that no objective function can be improved without some deterioration in another objective function.

This is known as “Pareto optimally” (Haftka and Gürdal, 1992; Marler and Arora, 2004). Formally, the Pareto optimal can be defined as follows (Marler and Arora, 2004):

- A point, $\mathbf{x}^* \in FeasibleDesignSpace$, is a Pareto optimal if there does not exist another point, $\mathbf{x} \in FeasibleDesignSpace$, such that $f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*)$ for all $i = 1, 2, \dots, n_{obj}$, and $f_i(\mathbf{x}) < f_i(\mathbf{x}^*)$ for at least one objective function.

The set of Pareto optimal points creates what is typically referred to as the Pareto front. Another important concept in multi-objective optimization is the concept of utopia point (Marler and Arora, 2004):

- A point, $\mathbf{f}^\circ(\mathbf{x}) \in FeasibleObjectiveSpace$, is an utopia point (or ideal point) if for each $i = 1, 2, \dots, n_{obj}$, $f_i^\circ(\mathbf{x}) = \min(f_i(\mathbf{x}))$.

In general, the utopia point is unattainable, and the best solution is as close as possible to the utopia point. Such a solution is called a compromise solution and is Pareto optimal. Figure 2.8 illustrates these concepts by considering the example of two conflicting objectives. The shaded area defines non-optimal solutions.

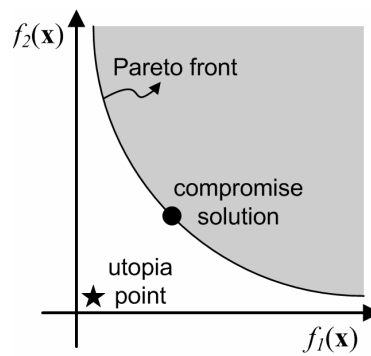


Figure 2.8. Pareto front.

As in the case of design variables, performing a mapping of the functions to a common range may be convenient. By doing this, they will not be erroneously interpreted when combined together by the handling techniques. There are different methods of scalarization, depending on whether a function is an objective or a constraint.

Considering the objective functions, the first approach is as simple as:

$$\tilde{f}_i(\mathbf{x}) = \frac{f_i(\mathbf{x})}{|f_i^{\max}(\mathbf{x})|}, \quad (2.13)$$

where $\tilde{f}_i(\mathbf{x})$ is the scaled version of $f_i(\mathbf{x})$ and $f_i^{\max}(\mathbf{x})$ is the maximum value possible for $f_i(\mathbf{x})$. Note that $f_i^{\max}(\mathbf{x}) \neq 0$ is assumed. The result, $\tilde{f}_i(\mathbf{x})$, is a non-dimensional objective function with an upper limit of one (or negative one) and an unbounded lower limit.

An alternative to Eq. (2.13) is:

$$\tilde{f}_i(\mathbf{x}) = \frac{f_i(\mathbf{x}) - f_i^{\circ}(\mathbf{x})}{f_i^{\circ}(\mathbf{x})}. \quad (2.14)$$

In this case, the lower value of $\tilde{f}_i(\mathbf{x})$ is restricted to zero, while the upper value is unbounded.

However, a most robust approach is given as follows:

$$\tilde{f}_i(\mathbf{x}) = \frac{f_i(\mathbf{x}) - f_i^{\circ}(\mathbf{x})}{f_i^{\max}(\mathbf{x}) - f_i^{\circ}(\mathbf{x})}. \quad (2.15)$$

In this case, the values assumed by $\tilde{f}_i(\mathbf{x})$ are between zero and one.

It is easy to see that the mapping as suggested by Eqs. (2.13) to (2.15), relies on $f_i^{\max}(\mathbf{x})$ and $f_i^{\circ}(\mathbf{x})$. Unfortunately, this information is not available. Indeed, there might be no reason for optimization if $f_i^{\circ}(\mathbf{x})$ was known. In practice, these values are either estimated or assumed as based on any a priori knowledge regarding the problem. Vanderplaats (2005) suggests $f_i^{\text{worst}}(\mathbf{x})$ as the objective function associated with the initial design (also called baseline design) to replace $f_i^{\max}(\mathbf{x})$; and $f_i^*(\mathbf{x})$ as the desired value of this objective function to be used in the place of $f_i^{\circ}(\mathbf{x})$. The counterparts on the case of heuristic optimization algorithms are the worst value of the objective function $f_i(\mathbf{x})$ found in the initial population as suggestion for $f_i^{\text{worst}}(\mathbf{x})$, and in the case of lacking an initial guess for $f_i^*(\mathbf{x})$, the best value found in the initial population for $f_i(\mathbf{x})$ could be used as $f_i^*(\mathbf{x})$.

As in the case of objective functions, inequality constraint functions can also lie in different ranges. Then, when treated by penalty methods (as explained later), that can be misinterpreted as different level of violation instead of different ranges. In that case, it is desired that the scaled versions of the constraints have the same range (or order of magnitude) while restoring the formulation as presented in Eq. (2.12). Thus, assuming that an inequality constraint is as $g_j(\mathbf{x}) \leq g_j^*$, there are just two possibilities for the target value, g_j^* : $g_j^* = 0$, or $g_j^* \neq 0$. For the first case, i.e. $g_j^* = 0$, the scaled version is given by:

$$\tilde{g}_j(\mathbf{x}) = \frac{g_j(\mathbf{x})}{g_j^\bullet} \leq 0, \quad (2.16)$$

where $\tilde{g}_j(\mathbf{x})$ is the scaled version of $g_j(\mathbf{x})$, and $g_j^\bullet$ is its order of magnitude.

For the second case, i.e. $g_j^* \neq 0$, $\tilde{g}_j(\mathbf{x})$ is computed using:

$$\tilde{g}_j(\mathbf{x}) = \frac{g_j(\mathbf{x})}{g_j^*} - 1 \leq 0. \quad (2.17)$$

On other circumstances, the optimum must satisfy $g_j(\mathbf{x}) \geq g_j^*$, and as before, the scaled version depends on g_j^* . For $g_j^* = 0$, the scaled version is given by changing the inequality on Eq. (2.16). For $g_j^* \neq 0$, the scaled version is as:

$$\tilde{g}_j(\mathbf{x}) = 1 - \frac{g_j(\mathbf{x})}{g_j^*} \leq 0. \quad (2.18)$$

Finally, for the same reasons, re-writing the set of equality constraints may be convenient. If the target value of $h_j(\mathbf{x})$ is $h_j^* = 0$, the scaled version, $\tilde{h}_j(\mathbf{x})$, is as:

$$\tilde{h}_j(\mathbf{x}) = \frac{h_j(\mathbf{x})}{h_j^\bullet} = 0, \quad (2.19)$$

where $h_j^\bullet$ is the order of magnitude of $h_j(\mathbf{x})$.

If on the other hand, $h_j^* \neq 0$, then $\tilde{h}_j(\mathbf{x})$ is given by:

$$\tilde{h}_j(\mathbf{x}) = \frac{h_j(\mathbf{x})}{h_j^*} - 1 = 0. \quad (2.20)$$

Now that all functions considered in the optimization problem have been properly scaled, the general optimization problem can be re-stated to be solved by using heuristic optimization algorithms. Equations (2.11) and (2.12) are combined in a functional, $J(\mathbf{x})$, to be minimized. This functional is obtained as follows (Marler and Arora, 2004):

$$J(\mathbf{x}) = F(\tilde{\mathbf{f}}(\mathbf{x})) + P(\tilde{\mathbf{g}}(\mathbf{x}), \tilde{\mathbf{h}}(\mathbf{x})), \quad (2.21)$$

where $F(\tilde{\mathbf{f}}(\mathbf{x}))$ plays the role of the multi-objective handling technique that combines the vector of scaled objective functions, $\tilde{\mathbf{f}}(\mathbf{x})$, and $P(\tilde{\mathbf{g}}(\mathbf{x}), \tilde{\mathbf{h}}(\mathbf{x}))$ plays the role of the constrained optimization handling technique that combines the vectors of scaled inequality constraints, $\tilde{\mathbf{g}}(\mathbf{x})$, and equality constraints, $\tilde{\mathbf{h}}(\mathbf{x})$. $P(\tilde{\mathbf{g}}(\mathbf{x}), \tilde{\mathbf{h}}(\mathbf{x}))$ is zero, if no violation occurs, and positive otherwise (actually, the worst the violation the larger the value of $P(\tilde{\mathbf{g}}(\mathbf{x}), \tilde{\mathbf{h}}(\mathbf{x}))$).

2.4. Multi-Objective Optimization Techniques

When considering multi-objective problems, the algorithms offer alternatives either to compute the compromise solution (Van Veldhuizen and Lamont 2000; and Marler and Arora, 2004) or to build the Pareto front (Zitzler and Thiele, 1999; and Deb, 2001). In this work, the focus is on algorithms used to obtain the compromise solution.

Basically, there are two main strategies used for these techniques. The first consists in building a functional through the manipulation of the objective functions. The second one redefines the multi-objective problem as a constrained optimization problem. For the sake of simplicity, in this work only the most common methods following the first approach are covered.

2.4.1. Weighted Sum Method

This is the most common approach used in multi-objective optimization. The functional $F(\tilde{\mathbf{f}}(\mathbf{x}))$ is given by:

$$F(\tilde{\mathbf{f}}(\mathbf{x})) = \sum_{i=1}^{n_{obj}} w_i \tilde{f}_i(\mathbf{x}), \quad (2.22)$$

where $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_{n_{obj}}]^T$ is the vector of weighting factors, which indicate the importance of each objective, such as $\sum_{i=1}^{n_{obj}} w_i = 1$ and $w_i > 0$.

The results of the optimization problem formulated by Eq.(2.22) can vary significantly according to the choice of the weights. Consequently, it is possible to generate the Pareto front by systematically changing the values of the weights (Coello, 1999; and Gerace, 2007).

2.4.2. Compromise Programming Method

While the Weighted Sum Method has the strength of being able to generate the Pareto frontier, it is sensitive to the values of the weights if used to obtain the compromise solution. The Compromise Programming Method represents a variation of Eq.(2.22) pursuing more robustness in generating the compromise solution. The functional $F(\tilde{\mathbf{f}}(\mathbf{x}))$ is obtained by:

$$F(\tilde{\mathbf{f}}(\mathbf{x})) = \left[\sum_{i=1}^{n_{obj}} (w_i \tilde{f}_i(\mathbf{x}))^p \right]^{\frac{1}{p}} . \quad (2.23)$$

Other than the vector $\mathbf{w}$, the solution also depends on the value of the parameter p (generally $p = 2$). Since $\tilde{f}_i(\mathbf{x})$ is between zero and one (or at least $\tilde{f}_i(\mathbf{x}) \geq 0$), p is proportional to the amount of emphasis placed on minimizing the function with the largest difference between $f_i(\mathbf{x})$ and $f_i^{\circ}(\mathbf{x})$ (or, more practically, $f_i^*(\mathbf{x})$).

2.4.3. Weighted Min-Max Method

The Weighted Min-Max Method is presented as follows:

$$F(\tilde{\mathbf{f}}(\mathbf{x})) = \max_i (w_i \tilde{f}_i(\mathbf{x})) . \quad (2.24)$$

This is equivalent to the limit of Eq. (2.23) as $p \rightarrow \infty$. Therefore, Eq. (2.24) can provide the complete Pareto optimal set by varying the values of the weights as well (Marler and Arora, 2004).

2.5. Constrained Optimization Techniques

There exist several methods proposed for handling constraints by heuristic algorithms for numerical optimization problems (Michalewicz, 1995; and Coello, 2002). In this research, the focus is on those based on the concept of penalty functions.

