

Timo Aittokoski

On Optimization of Simulation Based Design

JYVÄSKYLÄ LICENTIATE THESES IN COMPUTING 8

Timo Aittokoski

On Optimization of
Simulation Based Design



UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2007

On Optimization of Simulation Based Design

JYVÄSKYLÄ LICENTIATE THESES IN COMPUTING 8

Timo Aittokoski

On Optimization of
Simulation Based Design



UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2007

Editor
Timo Männikkö
Department of Mathematical Information Technology, University of Jyväskylä

ISBN 978-951-39-2865-0 (PDF), 978-951-39-2788-2 (nid.)

ISSN 1795-9713

Copyright © 2007, by University of Jyväskylä

Jyväskylä University Printing House, Jyväskylä 2007

ABSTRACT

Aittokoski, Timo

On optimization of simulation based design

Jyväskylä: University of Jyväskylä, 2007, 110 p.

(Jyväskylä Licentiate Theses in Computing

ISSN 1795-9713; 8)

ISBN 978-951-39-2788-2

Finnish summary

In many industrial processes it is imperative to be able to control features (quality, production cost, strength, etc.) of the end product. Processes can be simulated using specific mathematical models, upon which the effect of different input variable values of the process can be tested to produce desired end result. Often input variable values are adjusted by trial-and-error, based on the designer's expertise, producing arbitrary results varying from poor to excellent.

With simulation based optimization, designer no longer adjusts single design variable values, but rather he/she describes the desired end result with a higher abstraction level, and an appropriate optimization algorithm finds proper values for any design variable.

In this study, properties (such as maximum power, efficiency and suitability of gearing to engine properties) of internal combustion engines are optimized. Our problem setup, as several other real life engineering problems, places some special requirements for the optimization system, and we consider three important characteristics: efficiency (generally, running the simulator is computationally expensive) in terms of objective function evaluations, global search instead of local one (to avoid local minima often present because of high nonlinearity), and treating multiple objectives simultaneously in the problem. One more characteristic of our problem is having no gradient information available. In this study we give some perspectives to our problem and present cost effective ways to solve it.

In our study, we want to solve efficiently (using a minimal number of objective function evaluations) and in a global and inherently multiobjective manner a complex task with a practical value. We alter dimensions and the shape of the exhaust pipe of a 2-stroke engine, and thus affect power output characteristics of the engine. Our objective functions are more complicated than in some of the previous studies, and we address the very important problem (from the perspective of vehicle performance) of fitting the engine properties to gearing and transmission. This study also adheres to an important topic often faced in real world applications, namely solving optimization tasks in case of several conflicting objectives via the use of scalarization methods.

Keywords: optimization, multiobjective, global, efficient, simulation, scalarization, reference point

Author

Timo Aittokoski
Department of Mathematical Information Technology
University of Jyväskylä
Finland
timaitt@jyu.fi

Supervisor

Professor Kaisa Miettinen
Helsinki School of Economics
Finland
kaisa.miettinen@hse.fi

Examiners

Professor Jouni Lampinen
Department of Information Technology
Lappeenranta University of Technology
Finland
jlampine@lut.fi

Professor Henrik Saxén
Faculty of Technology, Heat Engineering Laboratory
Åbo Akademi University
Finland
hsaxen@abo.fi

ACKNOWLEDGEMENTS

This work would not have been possible without the encouragement, expertise and insights of my supervisor, professor Kaisa Miettinen. For technical issues related to the integration of NIMBUS software to the optimization system of this study I am indebted to Vesa Ojalehto. For interest to my work, and for evaluating my thesis, I would like to thank professor Jouni Lampinen and professor Henrik Saxén. I would like to thank doctor M.M. Ali for providing the source codes for the CRS2 and CRS4 algorithms. My lifetime fascination with engine design and improvement originates from my boyhood years and writings of S. Tiittanen and late and widely appreciated Gordon Jennings.

This study was financially supported by COMAS graduate school, Academy of Finland (grant number 104641), Ellen & Artturi Nyyssönen Foundation and Jenny & Antti Wihuri Foundation.

Finally, I would like to thank my parents for creation of my essence, my loved ones, friends and colleagues for being there, and last but not least, Fourmyle of Ceres for the source of amaranthine inspiration.

Jyväskylä 15th November 2006

Timo Aittokoski

LIST OF FIGURES

FIGURE 1	Overview of the modules of a heterogenous optimization system.	13
FIGURE 2	An example of a non-convex function with two local and one global minima.	19
FIGURE 3	Nelder-Mead simplices after a reflection and an expansion step. The original simplex is shown with a dashed line. Figures from [39].	24
FIGURE 4	Nelder-Mead simplices after an outside contraction, an inside contraction, and a shrink. The original simplex is shown with a dashed line. Figures from [39].	24
FIGURE 5	Niederreiter random sampling using 1000 points in a two dimensional case.	32
FIGURE 6	Uncertainty about the function's value at a point (e.g. $x = 8$) can be treated as if the value there were a realization of a normal random variable with mean and standard deviation given by the DACE predictor and its standard error. Figure from [32].	36
FIGURE 7	An example of SCGA crossover in two dimensions. Figure from [26].	38
FIGURE 8	Steps of the main loop of the DE algorithm. Notice that a chromosome equals to an individual, and a gene equals to a design variable in this figure. Figure from [8].	42
FIGURE 9	Pareto optimal sets, ideal and nadir objective vectors in two cases. Figures from [43].	45
FIGURE 10	Sample screenshot of NIMBUS software.	50
FIGURE 11	A flowchart of the NIMBUS algorithm. Figure from [22].	51
FIGURE 12	4-stroke engine cycle. Figures from [34].	54
FIGURE 13	2-stroke engine cycle. Figures from [34].	55
FIGURE 14	Basic pipe shape consisting of four sections: diffuser (aggregated with header pipe), belly, baffle cone and stinger.	59
FIGURE 15	Waves and pressure distribution inside the exhaust pipe. Figures from [62].	60
FIGURE 16	Simulated and experimental dynamometer results for the Parilla engine. Crosses: simulated, solid line: experiment. Figure based on [41].	66
FIGURE 17	Examples of exhaust pipe shape produced using the Blair model. Shape is presented to simulator using seven pieces, L1-L7.	69
FIGURE 18	Two examples of exhaust pipe shapes produced using horn model. Shape is presented to simulator using seven pieces.	72
FIGURE 19	Inflexibility of the horn model. This shape of the exhaust pipe is not reachable using horn model.	73
FIGURE 20	The shape envelope for the FFS model and one resulting shape. The shape is presented to the simulator using ten sections.	74