Before calculating $P(\tilde{\mathbf{g}}(\mathbf{x}), \tilde{\mathbf{h}}(\mathbf{x}))$, it is necessary to pre-handle the set of constraints. The set of $\tilde{\mathbf{g}}(\mathbf{x})$ and $\tilde{\mathbf{h}}(\mathbf{x})$ functions are grouped to form a new set of functions, $c_j(\mathbf{x})$, that represents all the constraints. One of the possible options is given below:

$$c_j(\tilde{\mathbf{g}}(\mathbf{x}), \tilde{\mathbf{h}}(\mathbf{x})) = \begin{cases} \max(\tilde{g}_j(\mathbf{x}), 0), & j = 1, 2, \dots, n_{ineqcnstr} , \\ |\tilde{h}_j(\mathbf{x})|, & j = m + 1, m + 2, \dots, n_{cnstr} . \end{cases} \quad (2.25)$$

Equation (2.25) presents a proper treatment for the inequality constraints, i.e., $c_j(\mathbf{x})$ is zero if there is no violation, and $\tilde{g}_j(\mathbf{x})$ otherwise. However, if the equality constraints follow rigorously the definition given by Eq. (2.12), it does not present a practical solution for the heuristic algorithms. The formulation as in $|\tilde{h}_j(\mathbf{x})|$ has the inconvenience of making infeasible regions of the design space close to $\tilde{h}_j(\mathbf{x}) = 0$. Then, the solution is transforming an equality function, $\tilde{h}_j(\mathbf{x})$, to an inequality one, $\tilde{g}_j(\mathbf{x})$, of the form:

$$\tilde{g}_j(\mathbf{x}) = |\tilde{h}_j(\mathbf{x})| - \varepsilon \leq 0, \quad (2.26)$$

where ε is a tolerance to the violation, usually a “small” value.

Finally, with the redefinition of the equality constraints as inequality constraints as given by Eq. (2.26), Eq. (2.25) is re-written as:

$$c_j(\tilde{\mathbf{g}}(\mathbf{x})) = \max(\tilde{g}_j(\mathbf{x}), 0), \quad j = 1, 2, \dots, n_{cnstr}, \quad (2.27)$$

and the functional for the constraints $P(\tilde{\mathbf{g}}(\mathbf{x}), \tilde{\mathbf{h}}(\mathbf{x}))$ turns into $P(\tilde{\mathbf{g}}(\mathbf{x}))$.

2.5.1. Static Penalties Method

In this method, the functional $P(\tilde{\mathbf{g}}(\mathbf{x}))$ is given by:

$$P(\tilde{\mathbf{g}}(\mathbf{x})) = \sum_{i=1}^{n_{obj}} r_i [c_i(\tilde{\mathbf{g}}(\mathbf{x}))]^\beta \quad (2.28)$$

where β is a constant (usually $\beta = 2$) and $\mathbf{r} = [r_1 \ r_2 \ \dots \ r_{n_{cnstr}}]^T$ is the vector of weighting factors used to express the penalty degree of each constraint.

Differently from what happens in multi-objective problems, for which the weights lead to different solutions on the Pareto front, here the weights may be used to distinguish the importance of individual constraints. While this can change the level of violation, it may not change the solution of the problem (i.e. the set of optimum design variables and the value of the objective functions). As a consequence, using the same value, r , for all elements of the vector $\mathbf{r}$ is common. In this case, Eq. (2.28) is re-defined as:

$$P(\tilde{\mathbf{g}}(\mathbf{x})) = r \sum_{i=1}^{n_{obj}} [c_i(\tilde{\mathbf{g}}(\mathbf{x}))]^\beta \quad (2.29)$$

2.5.2. Dynamic Penalties Method

The second and last method used in this research imposes dynamic penalties. The idea is to make the penalization harder along the evolution of the optimization process. At the beginning of the process there is little or no knowlegment about the problem. At this point, penalization could inhibit the proper exploration of the design space. As the optimization evolves, it is expected that infeasible regions get harder penalties. In practice, the functional $P(\tilde{\mathbf{g}}(\mathbf{x}))$, at the iteration t , is given by:

$$P(\tilde{\mathbf{g}}(\mathbf{x})) = (C \times t)^\alpha \sum_{i=1}^{n_{obj}} [c_i(\tilde{\mathbf{g}}(\mathbf{x}))]^\beta, \quad (2.30)$$

where C , α , and β are constants. As suggested in Michalewicz (1995), a reasonable choice for these parameters is $C = 0.5$ and $\alpha = \beta = 2$.

When using Eqs. (2.28), (2.29) or (2.30), heuristic optimization methods do not have any extra information (such as Lagrange multipliers). Then, these equations define the level of violation, if any, of each individual of the population. As a consequence, the penalty parameters ($\mathbf{r}$, r , or C) may assume large values in order to better differentiate the feasible region of the design space.

This chapter reviewed concepts about surrogate modeling and the optimization problem formulation as used by heuristic algorithms. The algorithms themselves are presented in the next chapter.

CHAPTER III

HEURISTIC OPTIMIZATION ALGORITHMS

3.1. Introduction

Chapter II showed the set of activities performed during the solution of an optimization problem as well as general concepts on surrogate modeling and formulation of the optimization problem as treated by heuristic optimization algorithms. This chapter presents the basic schemes of the algorithms and theoretical aspects of the contributions proposed during this doctoral research.

Even though classical methods (where the search process is guided by information regarding the gradient) have been widely used due to their computational efficiency, they have difficulties when a local minimum is found (often erroneously interpreted as the global one). On the other hand, Venter and Sobieszczanski-Sobieski (2003) discussed that in recent years non-gradient nature based, probabilistic search algorithms (which generally mimic some natural phenomena) have attracted much attention from the scientific community due to features such as easiness to code, ability of handle non-continuous functions, capability of using parallel architectures, and the trend of finding the global, or near global, solution. The large number of function evaluations required by these methods often presents a drawback when compared with classical gradient-based algorithms, despite the advances in computational throughput that have helped the use this class of algorithm in real world problems (Venter and Sobieszczanski-Sobieski, 2003 and Viana and Steffen Jr, 2005).

As introduced in Chapter I, the focus of this work is on heuristic and population-based optimization algorithms (i.e. algorithms that use a set of points updated via statistical rules for exploration of the design space during the optimization task). More precisely, Ant Colony Optimization (ACO), Differential Evolution (DE), Genetic Algorithms (GA), Particle Swarm Optimization (PSO), LifeCycle Optimization (LC), and the Enhanced Stochastic Evolutionary Algorithm (ESEA) are the set of algorithms studied in this thesis. These methods generate new points in the design space by applying operators to current points and statistically moving toward more optimal places in the design space. They rely on a heuristic-driven search of a finite solution space and do not require function derivatives. Consequently, they

can deal with discrete variables and non-convex and non-continuous functions. The following sections describe specific details as well as the common ground of the algorithms.

3.2. Ant Colony Optimization

Ant Colony Optimization (ACO) is inspired in the behavior of real ants and their communication scheme by using pheromone trail (Dorigo, 1992). When searching for food, real ants start moving randomly, and upon finding food they return to their colony while laying down pheromone trails (Socha, 2004). This means that if other ants find such a path, they return and reinforce it. However, over time the pheromone trail starts to evaporate, thus reducing its attractive strength. When a short and a long path are compared, it is easy to see that a short path gets marched faster and thus the pheromone density remains high. Consequently, if one ant finds a short path (from the optimization point of view, it means a good solution) when marching from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually encourages all the ants in following the same single path. ACO follows some basic concepts, as presented below (Socha, 2004):

- A search performed by a population of ants, i.e., by simple independent agents.
- Incremental construction of solutions.
- Probabilistic choice of solution components based on stigmergic information of pheromone. A stigmergic process is the process through which the results of a worker insect's activity act as a stimulus to further activities.
- No direct communication between ants.

The outline of a basic ACO algorithm is presented in Figure 3.1.

The question that arises is how ACO, first used to solve combinatorial problems (Dorigo et al., 1996, Dorigo and Gambardella, 1997), deals with continuous variables. A possible answer to this question would be the conversion of the continuous variables and the pheromone trail into a string of bits as proposed in the past by Monmarché et al. (1999). Instead, the present continuous version extends ACO in order to work directly with continuous variables.

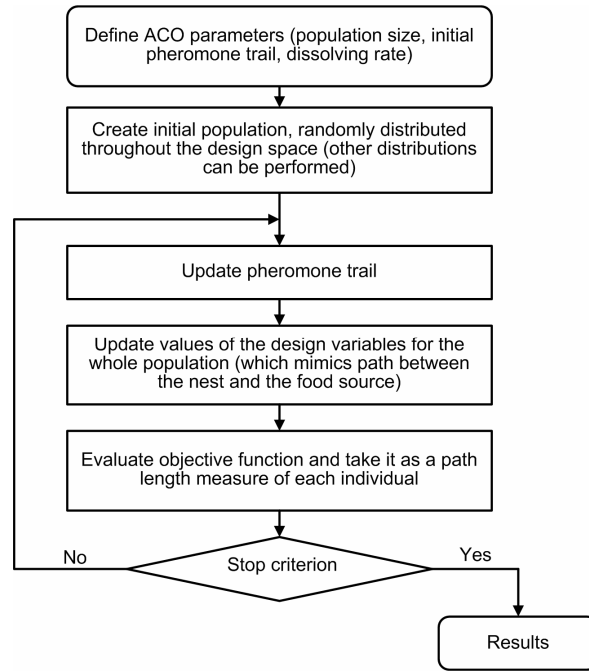


Figure 3.1. Basic scheme for ACO.

Following the idea of directly using continuous variables, a population of n_{pop} ants (herein referred to as individuals) is expressed as a $n_{pop} \times n_{dv}$ matrix, $\mathbf{P} = [\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_{n_{pop}}]^T$. The continuous version of ACO models the pheromone trail as an amount of pheromone laid on the path. As suggested by Socha (2004) and Pourtakdoust and Nobahari (2004), for the continuous model implementation, this is done by using a normal probability distribution function (PDF). Then, for the i -th dimension of the design space, the pheromone trail, τ_i , is given by:

$$\tau_i(x) = \exp\left(-\frac{(x - x_i^*)^2}{2(\sigma_i^{ACO})^2}\right) \quad (3.1)$$

$$\sigma_i^{ACO} = \sqrt{\frac{1}{n_{pop} - 1} \sum_{j=1}^{n_{pop}} (z_j - \bar{z})^2} \quad (3.2)$$

where:

- $\exp(\cdot)$ is the exponential function ;
- x_i^* is the i -th coordinate of the best point found by the optimization task within the design space until the current iteration;

- σ_i^{ACO} is an index related to the aggregation of the population around the current minimum for the i -th coordinate of the design space;
- $\mathbf{z}$ is the i -th column of the population matrix $\mathbf{P}$; and
- $\bar{z}$ is the mean value of the vector $\mathbf{z}$.

The updating process of the values of each design variable for all individuals is the process in which, for a given iteration, each individual sets the values for the trial solution based on the probability distribution given by Eq. (3.1). Computationally, this is implemented through a random number generator based on a normal PDF. From Eq. (3.1), it can be noticed that each variable uses a different random number generator together with its respective PDF. About the pheromone scheme, it is possible to see that the concentration of pheromone increases in the area of the candidate to the optimum. This approach (also called as positive update) reinforces the probability of the choices that lead to good solutions. However, for avoiding premature convergence, negative update procedures are not discarded, as described by Socha (2004). In this work, a simple method to perform negative update is used: dissolving of the pheromone. The idea of this scheme is to spread the amount of pheromone by changing the current standard deviation (for each variable) according to the following equation:

$$\sigma_{new}^{ACO} = \gamma \sigma_{old}^{ACO} , \quad (3.3)$$

where $\gamma > 1$ is the dissolving rate (usually $\gamma = 1.25$). The dissolving rate affects the exploration capabilities, and consequently, the convergence of the algorithm.

At the initialization the algorithm:

- $\mathbf{x}_{min}$ is randomly chosen within the design space using a uniform PDF;
- σ^{ACO} is taken as being 3 times greater than the length of the search interval.

The SIMPLE Optimization toolbox of Viana and Steffen Jr (2008) was used to run ACO algorithm.

3.3. Differential Evolution

Differential Evolution (DE) was proposed by Storn and Price (1995) as an algorithm to solve global optimization problems of continuous variables. Even though the DE community has adopted biological terms to describe arithmetic operations on real (and discrete) variables, i.e., mutation and recombination, the operations are more akin to dynamics/thermodynamics and physics than they are to biology. Selection is perhaps the

DE's most Darwinian aspect, but even here, competition for survival is more limited than in most evolutionary algorithms. The main idea behind DE is how possible solutions taken from the population of individuals are set, recombined and chosen to evolve the population to the next generation (Storn and Price, 1997, Ronkkonen et al., 2005). This way, the weighted-difference scheme seems to be the most important feature of the present heuristic.

In a population of individuals (seen as potential solutions), a fixed number of points (or vectors in the DE nomenclature) are randomly initialized, and then evolved over the optimization task to explore the design space and hopefully to locate the optimum of the objective function. At each iteration, new vectors are generated by the combination of vectors randomly chosen from the current population (this is also referred to as “mutation” in the DE literature). The out coming vectors are then mixed with a predetermined target vector. This operation is called “recombination” and produces a “trial vector”. Finally, the “trial vector” is accepted for the next generation if and only if it yields a reduction in the value of the objective function. This last operator is referred to as “selection.” As can be seen, the basic algorithm preserves some common aspects of the traditional simple GA (specially the nomenclature of some operators, such as selection, crossover and mutation).

Again, a population of n_{pop} individuals is expressed as a $n_{pop} \times n_{dv}$ matrix,

$\mathbf{P} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_{n_{pop}} \end{bmatrix}^T$. The outline of a basic DE is as shown in Figure 3.2.

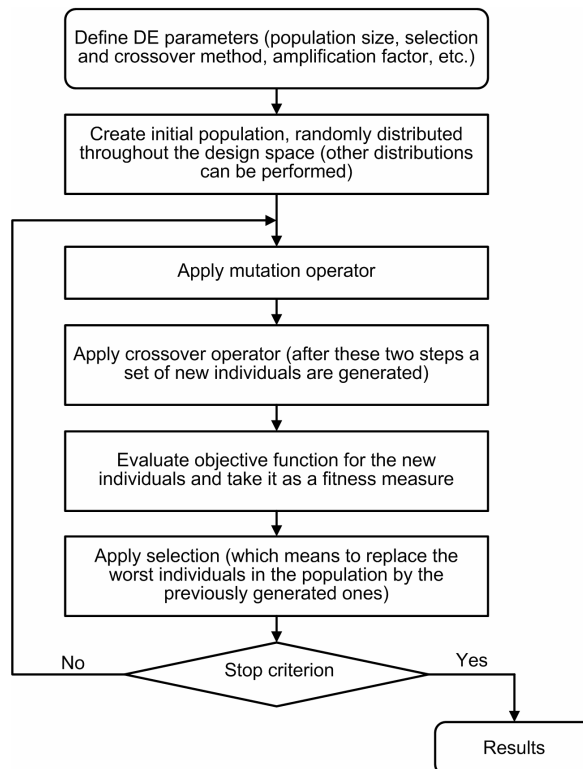


Figure 3.2. Basic scheme for DE.

The DE community has established the following notation to determine how the mutation and crossover operators work together (Storn and Price, 1995; Storn and Price, 1997; and Ronkkonen et al., 2005): $DE / a / b / c$, where:

- a specifies the vector to be mutated, which currently can be one of the following:
 - “rand”: a randomly chosen population vector,
 - “best”: the vector of best objective value from the current population,
 - “rand-to-best”: takes both randomly and best individuals.
- b is the number of difference vectors used, and
- c denotes the crossover scheme, which can be either “bin” (independent binomial experiments) or “exp” (exponential crossover).

3.3.1. Mutation

As described before, the DE mutation is the operator that first adds the weighted difference between two individuals to a third individual. Table 3.1 gives more details about each mutation scheme.

Table 3.1 Different implementations of the DE mutation.

Type	Updating equation	Target vector	Population size
$rand / 1$	$\mathbf{x}_{trial} = \mathbf{x}_{r1} + F(\mathbf{x}_{r2} - \mathbf{x}_{r3})$	$\mathbf{x}_{r1}$	> 3
$rand / 2$	$\mathbf{x}_{trial} = \mathbf{x}_{r1} + F(\mathbf{x}_{r2} - \mathbf{x}_{r3} + \mathbf{x}_{r4} - \mathbf{x}_{r5})$	$\mathbf{x}_{r1}$	> 5
$best / 1$	$\mathbf{x}_{trial} = \mathbf{x}_{best} + F(\mathbf{x}_{r2} - \mathbf{x}_{r3})$	$\mathbf{x}_{best}$	> 3
$best / 2$	$\mathbf{x}_{trial} = \mathbf{x}_{best} + F(\mathbf{x}_{r1} - \mathbf{x}_{r2} + \mathbf{x}_{r3} - \mathbf{x}_{r4})$	$\mathbf{x}_{best}$	> 5
$rand\text{-}to\text{-}best / 1$	$\mathbf{x}_{trial} = \mathbf{x} + F(\mathbf{x}_{best} - \mathbf{x} + \mathbf{x}_{r1} - \mathbf{x}_{r2})$	$\mathbf{x}$	> 5
$rand\text{-}to\text{-}best / 2$	$\mathbf{x}_{trial} = \mathbf{x} + F(\mathbf{x}_{best} - \mathbf{x}) + F(\mathbf{x}_{r1} - \mathbf{x}_{r2} + \mathbf{x}_{r3} - \mathbf{x}_{r4})$	$\mathbf{x}$	> 6

where:

- $\mathbf{x}_{trial}$ is the trial vector (individual);
- $r1, r2, r3, r4, r5 \in \{1, 2, \dots, m\}$ are random integer indexes and mutually different;

- F is a real constant factor $\in [0, 2]$, which controls the amplification of the differential variation;
- $\mathbf{x}_{best}$ is the best individual of the current population; and
- $\mathbf{x}$ is the current individual.

In DE, the second step of the design variable update is the crossover operator. There are two types of crossovers: binary and exponential. The superiority of each crossover scheme over the other can not be uniquely defined, which means that this is a problem-dependent operator. Thus, the final trial vector, $\mathbf{x}_{trial}$, is formed in such a way that the i -th coordinate is given by:

$$x_{trial} = \begin{cases} x_i^{mutation}, & \text{if crossover is applicable,} \\ x_i, & \text{otherwise,} \end{cases} \quad i = 1, 2, \dots, n_{dv}. \quad (3.4)$$

In practical terms, DE has a parameter that controls the crossover, namely the crossover probability, CR . In exponential crossover, the crossover is performed in a loop controlled by a uniform random number, rn , and a counter, i , that ranges from 1 to n_{dv} . The first time a randomly picked number between 0 and 1 goes beyond the CR value, no crossover is performed and the remaining of the n_{dv} variables is left intact (Storn and Price, 1995; Storn, 1996; Storn and Price, 1997; and Ronkkonen et al., 2005). Figure 3.3 shows a flowchart for the exponential crossover.

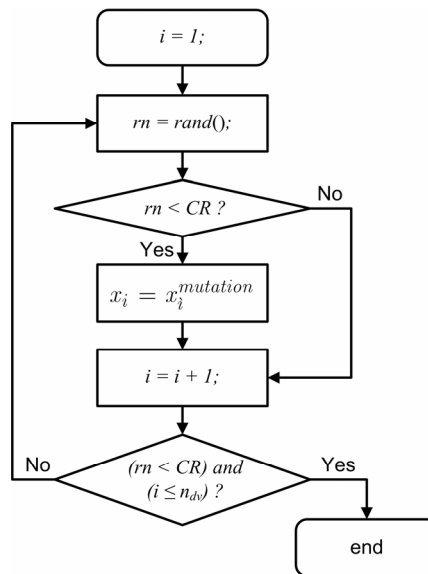


Figure 3.3. Basic scheme for the DE exponential crossover. Consider $rand()$ as a function for generating a uniform random number.

In the binomial crossover, the operation is performed on each of the n variables whenever a randomly picked number between 0 and 1 is lesser than the CR value (Storn and Price, 1995; Storn, 1996; Storn and Price, 1997; and Ronkkonen et al., 2005). Figure 3.4 shows a flowchart for the binomial crossover.

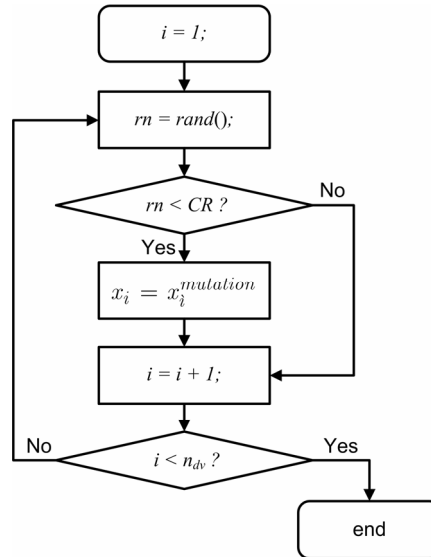


Figure 3.4. Basic scheme for the DE binomial crossover. Consider $\text{rand}()$ as a function for generating a uniform random number.

It can be noticed that for high values of CR , the exponential and binomial crossovers yield similar results.

3.3.2. Selection

Selection is the simplest DE operator. It is used to decide whether or not the new vector, $\mathbf{x}_{new}$, should become a member of the next generation. The approach is straightforward: the objective function value of the new individual, $f(\mathbf{x}_{new})$, is compared with the one of the target vector, $f(\mathbf{x}_{target})$. If there is an improvement, $\mathbf{x}_{new}$ is set to the next generation; otherwise, $\mathbf{x}_{target}$ is retained.

3.3.3. Remarks about DE

Storn (1996) gives a set of practical rules useful when it comes to choose the DE parameters, namely:

- A population size of 10 times the number of design variables is a good choice for many applications. However, it is worth mentioning that various scientists have been successful in using a smaller population for DE.

- Most often the crossover probability, CR , must be considerably lower than one (e.g. 0.3). However, if no convergence can be achieved, $CR \in [0.8, 1]$ should be tried. This is especially true when using exponential crossover.
- The higher the population size is chosen, the lower the weighting factor F .
- F is usually chosen $\in [0.5, 1]$.

Last but not least, it is worth mentioning that by combining the six mutation schemes shown in Table 3.1 with the two previously presented crossover schemes, a total number of twelve DE strategies can be explored. By following the $DE/a/b/c$ notation: DE/rand/1/bin, DE/rand/1/exp, DE/rand/2/bin, DE/rand/2/exp, DE/best/1/bin, DE/best/1/exp, DE/best/2/bin, DE/best/2/exp, DE/rand-to-best/1/bin, DE/rand-to-best/1/exp, DE/rand-to-best/2/bin, and DE/rand-to-best/2/exp.

The code developed by Price et al. (2005) was used to execute the DE algorithm.

3.4. Genetic Algorithm

Genetic Algorithm (GA) is an optimization algorithm used to find approximate solutions to difficult-to-solve problems through the application of the principles of evolutionary biology to computer science. GA is based on Darwin's theory of survival and evolution of species, as explained by Gen and Cheng (1999), Michalewicz and Fogel (2000) and Haupt and Haupt (2004). GA uses biologically-derived concepts such as inheritance, mutation, natural selection, and recombination (or crossover). Due to the vast literature on GA, for the sake of simplicity, just an overview is provided in this text.

The algorithm starts from a population of random individuals, viewed as candidate solutions to a problem. During the evolutionary process, each individual of the population is evaluated, reflecting its adaptation capability to the environment. Some of the individuals of the population are preserved while others are discarded; this process mimics the natural selection in Darwinism. The members of the remaining group of individuals are paired in order to generate new individuals to replace those that are discarded in the selection process. Finally, some of them can be submitted to mutation, and as a consequence, the chromosomes of these individuals are altered. The entire process is repeated until a satisfactory solution is found. The outline of a basic GA is as shown in Figure 3.5.

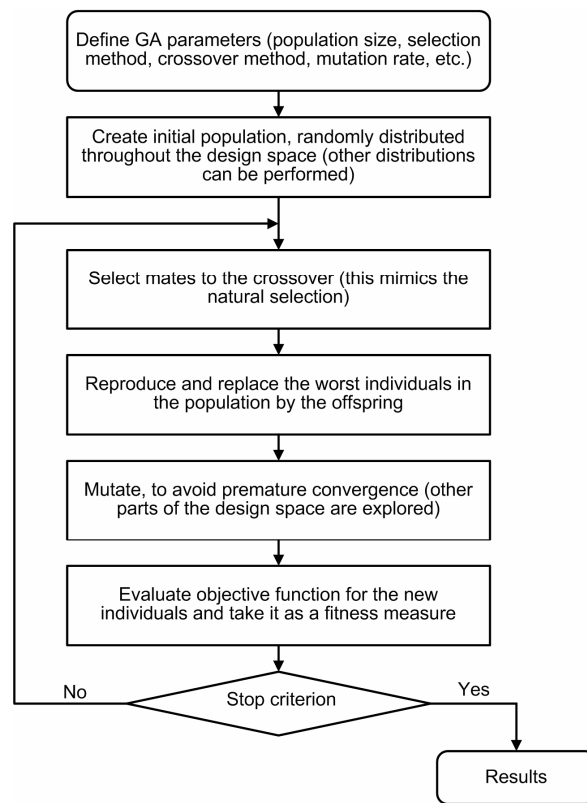


Figure 3.5. Basic scheme for GA.

Although the initially proposed GA algorithm was dedicated to discrete variables only, nowadays improvements are available to deal with discrete and continuous variables (see Gen and Cheng, 1999; Michalewicz and Fogel, 2000; and Haupt and Haupt, 2004).

The SIMPLE Optimization toolbox of Viana and Steffen Jr (2008) was used to run GA algorithm.

3.5. Particle Swarm Optimization

Particle Swarm Optimization (PSO) was introduced by Kennedy and Eberhart (1995), as emerged from experiences with algorithms inspired in the social behavior of some bird species. Consider the following situation: a swarm of birds searching for food around a delimited area. Suppose there is just one place with food and the birds do not know where it is. The success of one bird is shared by the whole swarm (learning from the experience of other individuals). In this sense, the adjustment between exploration (the capacity of individual search) and exploitation (taking advantage of someone else's success) is required. If there is little exploration, the birds will all converge on the first good place encountered. On the other hand, if there is little exploitation, the birds will slowly converge or they will try to find food alone. It is clear that the best policy is a trade-off between both policies.

In PSO, the flight of each bird (herein referred as individual) is modeled by using a velocity vector, which considers the contribution of the current velocity, as well as two other parts accounting for the self-knowledge of the individual and the knowledge of the swarm (herein referred to as population) about the search space. This way, the velocity vector is used to update the position of each individual in the population (Kennedy and Eberhart, 1995; Parsopoulos and Vrahatis, 2002; and Venter and Sobieszczanski-Sobieski, 2003). An outline of a basic PSO algorithm is shown in Figure 3.6.

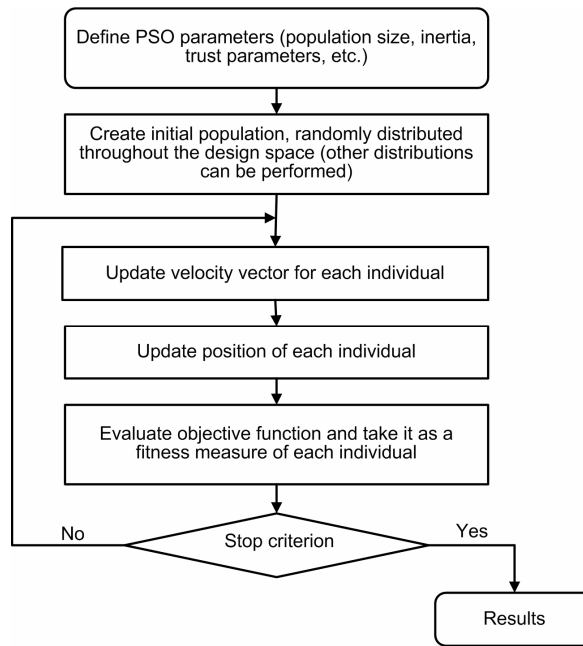


Figure 3.6. Basic scheme for PSO.

The position of each individual is updated according to the following equation:

$$x_k^i = x_{k-1}^i + v_k^i \Delta t, \quad (3.5)$$

where x_k^i is the position of the individual i in the iteration k ; v_k^i represents the corresponding velocity vector; and Δt is the time step (generally $\Delta t = 1$).

The velocity vector is updated as follows:

$$v_k^i = w v_{k-1}^i + c_1 r_1 \frac{(p^i - x_{k-1}^i)}{\Delta t} + c_2 r_2 \frac{(p_{k-1}^s - x_{k-1}^i)}{\Delta t}, \quad (3.6)$$

where w is the inertia of the individual; c_1 and c_2 are the two "trust" parameters; r_1 and r_2 are random numbers between 0 and 1; p^i is the best position found by the individual i up to the current iteration; and p_{k-1}^s is the best individual in the iteration $k - 1$.

Equation (3.6) can be interpreted as a sum of three different contributions. The first term, pondered by the inertia, w , gives how much of “exploration” there will be in the resulting velocity. The second term, pondered by the self trust parameter, c_1 , indicates how much the knowledge of the individual itself there will be in the resulting velocity. Finally, the term weighted by the trust on the population, c_2 , shows how much of the population success will influence the resulting velocity. As a consequence, the larger the inertia value, the more global (as opposed to individualistic) will be the behavior. The trust parameters c_1 and c_2 must be set to balance the influence of the amount of knowledge that was acquired by the individual itself and the knowledge acquired by the population.

3.5.1. Initial Population

The initial population is created by randomly distributing the individuals throughout the search space. The initial position and the initial velocity vectors are given by:

$$x_0^i = x_{\min} + r_1(x_{\max} - x_{\min}) , \quad (3.7)$$

$$v_0^i = \frac{x_{\min} + r_2(x_{\max} - x_{\min})}{\Delta t} , \quad (3.8)$$

where $x_{\min}$ is the lower bound vector; and $x_{\max}$ is the upper bound vector for the variables.

3.5.2. Algorithm Parameters

The velocity vector update formula has some parameters that can be adjusted according to the problem, namely the trust parameters, c_1 and c_2 , and the inertia weight, w .

Kennedy and Eberhart (1995) initially suggested $c_1 = c_2 = 2$, as a way to equally weight the contribution given by the individual and the population. Later, various works (Shi and Eberhart, 1998; Fourie and Groenwold, 2002; and Venter and Sobieszczanski-Sobieski, 2003) proposed that trust parameters could be set to different values, generally satisfying $c_1 + c_2 = 4$. In this work, the default values are $c_1 = 1.5$ and $c_2 = 2.5$, as used by Venter and Sobieszczanski-Sobieski (2003).

The inertia parameter, in general, is such that $0.8 \leq w \leq 1.4$. In addition, a scheme of dynamic reduction of w has several advantages (Venter and Sobieszczanski-Sobieski, 2003). It improves the convergence to the optimum solution and makes the problem less sensitive to the w parameter. The idea is to start with a large value for w , since the knowledge bases of the individual and the population are empty, and then, to reduce

continuously that value along the optimization process, privileging the flying experiences. The following equation gives the updating scheme for w :

$$w_{new} = f_w w_{old} , \quad (3.9)$$

where w_{new} is the updated value; w_{old} is the previous value; and f_w is a constant between 0 and 1 (usually, $f_w = 0.975$).

The inertia is not updated each iteration. Instead, a dispersion measure, DM , as will be described in Section 3.7.1, of a subset of the individuals is used to guide when to apply the inertia reduction. If the DM value of this subset of individuals falls below a pre-defined threshold value, it is understood that the algorithm is converging to an optimum, then Eq. (3.9) is applied. As suggested by Venter and Sobieszczanski-Sobieski (2003), a subset of the best 20% of the individuals from the population is monitored and $DM_{thresh} = 0.1 \times D_{max}$ is used as threshold for DM .

3.5.3. Dealing with Violated Constraints

When in an optimization problem the individuals violate the constraints, a strategy for repairing the violation has to be considered. As suggest by Venter and Sobieszczanski-Sobieski (2003), the idea of feasible directions (Vanderplaats, 2005) is used to achieve this goal. This way, when the constraints are violated, the velocity vector is recomputed according to the equation:

$$v_k^i = c_1 r_1 \frac{(p^i - x_{k-1}^i)}{\Delta t} + c_2 r_2 \frac{(p_{k-1}^s - x_{k-1}^i)}{\Delta t} \quad (3.10)$$

After obtaining the new velocity vector, the position is recalculated by using Eq. (3.5). The difference between this formula and the initial one is that the current velocity is neglected. Since just the elements of the best solution found by the individual and the population are considered, it is expected that this new velocity vector will bring the individual back to a feasible region of the search space. When this is not the case, Eq. (3.10) is applied repetitively until the individual falls in a feasible region.

The SIMPLE Optimization toolbox of Viana and Steffen Jr (2008) was used to execute PSO algorithm.

3.6. LifeCycle Algorithm

LyfeCycle (LC) algorithm is a hybrid heuristic optimization method inspired by the idea of life cycle stages, initially proposed by Krink and Løvberg (2004). From the mathematical point of view, natural algorithms such as ACO, DE, GA, and PSO are heuristic search methods of proven efficiency as optimization tools. LC is intended to avoid poor performance of individual methods and creates a self-adaptive optimization approach (Flores et al., 2007). Each individual, as a candidate solution, decides based on its success if it would prefer to belong to a population of an ACO, DE, GA, or PSO algorithm. This means that various heuristic techniques contribute to form a robust high performance optimization tool. The idea is that complex problems can be conveniently considered from the optimization viewpoint. As can be seen, the less successful individuals must change their status in order to improve their fitness. This means that the optimization approach does not follow a rigid scheme, in which various techniques are used sequentially in a cascade-type of structure. In other words, it is the mechanism of self-adaptation to the optimization problem that rules the procedure. A LC outline follows in Figure 3.7.

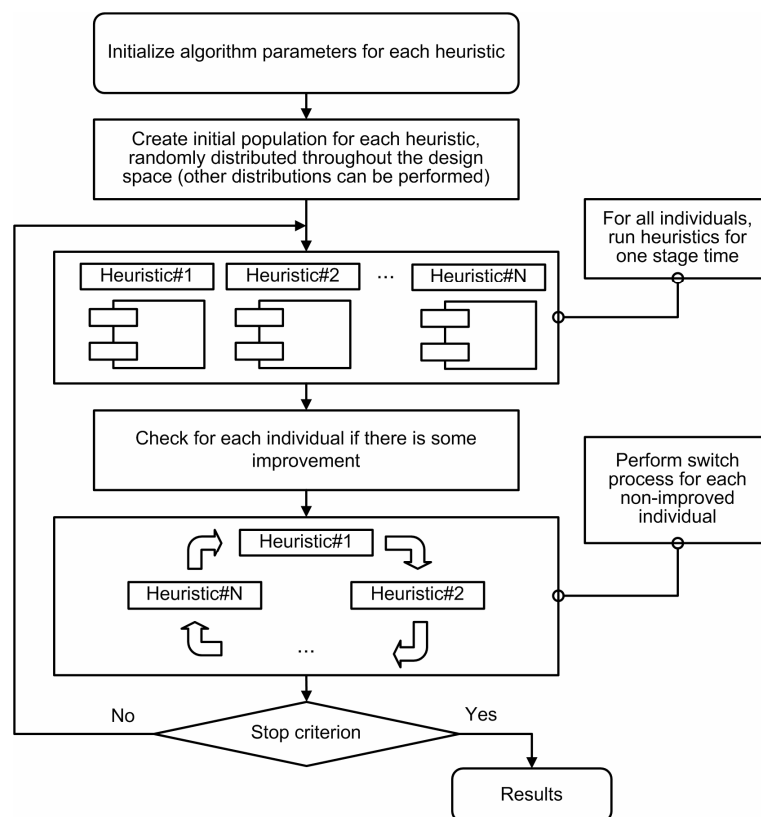


Figure 3.7. Basic scheme for LC.

Since the algorithm is composed by various heuristics, it is necessary to set the parameters of every heuristic used in the LC. Nevertheless, there is a parameter inherent to

the LC, namely the number of iterations that represent a stage of the LC, known as stage interval. At the end of each stage interval, the less successful individuals must change their stage in order to improve their fitness. During the optimization procedure the agents of each subpopulation commute to the other in such a way that its own fitness is improved. In other words, it is the mechanism of self-adaptation to the optimization problem that rules the procedure. To close the definition, LC stages must be presented. In the present work, two or more of the previously presented heuristics are used as stages. Other versions of the LC can be proposed by considering other heuristics and a mix of them, as shown by Krink and Løvberg (2004).

The SIMPLE Optimization toolbox of Viana and Steffen Jr (2008) was used to run LC algorithm.

3.7. Common Ground of the Basic Algorithms

Despite the differences on the inspiration of the previously presented algorithms, they share common points such as:

- Modeling a set of points on the design space as a population of individuals: each individual represents a vector of design variables, whose values are iteratively updated according to a heuristic rule.
- Ability to have the population split in clusters during the optimization task: this is when the population is divided in sub-populations, or clusters, that run the optimization separately from each other. Eventually, these clusters can exchange information when a migration operator is applied.
- Possibility of applying elitism and additional randomness operators: even though these two operators were first used in GA, they can be easily extended to all previously discussed methods. In fact, DE uses a sort of the elitism during the selection operator, and the PSO community has already tested a randomness operator as can be viewed in Venter and Sobieszczanski-Sobieski (2003).

Therefore, a unified framework is naturally proposed, as illustrated by Figure 3.8.

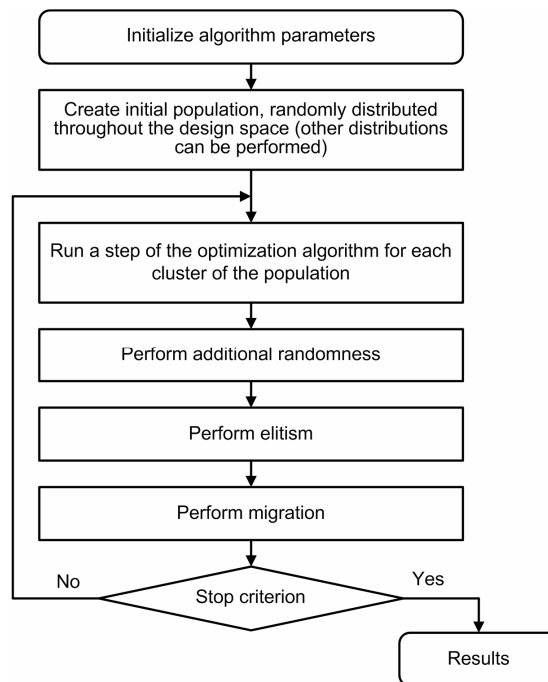


Figure 3.8. Framework for population-based heuristic optimization algorithms.

Going deeper on the discussion of each step:

1. Initialize algorithm parameters: this is when different operators and parameters are set, including both general, such as migration scheme and population size, and heuristic-specific, such as the GA crossover operator or the PSO trust parameters.
2. Create the initial population: at this step, the design space is sampled. The intention is to raise as much information as possible. To accomplish that, a random sampling or other statistic-based sampling strategies are commonly applied.
3. Run a step of the optimization algorithm for each cluster of the population: as described in previous sections, each of the algorithms has an internal loop that besides dealing with specific data structures, basically, performs selection, generation and replacement of individuals in the population. It is important to notice that the evaluations of the functions occur during this step.
4. Perform additional randomness: as in the case of the GA mutation, an additional randomness can be applied to the new population in order to avoid premature convergence so that other parts of the design space can be explored. It is worth mentioning that since LC performs complete steps of each algorithm, there is no need to be redundant here. Therefore, LC has no additional randomness operator.
5. Perform elitism: a pre-defined number of individuals are kept from iteration to iteration according to their rank in the population.
6. Perform migration: when the population is divided in clusters, migration refers to the transfer of an individual from one cluster to another. This allows clusters to

communicate, increasing diversity. A migration rate must be chosen to determine how often individuals migrate to other clusters. Cantu-Paz (2001) reviews several migration schemes. In this work, a simple ring-type scheme is implemented, i.e., the n -th cluster migrates into the $(n + 1)$ -th cluster (the last cluster migrates into the first). A migration fraction parameter controls how many individuals move from one cluster to the other and the migration interval controls how many iterations occur before another migration occurs.

7. Stop criterion: this is necessary to stop the optimization task avoiding any additional function evaluations.

Now that the common framework is clear, the following sub-sections cover details about how to deal with discrete/integer variables, different implementations of the randomness operator, and a set of stopping criteria.

3.7.1. Dispersion Measure of the Population

Some of the operators are only applied when a dispersion measure of the population follows below a threshold. Since all design variables are normalized between 0 and 1, a convenient dispersion measure, DM , can be defined as follow:

$$DM = \frac{\bar{D}}{D_{\max}} \quad (3.11)$$

where:

- $\bar{D} = \frac{1}{n_{pop}} \sum_{i=1}^{n_{pop}} \|\mathbf{x}_i - \bar{\mathbf{x}}\|$: defines the dispersion of the population and it is equivalent to the mean value of the Euclidian distance between each individual and $\bar{\mathbf{x}}$.
- $\bar{\mathbf{x}}$: is the vector of mean values of each design variable for the population $\mathbf{P}$, i.e. $\bar{x}_j = \frac{1}{n_{pop}} \sum_{i=1}^{n_{pop}} \mathbf{P}_{ij}$ (i and j represents the design variable and the individual of the population, respectively).
- $D_{\max} = \frac{1}{2} \|\mathbf{1}_{n_{dv} \times 1}\|$: defines the maximum possible dispersion of the population.

Each of the n_{dv} elements of the vector $\mathbf{1}$ is equal to 1. Assuming that all individuals are uniformly distributed on the borders of the design space, $D_{\max}$ is the mean value of the Euclidian distance between each individual and the center of the design space.

DM represents the ratio between the current dispersion of the population around its mean value and the maximum possible dispersion. Then, the smaller the DM value the more aggregated is the population.

Frequently, the operators are only applied when a dispersion measure of the population follows below a threshold, DM_{thresh} , which can be calculated by:

$$DM_{thresh} = thresh \times D_{\max} \quad (3.12)$$

where $0 \leq thresh \leq 1$ is the threshold value.

3.7.2. Discrete/Integer Variables

ACO, DE, GA, PSO and LC differ historically at this point, since ACO was originally introduced to solve combinatorial optimization problems, DE, PSO and LC were first used to deal with continuous problems and, GA was initially designed to solve discrete problems. However, this aspect does not make unfeasible the implementation of each algorithm to solve continuous, discrete or discrete-continuous problems. A simple modification can make the algorithms able to deal with the design variables as integer numbers. The approach is straightforward: the set of discrete variables of each individual is rounded to its closest integer value before evaluating the objective function. This method, although simple, has shown to be quite effective in several problems tested in this research.

3.7.3. Additional Randomness

To avoid premature convergence of the algorithms, additional randomness is introduced using a randomness operator. The randomness operator behaves like the mutation operator in GA. The randomness operator used here changes the vector of design variables that represents an individual of the population.

In the present implementation, the individuals to be modified are identified by using the dispersion measure DM , as in the case of the inertia reduction for the PSO algorithm. If DM falls below a pre-defined threshold value, it is assumed that the population is becoming too uniform. In this case, a random subset of the population is subjected to the randomness operator. As in the case of the GA mutation operator, a randomness operator rate, r_{frac} , controls the maximum size of this subset. Finally, $DM_{thresh} = thresh \times D_{\max}$ is used as threshold value for DM .

Table 3.2 summarizes the different implementations for the randomness operator used in this work.

3.7.4. Stopping Criteria

A convergence criterion is necessary to avoid any additional function evaluations after an optimum solution is found. Ideally, the convergence criterion should not have any problem specific parameters. In this work, the algorithms repeat an iterative loop until at least a stop criterion is achieved. The implemented stopping criteria are shown in Table 3.3.

Table 3.2. Randomness operators.

Name	Description
Boundary	It changes randomly one of the parameters of the individual either to its upper or its lower boundary value.
Non-uniform	It changes one of the parameters of the individual by using a Gaussian distribution. This distribution starts wide, and narrows to a point distribution as the current iteration approaches the maximum number of iterations.
Multi-non-uniform	It changes all of the parameters of the individual based on a Gaussian distribution. This distribution starts wide, and narrows to a point distribution as the current iteration approaches the maximum number of iterations.
Uniform	It changes one of the parameters of the individual by using a uniform probability distribution.

Table 3.3. Stopping criteria.

Criterion	Description	Default value
Iterations	Maximum number of iterations.	100
Function Evaluations	Maximum number of function evaluations.	$n_{pop} \times \text{Iterations}$
Time	Total time (in seconds) allowed for the optimization task.	$+\infty$
Objective limit	The lowest value allowed for the objective function.	$-\infty$
Iterations for convergence	Number of consecutive iterations without improvements allowed for the optimization task.	15
Time for convergence	Time without improvements allowed for the optimization task.	$+\infty$

The SIMPLE Optimization toolbox of Viana and Steffen Jr (2008) is also used for an easy manipulation of all different codes previously presented.

3.8. Enhanced Stochastic Evolutionary Algorithm

The Enhanced Stochastic Evolutionary Algorithm (ESEA) is a modified version introduced by Jin et al. (2005) in the Stochastic Evolutionary Algorithm, initially developed by Saab and Rao (1991). The ESEA, as shown in Figure 3.9, consists of an inner loop and an outer loop, as described in the following. The inner loop constructs new designs by an element-exchange approach and dictates whether to accept the designs based on a certain acceptance criterion dealing with the location of the points. A set of n_D designs is taken by exchanging two elements within the column and then the best of this n_D designs is chosen to be compared with the current best solution. The outer loop controls the entire optimization process by adjusting the threshold value, T_h , in the acceptance criterion of the inner loop. According to Jin et al. (2005), this dynamic adjustment of the acceptance criterion enables the algorithm to avoid local minima designs. During this process, $\mathbf{x}_{best}$ is used to keep track of the best design.

3.8.1. Inner Loop

The inner loop is responsible for constructing new designs by using an element-exchange approach. The new designs can be accepted or rejected, based on an acceptance criterion that deals with the location of the points. As can be seen in Figure 3.9 (b), the inner loop has M iterations. At iteration i , a set of n_D designs is created by exchanging two elements within the column of the current design, $\mathbf{x}$, with the best of these n_D designs, $\mathbf{x}_{trial}$, selected for comparison with the current solution, $\mathbf{x}$. The substitution of $\mathbf{x}$ by $\mathbf{x}_{trial}$ depends on the acceptance criterion given by:

$$\Delta f \leq T_h \times \text{rand}(0,1) , \quad (3.13)$$

where:

- $\Delta J = J(\mathbf{x}_{trial}) - J(\mathbf{x})$,
- $\text{rand}(0,1)$ is a function that generates uniform random numbers between 0 and 1, and
- $T_h > 0$ is the threshold control parameter.

$\mathbf{x}_{trial}$ will be accepted only if it satisfies $0 \leq \Delta J \leq T_h$ and the probability of acceptance given by:

$$P\left(S \geq \Delta J / T_h\right) = 1 - \Delta J / T_h . \quad (3.14)$$

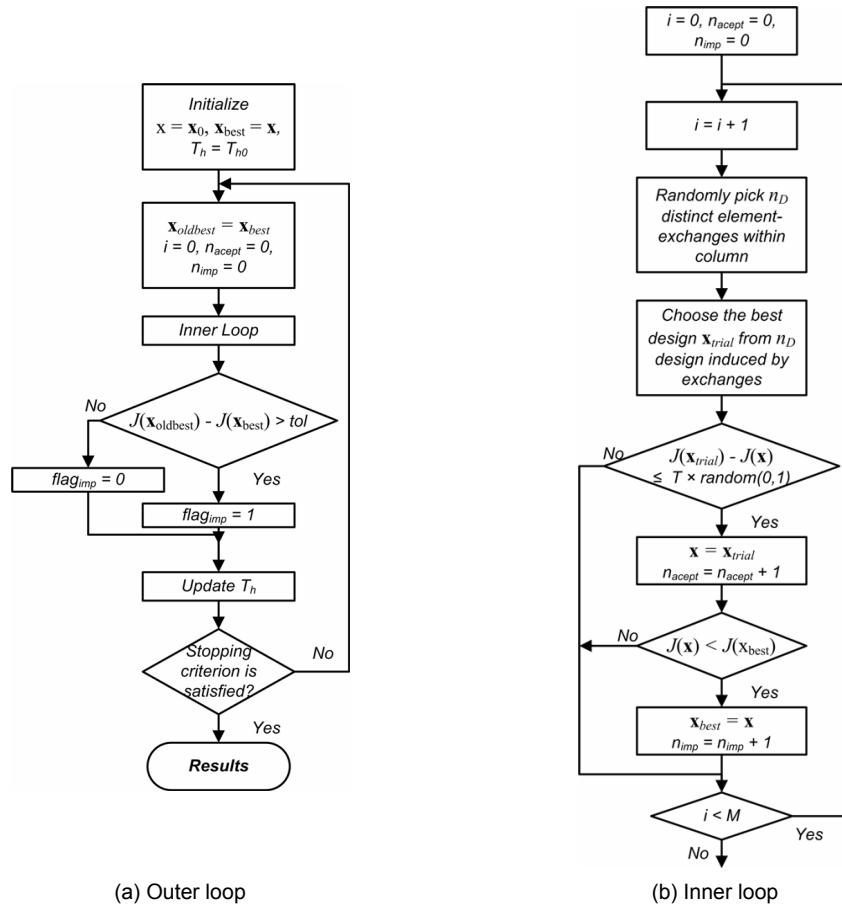


Figure 3.9. Basic scheme for ESEA.

Using the scheme guided by Eqs. (3.13) and (3.14), the algorithm attempts to avoid local solutions. It can be noticed that a temporarily worse design, $\mathbf{x}_{trial}$, could be accepted to replace the current design, $\mathbf{x}$. Table 3.4 gives the values for the inner loop parameters, according to Jin et al. (2005).

3.8.2. Outer Loop

The outer loop controls the entire optimization process by adjusting the threshold value T_h in the acceptance criterion of the inner loop. Initially, T_h is taken as a small fraction of the initial design ($T_h = 0.005 \times J(\mathbf{x}_0)$) and its value is dynamically updated during the search process. The search process is divided in two main stages: (a) the *improving process*, which

attempts to find a locally optimal design, and, (b) the *exploration process*, which try to escape from the locally optimal design. These two stages are now discussed in more detail.

Table 3.4. Inner loop parameters.

Parameter	Value	Comments
n_D	$n_e/5$, but no larger than 50	n_e is the number of all possible distinct $k = 2$ element-exchanges in a column. In the Latin Hypercube case $n_e = \frac{n_{dv}!}{k!(n_{dv} - k)!}$
M	$2 \times N \times n_{dv}/J$, but no larger than 100 (if the candidate solution is represented as a vector $N = 1$; otherwise, N is the number of rows)	M is the number of iterations in the inner loop

Improving process: The search process switches to the improving process if the objective function is improved after completing an entire inner loop. This mechanism is controlled by $flag_{imp}$, i.e. $flag_{imp} = 1$. Once in the improving process, T_h is adjusted to rapidly find a local optimal design. This is done keeping the value of T_h small, so that only a better, or a slightly worse, design is accepted to replace $\mathbf{x}$. T_h is updated based on the acceptance ratio n_{acpt}/M (number of accepted design divided by the number of tries in the inner loop) and the improvement ratio n_{imp}/M (number of improved design divided by the number of tries in the inner loop). Thus, there are the following possibilities:

1. T_h will decrease if the acceptance ratio is larger than a small percentage (e.g., 10%) and the improvement ratio is less than the acceptance ratio.
2. T_h will remain constant if the acceptance ratio is larger than the small percentage and the improvement ratio is equal to the acceptance ratio (meaning that T_h is so small that only improving designs are accepted by the acceptance criterion).
3. T_h will increase otherwise. The following equations are used to decrease and increase T_h respectively:

$$T_h^{new} = \alpha_1 T_h^{old}, \quad (3.15)$$

$$T_h^{new} = T_h^{old} / \alpha_1, \quad (3.16)$$

where $0 < \alpha_1 < 1$. As in Jin et al. (2005), setting $\alpha_1 = 0.8$ worked well in all tests.

Exploration process: If no improvement is made after completing an entire inner loop, the search process will turn to the exploration process. This mechanism is controlled by $flag_{imp}$, i.e. $flag_{imp} = 0$. During the exploration process, T_h is adjusted to help the algorithm escape from a locally optimal design. Unlike the improving process, T_h fluctuates within a range based on the acceptance ratio. If the acceptance ratio is less than a small percentage (e.g., 10%), T_h will rapidly increase until the acceptance ratio is larger than a large percentage (e.g. 80%). If this happens, T_h will slowly decrease until the acceptance ratio is less than the small percentage. This process will be repeated until an improved design is found.

The following equations are used to decrease and increase T_h , respectively:

$$T_h^{new} = \alpha_2 T_h^{old}, \quad (3.17)$$

$$T_h^{new} = T_h^{old} / \alpha_3, \quad (3.18)$$

where $0 < \alpha_3 < \alpha_2 < 1$. As in Jin et al. (2005), $\alpha_2 = 0.8$ and $\alpha_3 = 0.7$ worked well in all tests.

As a result, during the exploration process T_h increases rapidly (so that worse designs could be accepted) to help escaping a local optimal design. T_h decreases slowly for finding better designs after moving away from the local optimal design. For the whole algorithm, the maximum number of cycles is used as the stopping criterion.

ESEA was implemented during an internship of this doctoral research at Vanderplaats Research and Development, Inc. The current version of the VisualDOC general-purpose design optimization software from (VR&D contributors, 2006) uses this module to generate optimum Latin hypercube DOEs. Chapter V shows the results of this collaboration.

To close this chapter, a summary of the contributions to the basic algorithms is given:

- Multiple schemes for DE and GA: the developed computational code allows the use of multiple schemes for both DE and GA. For instance, DE can be set up with a combination of DE/rand/1/bin and DE/best/1/exp; or GA can be set up with two different selection operators (e.g. roulette and tournament).

- Additional randomness for ACO and DE: the general framework as presented by Figure 3.8 makes possible to include additional randomness in these two algorithms, which have limited capabilities of avoiding premature convergence.
- Dispersion measure on the normalized design space: in opposition to schemes based on the function space, the proposed approach prevents difficulties in detecting aggregation of the population when the function space has several minima.
- Implementation of ESEA in an environment of commercial software development to generate the optimal Latin hypercube.

Next chapter presents different approaches on how to couple surrogate modeling and heuristic algorithms during the solution of the optimization problem.

CHAPTER IV

MIXING SURROGATES AND HEURISTIC OPTIMIZATION ALGORITHMS

4.1. Introduction

Chapter I to Chapter III clarified concepts related to the general formulation of the optimization problem, surrogate modeling and heuristic algorithms. The level of accuracy in representing the physical reality was used to define the level of fidelity of the model; the higher the fidelity the more accurate is the model. Chapter III even showed theoretical aspects of some contributions of this research. This is the last theoretical chapter of this thesis; it discusses how to combine surrogate models and heuristic algorithms during the solution of the optimization problem.

As explained in Chapters I and II, surrogate models are used when computational models of high physical fidelity or numerical accuracy are prohibitively expensive in iterative procedures such as in numerical optimization. In this scenario, another alternative is the use of lower-fidelity models, which may not capture a particular feature of the physical phenomenon to the same degree of accuracy as its higher-fidelity counterpart, but may still have satisfactory predictive properties for the purposes of finding a good direction of improvement for the higher-fidelity model. The combination of high-fidelity, low-fidelity and surrogate models determine the model management framework, which aims at maximizing the use of lower-fidelity and surrogate models in iterative procedures with occasional, but systematic, recourse to higher-fidelity, and more expensive models for monitoring the progress of the optimization (Alexandrov et al. 1999, and Vitali et al. 2002).

Starting from the high-fidelity models, three approaches are described in the following sections: (i) surrogates directly replace actual models; (ii) variable-fidelity models, i.e. aiming to optimize high-fidelity but expensive models, the optimization is conducted with cheaper low-fidelity models corrected by surrogates; and (iii) simultaneous use of actual high-fidelity and surrogate models (during the optimization, surrogate models can eventually be updated).

4.2. First Approach: Direct Use of Surrogate Models

This is the simplest case of direct substitution of the high-fidelity models by their surrogate counterparts. When using this approach it pays the use of multiple surrogates instead of a single one for reasons such as the most accurate surrogate does not guarantee the global optimum, and fitting many surrogates and repeating optimizations is cheap as compared to the cost of the simulations. Figure 4.1 shows how this approach works. Most of the steps were already discussed in Chapter I and II, however here both full cycles of surrogate modeling and optimization are combined together. This is not an automatic process since steps involving (i) assessment of quality of fitting of the set of surrogates and selection of a reduced subset; (ii) assessment of the quality of the results; and (iii) redefinition of the design space and sampled points usually requires external intervention. As a consequence, the optimal solution is obtained gradually. At the end of each iteration, candidate solutions can be used in the decision of narrowing the design space, on the re-fitting of the set of surrogates or even as extra points for assessment of the quality of the surrogates.

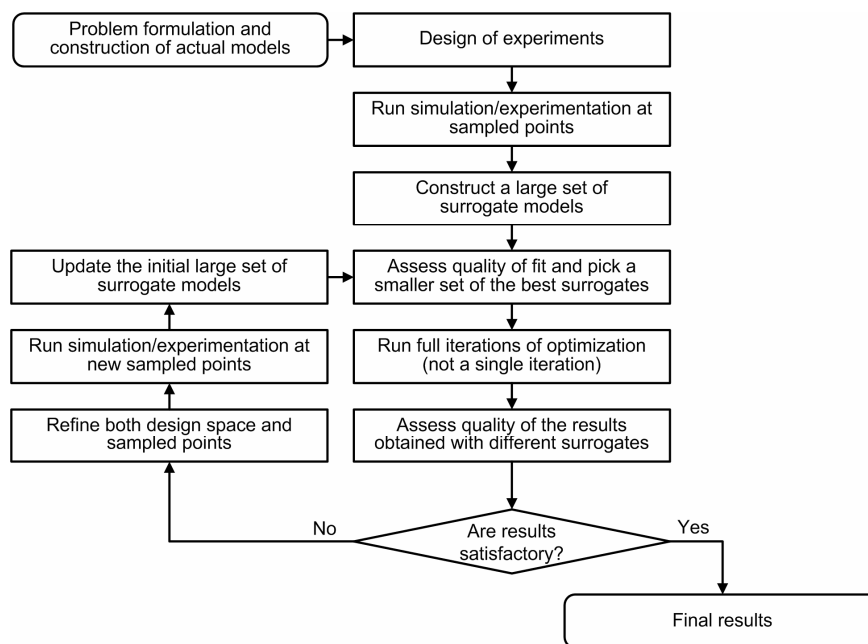


Figure 4.1. Basic scheme for direct use of surrogate models.

If multiple surrogates are used instead of the high-fidelity models, one could ask why not use classical gradient-based algorithms to solve the optimization problem. The answer is that surrogate modeling may also lead to local optima. In a scenario of easy function evaluations, heuristic algorithms may be a more suitable alternative since they present a trend to finding the global, or near global, solution. The bottom line could be the use of both

heuristic and gradient-based algorithms in a cascade-type scheme in which the heuristic algorithm generate proper initial guesses for the gradient-based algorithm.

4.3. Second Approach: Use of Different Levels of Fidelity

This is the case in which surrogates are used to map the responses obtained from the low-fidelity to the high-fidelity space (i.e., they correct the responses obtained by using low-fidelity models). This allows the use of faster low-fidelity models during the optimization; while a reduced set of expensive high-fidelity models are run “off-line.” A common approach is to use the ratio or difference between the low-fidelity and high-fidelity models at one or several points in order to correct the low-fidelity model at other points (Alexandrov et al. 1999, Vitali et al. 2002, Gano et al. 2004, and Marduel et al. 2006). Figure 4.2 shows a possible flowchart for this approach.

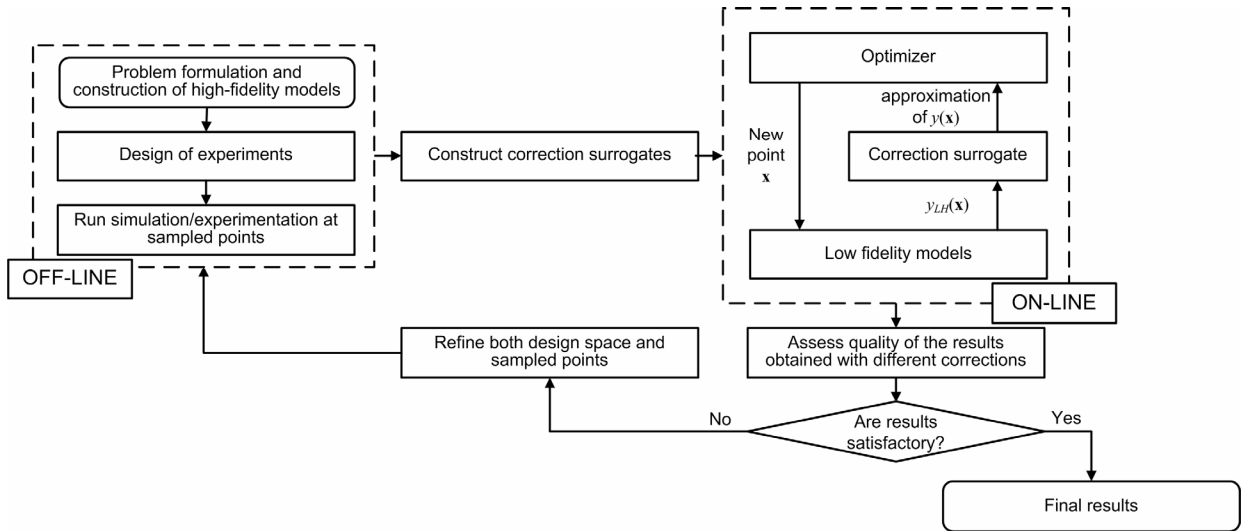


Figure 4.2. Basic scheme for variable-fidelity framework.

Correction surrogates couple high-fidelity and low-fidelity methods of analysis at a number of points in the design domain. Here, two ways of coupling the ratio and difference methods of analysis are presented. The process starts with high and low-fidelity computation of the response of interest, y , at different p points. At these points, the ratio β :

$$\beta = \frac{y_{HF}}{y_{LF}}, \quad (4.1)$$

and the difference δ :

$$\delta = y_{HF} - y_{LF}, \quad (4.2)$$

are computed. The subscripts HF and LF in Eqs. (4.1) and (4.2) indicate the value of y obtained from high-fidelity and low-fidelity models, respectively.

Equation (4.1) may lead to a conditioning problem of β , when y_{LF} tends to zero. Marduel et al. (2006) show that adding a user-defined constant, K , sufficiently large, can prevent ill-conditioning. β is then defined by:

$$\beta = \frac{y_{HF} + K}{y_{LF} + K} , \quad (4.3)$$

The next step is to build the surrogate models $\hat{\beta}(\mathbf{x})$ and $\hat{\delta}(\mathbf{x})$ for β and δ , respectively. Here, there is no surrogate model to be preferably used. Instead, as in the previous approach, a set of different surrogates can also be used as a way to prevent against bad fitted surrogates.

The variable-fidelity approximation to y at any other point, $\mathbf{x}$, is obtained from a low-fidelity analysis as:

$$y(\mathbf{x}) \cong y_{\beta}(\mathbf{x}) \quad \text{or} \quad y(\mathbf{x}) \cong y_{\delta}(\mathbf{x}) , \quad (4.4)$$

$$\begin{aligned} y_{\beta}(\mathbf{x}) &= \hat{\beta}(\mathbf{x}) y_{LF}(\mathbf{x}) , \\ y_{\delta}(\mathbf{x}) &= \hat{\delta}(\mathbf{x}) + y_{LF}(\mathbf{x}) . \end{aligned} \quad (4.5)$$

In order to avoid the need to interface the low-fidelity analysis with the optimization software, Vitali et al. (2002) also propose the replacement of the low-fidelity model by its surrogate counterpart. Instead of calling the low-fidelity analysis program every time a value of $y(\mathbf{x})$ is needed, the optimizer calls for the surrogate model. This way:

$$y(\mathbf{x}) \cong \hat{\beta}(\mathbf{x}) \hat{y}_{LF}(\mathbf{x}) \quad \text{or} \quad y(\mathbf{x}) \cong \hat{\delta}(\mathbf{x}) + \hat{y}_{LF}(\mathbf{x}) . \quad (4.6)$$

Vitali et al. (2002) go even further by proposing that in addition to those given by Eqs. (4.4) and (4.6), two other approximations of $y(\mathbf{x})$ can be obtained by correcting the low-fidelity values $y_{LF}(\mathbf{x})$ by $\hat{\beta}(\mathbf{x})$ and/or $\hat{\delta}(\mathbf{x})$, then fitting another surrogate of $y_{\beta}(\mathbf{x})$ and/or $y_{\delta}(\mathbf{x})$, i. e.:

$$y(\mathbf{x}) \cong \hat{y}_{\beta}(\mathbf{x}) \quad \text{or} \quad y(\mathbf{x}) \cong \hat{y}_{\delta}(\mathbf{x}) , \quad (4.7)$$

At the end, the cost of running $y_{LF}(\mathbf{x})$ may be used to decide whether use computations following Eq. (4.4) or use only Eqs.(4.6) and (4.7). Once again, the use of multiple variable-fidelity approximations to $y(\mathbf{x})$ is encouraged as a way to generate a diverse set of candidate solutions for the optimization problem. In this sense, if Eqs. (4.4) to (4.7) are used, at the end of the optimization a set of candidate solutions must be evaluated, instead of a single one.

4.4. Third Approach: Coupling Actual High-Fidelity and Surrogate Models

This is the case in which both high-fidelity and surrogate models are used during optimization. High fidelity models can be used both to check the accuracy and to update the surrogate models. There are a number of studies reporting the use of different surrogates and different strategies for coupling them with optimization algorithms. Liang et al. (2000) proposed a strategy for coupling heuristic algorithms with local search and quadratic response surface methods. However, when working with multimodal problems the accuracy of quadratic models can become questionable. Jin et al. (2002) presented a framework for coupling heuristic algorithms and neural-network-based surrogate models. This approach uses both the expensive and approximate models throughout the search, with an empirical criterion to decide the frequency at which each model should be used. Ong et al. (2003) employed both high-fidelity and support vector regression models (with linear-splines as kernel function) during the optimization. They propose an approach in which the local search is conducted with the surrogate models and the global search is performed with the actual high-fidelity models. Zhou et al. (2007) proposed a framework that uses different surrogate models during local and global search. The framework employs a data-parallel Gaussian process based global surrogate model to filter the heuristic algorithm population so that promising individuals are retained. Subsequently, these potential individuals undergo a trust-region gradient-based search strategy that employs radial basis function local surrogate models to accelerate convergence.

While several researchers have primarily been concerned with the choice among different surrogates, there has been virtually no work regarding the use of multiple surrogates. Thus, in this doctoral research, the following aspects are explored:

1. A large set of different surrogates is used during both off-line and on-line phases of the optimization.
2. Update of the surrogates during the on-line phase of the optimization through a pre-defined number of extra actual function evaluations (high-fidelity analysis).

3. *BestPRESS* (i.e. surrogate with the smallest *PRESS* value of the set) is used for global search. Since *PRESS* is used as an estimator of the *RMSE* (*RMS* error); the idea is to pick up the surrogate with best global approximation capabilities.

A scheme with two variations is proposed, as seen in Figure 4.3. There is a common off-line phase as well as a common on-line phase (see Figure 4.3-(a) and (b)). During the off-line phase, the basic steps of the surrogate modeling (discussed in Chapter II) are performed. However, instead of the traditional use of a single surrogate model, here a set of different basic surrogates are generated. In the sequence, parameters for the optimization are defined. Other than parameters of the optimization method (such as population size, number of iterations, etc.), the number of extra actual function evaluations, N_{mxp} , must be defined. N_{mxp} acts like the computational budget used during the on-life phase for updating the surrogates. Using *BestPRESS* as objective function, heuristic optimization algorithms are in charge of generating candidate solutions considered to be added the set of initial data points. If a candidate solution is not in the design matrix already, the high-fidelity analysis is performed. It is worth remembering that especially with discrete variables, different runs of the heuristic algorithm may lead to the same candidate solution. Thus, it is suggested that this loop run for up to $1.5 \times N_{mxp}$ iterations, instead of just for N_{mxp} . The two proposed variations differ on the actual evaluation phase. Figure 4.4-(a) illustrates the first and simplest approach, where the candidate solutions are added to the design matrix, $\mathbf{X}$ (and its respective actual function value to the $\mathbf{Y}$ vector). At the end of the loop, the resulting design matrix, $\mathbf{X}_{new}$ (and the respective $\mathbf{Y}_{new}$) is used for updating the set of basic surrogates. The update of the surrogates is performed just once, right before the final run of the heuristic algorithm, which finally generates the final candidate solution, $\mathbf{x}_{optm}$. The second approach updates the surrogates every time a new point is added to the design matrix, as shown in Figure 4.4-(b). It means that *BestPRESS*, used as objective function, may change in both shape and identity. It is easy to see that while the first approach is faster, the second one may benefit from the constant update of the surrogates. The downside of the second approach is the iterative computation of *PRESS* errors. In high dimensions, this may be an issue and the use of alternative *PRESS* approaches as the k -fold strategy may be considered (as discussed in Chapter II). In this case, instead of computing *PRESS* errors through the leave-one-out strategy every time a new point is added to the design matrix, $\mathbf{X}_{new}$, *PRESS* is performed using the k -fold strategy with pre-defined number of points and respective values of k . As an illustration of this question, consider the case of 12-point initial design matrix in a 2-dimensional space, and $N_{mxp} = 6$. Most probably, there are no

computational issues in performing *PRESS* calculations when the number of points in $\mathbf{X}_{new}$ is equal to 13, 14, 15, 16, 17, and 18. On the other hand, consider a 56-point initial design matrix in a 6-dimensional space, and $N_{mxp} = 14$. Here, the computation of *PRESS* may be expensive and instead of employing the leave-one-out strategy, the k -fold strategy may be used when, for example, the number of points in $\mathbf{X}_{new}$ reaches 60, 63, 66, and 68, with k equals to 20, 21, 22, and 17; respectively. For the sake of simplicity, the flowchart of Figure 4.3-(b) and Figure 4.4-(b) show just the leave-one-out implementation (the k -fold variation is left as an exercise for the reader). The SIMPLE Optimization toolbox of Viana and Steffen Jr (2008) has all previously discussed functionalities already implemented.

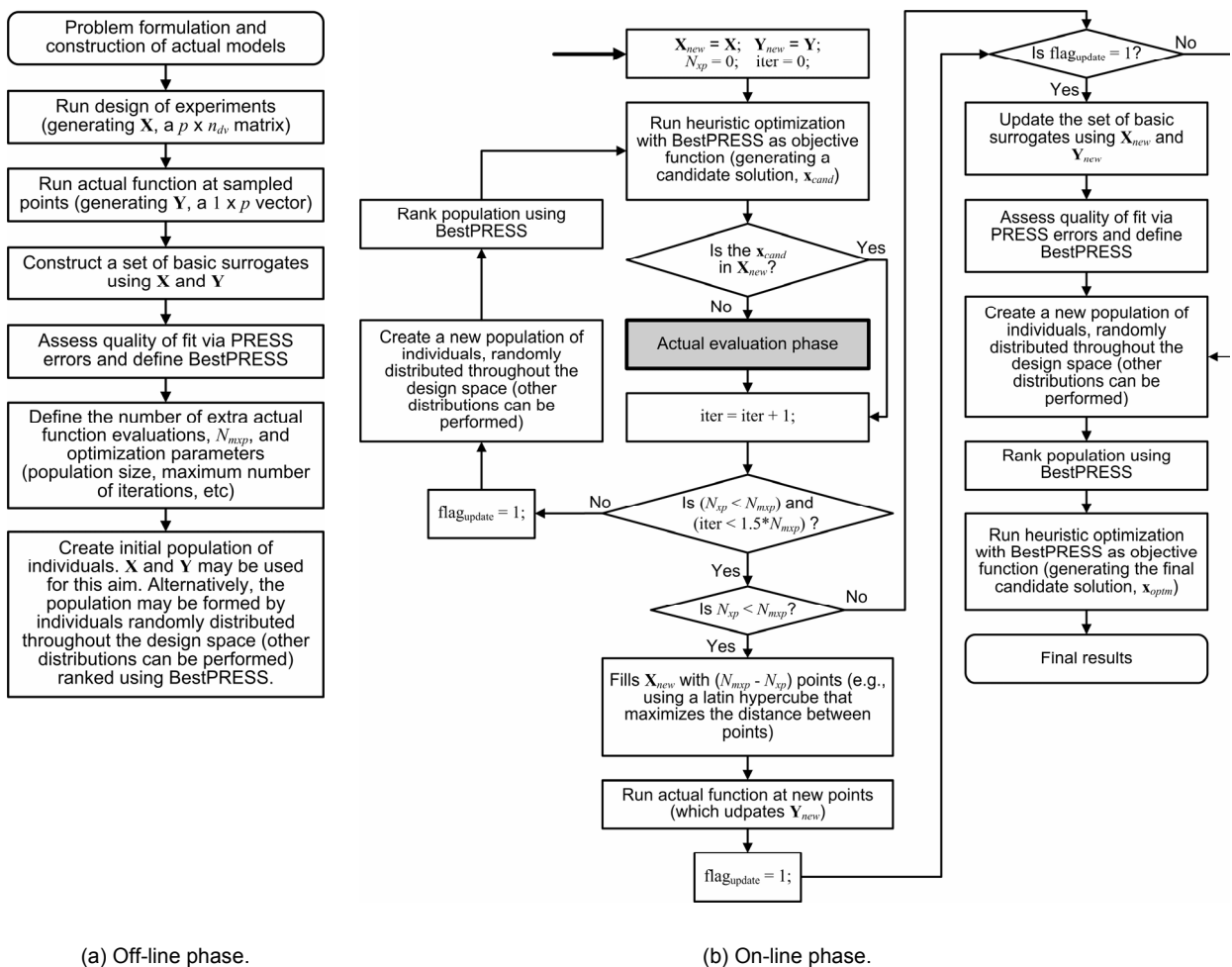
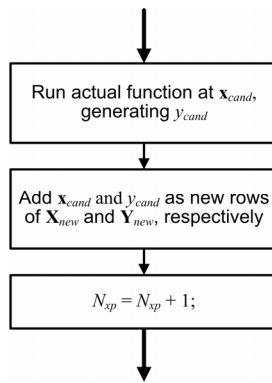
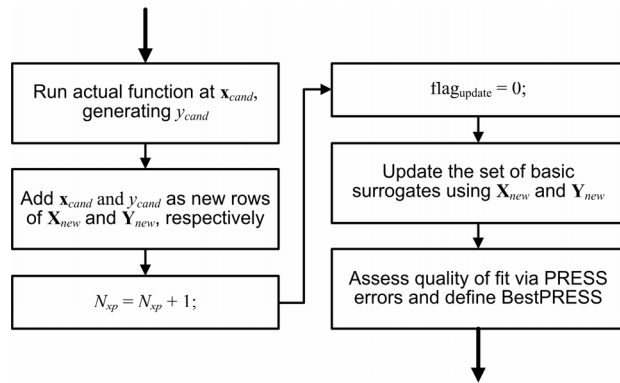


Figure 4.3. Basic scheme for the coupling of actual high-fidelity and surrogate models.



(a) Without update of surrogates.



(b) With update of surrogates.

Figure 4.4. Different alternatives for the “actual evaluation phase” of Figure 4.3-(b).

CHAPTER V

APPLICATIONS

The previous chapters showed the basis and theoretical contributions of this doctoral research. This chapter is devoted to the numerical applications. Ant Colony Optimization, Differential Evolution, Genetic Algorithms, Particle Swarm Optimization and LifeCycle Model were employed to solve the following list of direct/inverse optimization problems:

- Direct problems: (a) Vehicular three-dimensional structure design optimization; and (b) Optimization of aircraft structural components by using heuristic algorithms and multi-fidelity approximations.
- Coupling heuristic optimization and high-fidelity analysis to improve surrogate models in the region of the optima.
- Inverse problems: (a) Identification of a non-linear landing gear model using heuristic optimization; and (b) Aircraft longitudinal stability and control derivatives identification by using LifeCycle and a local optimization algorithm.
- Enhanced Stochastic Evolutionary Algorithm used to solve the combinatorial optimization of the optimal Latin hypercube design of experiments.

5.1. Direct Problems

5.1.1. Three-dimensional Vehicular Structure Design Optimization

This application explores discrete-continuous, multi-objective and constrained optimization using heuristic optimization methods. It consists in the design optimization of a vehicular three-dimensional structure (space frame type) by means of a set of nature-inspired optimization algorithms, namely ACO, GA, and PSO. The structure is used as a chassis of a high performance roadster vehicle. In general, for this type of vehicle, a chassis must be as rigid and lightweight as possible in order to fulfill drivability (the degree of smoothness and steadiness of acceleration of a car) and maneuverability (the quality of being capable of maneuvering or changing position) requirements. This application was fully reported in Viana et al. (2007b) and Oliveira et al. (2007).

1. Baseline Design Definition and Modeling

The use of three-dimensional vehicular structures became popular since 1960s, due to the interest in designing high performance race cars (Aird, 1997). Figure 5.1 shows one of the most common structures for this type of chassis.

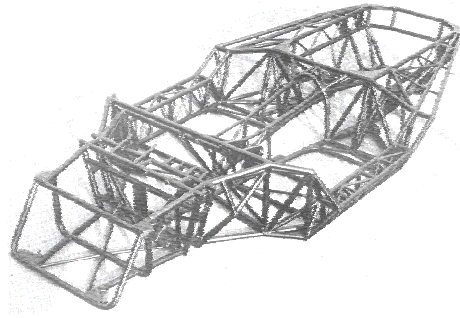


Figure 5.1. Space frame chassis in three-dimensional arrangement.

According to the literature (Thompson et al., 1998 and Aird, 1997), for design purposes the most effective strategy should consider both the mass reduction and the increase of the torsional stiffness. Translating this to the design level, the engineer must choose the material (by defining the material density, Poisson's ratio, and the Young's modulus) and design the proper geometry of the structure (by defining the total mass and moment of inertia). Even though steel alloy and aluminum alloy are the most attractive materials for the chassis construction, 1020 carbon steel was employed here as a low cost solution. Figure 5.2 illustrates the baseline design for a two-passenger vehicle as obtained by consulting the literature about high performance roadster vehicles (Thompson et al., 1998; and Aird, 1997). The lateral view shows indications of the suspensions anchorage points. Since lack of roof and the presence of side doors can make this structure flexible, left and right trusses were placed together with diagonal bars in the floor-plan (minimizing buckling tendencies). Table 5.1 gives more details about this initial design configuration. Regarding the finite element modeling of the structure, while the use of more sophisticated elements may be more accurate, they are also more expensive from the computational viewpoint, and thus, not convenient within the optimization framework. In this case study, bar elements were used as a compromise solution between accuracy and computational cost. Each element has twelve degrees of freedom (two nodes per element): translations along x, y and z nodal directions and rotations with respect to x, y and z nodal axes. The baseline design has 19,167 degrees of freedom, mass of 139.52 [Kg] and torsional stiffness of 1728.1 $[Kgf \cdot m/deg]$.

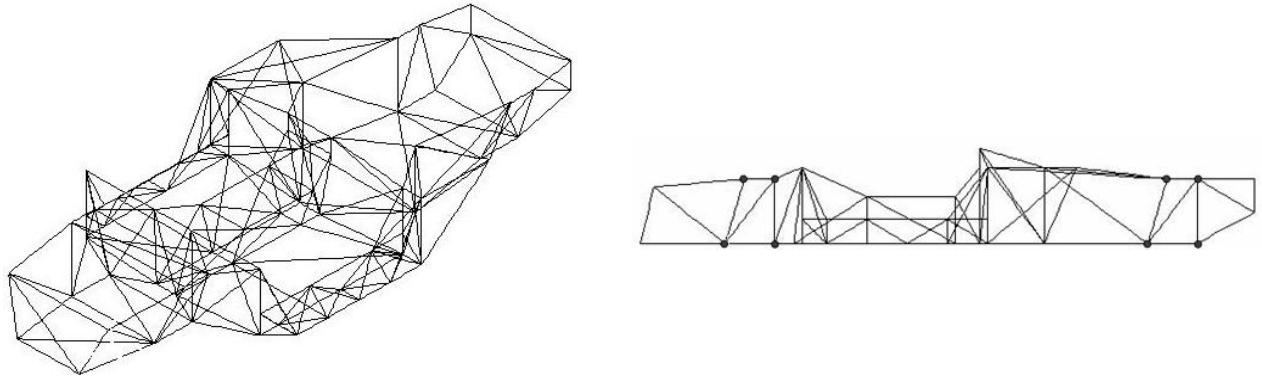


Figure 5.2. Baseline design of the space-frame chassis (lateral view indicates points of suspensions anchorage).

Table 5.1. Baseline details.

General		
Mass: 139.52 <i>Kg</i>	Torsional stiffness: 1,728.1 <i>Kgf · m/deg</i>	Initial geometry: circular tubes with $50.8 \times 10^{-3} \text{ [m]}$ of external diameter and $1 \times 10^{-3} \text{ [m]}$ of wall thickness
Wheelbase: 2.62 <i>m</i>		Front and rear tracks: 1.475 <i>m</i>
Front suspension type: short arm		Rear suspension type: MacPherson with aggregate
Material (carbon steel)		
Young's modulus: $2.1 \times 10^{11} \text{ N/m}^2$	Poisson ratio: 0.3	Density: 7850 <i>Kg/m</i> ³
FE model		
Element type: BEAM04	NDOF per element: 12	NDOF of the model: 18,675

The torsional stiffness is numerically obtained through a static analysis of the structure. Figure 5.3 illustrates the boundary conditions for this aim. The torsional stiffness is then obtained by using the information about the stress and strain, as detailed in Aird (1997). The commercial finite element package ANSYS (Ansys Contributors, 2006) is used to run the finite element analysis. More details about the torsional stiffness computation for this specific application can be found in Oliveira (2007).

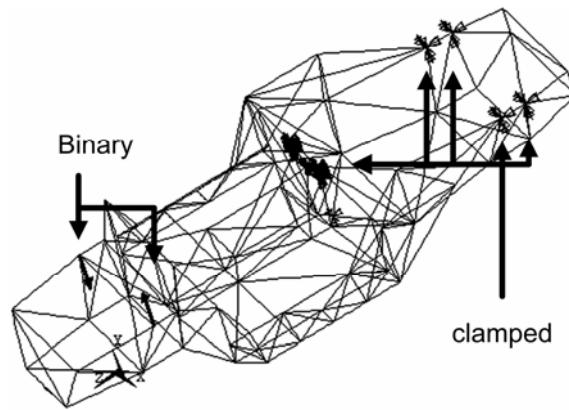


Figure 5.3. Boundary conditions for computing torsional stiffness.

2. Optimization Problem Formulation

Figure 5.4 shows a possible way of incorporating numerical optimization in the design cycle. Modeling of the structure and analysis of design requirements are steps of the “problem definition and formulation.” After this first stage, a baseline design is proposed. After each “optimization” step, the candidate solution is revised. If either the constraints were not satisfied or it is possible to achieve some extra improvement, the candidate solution is redefined as a baseline for a new iteration of the design cycle. This loop continues until the candidate solution satisfies the proposed goals. As a result of the optimization, it may be observed not only changes in the design in the sense of the nodal coordinates, but also changes in the finite element model. A candidate solution may lead the designer to include or eliminate elements in the structure.

The formulated optimization problem deals with two responses obtained from the finite element model, namely, the mass and the torsional stiffness (in a more realistic design scenario, it would be necessary to include other factors such as dynamic responses, manufacture constraints, etc.). A total number of 32 nodal coordinates on the central part of the chassis are assumed as continuous design variables and the external diameter of the tube is a discrete variable. The baseline design is placed at the center of the design space in such a way that and each node can be shifted to a position that has coordinates varying from 75% to 125% of the baseline counterparts. Figure 5.5 illustrates where the 32 nodal coordinates take place in the baseline design. The external diameter of the tube can assume values shown in Table 5.2. For all these diameter values, the thickness ranges from $1.06 \times 10^{-3} m$ to $3.00 \times 10^{-3} m$ (all values were obtained from commercial steel tubes found in the market). This way, the optimization is defined as a problem of 33 design variables (32 continuous and 1 discrete).