FIGURE 21	Bezier curve with two adjustable control points inside their respective boundary boxes.	77
FIGURE 22	Example of smooth exhaust pipe shape created using Bezier model.	78
FIGURE 23	Example of flexibility of Bezier model: approximation of horn shape.	78
FIGURE 24	Three possible properties for optimization: maximum power, integrated power and location emphasized integrated power. .	79
FIGURE 25	Ultimate driving force curve (UDFC) and torque cascades. Figure from [13].	83
FIGURE 26	Examples of actual pipe shapes using all four models in a CRS2 run. On top of each shell, the objective function value and the respective evaluation number in parentheses.	89
FIGURE 27	Examples of pipe shapes and corresponding power curves for all four cases.	96
FIGURE 28	Power curve and corresponding pipe shape of the final interactive solution (25.3, 0.87, 11.2).	100

LIST OF TABLES

TABLE 1	Typical classification scheme based on the nature of the problem functions as given in [19].	18
TABLE 2	Design variables and their bounds for Blair model.	68
TABLE 3	Design variables and their bounds.	71
TABLE 4	Design variables and their bounds for FFS model.	74
TABLE 5	Design variables and their bounds for the Bezier model.	76
TABLE 6	Results using Niederreiter random sampling. The best function value gained before a given number of evaluations in each cell is on top, and respective evaluation number is below in parenthesis.	88
TABLE 7	Results using CRS2 and population size $5n$. The best function value gained before a given number of evaluations in each cell is on top, and respective evaluation number is below in parenthesis.	88
TABLE 8	Comparison of different global optimization algorithms. Best function values gained before given number of evaluations in each cell are on top, and respective evaluation number is below in parenthesis.	91
TABLE 9	Ordinal comparison of different global optimization algorithms. Algorithms for each function evaluation level are ordered columnwise.	92
TABLE 10	Multiobjective optimization results and weights using neutral compromise solution (NCS) and weighted and scaled Chebychev (WSC) scalarization functions.	93

TABLE 11	Reference points and solutions with different scalarization methods.	95
TABLE 12	Effect of the number of function evaluations to solution. Reference point: power = 22, coverage = 1.0, bmep = 10. . .	97
TABLE 13	Effect of the number of function evaluations to solution. Reference point: power = 25, coverage = 0.88, bmep = 10. . .	97
TABLE 14	Interactive solution process.	99

CONTENTS

ABSTRACT

ACKNOWLEDGEMENTS

LIST OF FIGURES AND TABLES

CONTENTS

1	INTRODUCTION	11
2	OPTIMIZATION AND SOME METHODS	17
2.1	Local single objective optimization.....	21
2.1.1	Nelder-Mead simplex method	22
2.2	Global single objective optimization.....	23
2.2.1	Classification of methods.....	25
2.2.2	Niederreiter random sampling.....	32
2.2.3	Controlled random search	33
2.2.4	Efficient Global Optimization.....	34
2.2.5	Simplex Coding Genetic Algorithm.....	37
2.2.6	Simulated Annealing Heuristic Pattern Search, SAHPS	38
2.2.7	Differential Evolution.....	39
2.3	Multiobjective optimization.....	42
2.3.1	Scalarization methods	44
3	BASIC OPERATION OF INTERNAL COMBUSTION ENGINES	53
3.1	4-stroke engine cycle.....	53
3.2	2-stroke engine cycle.....	55
3.3	Focus of this study.....	56
3.4	How external ducting works	58
4	OPTIMIZATION TASK OF AN INTERNAL COMBUSTION ENGINE .	63
4.1	Framework of the optimization system.....	63
4.2	The engine simulator	64
4.3	Modeling the engine configuration.....	67
4.3.1	Blair model	68
4.3.2	Horn model	70
4.3.3	Free Form Shape model.....	73
4.3.4	Bezier model	74
4.4	Defining the objective function	77
4.4.1	On the implementation	82
4.5	Formulation of an optimization problem	84
4.5.1	Single objective case.....	84
4.5.2	Multiobjective case	85
5	RESULTS	86
5.1	Single objective case.....	87
5.1.1	Comparison of different shape models.....	87

5.1.2	Comparison of different optimization methods	90
5.2	Multiobjective case	93
5.2.1	Comparison of different scalarization functions.....	93
5.2.2	Number of function evaluations and solution quality.....	96
5.2.3	Interactive solution process using NIMBUS.....	98
6	DISCUSSION AND CONCLUSIONS	101
	REFERENCES	105
	YHTEENVETO (FINNISH SUMMARY)	110

1 INTRODUCTION

Behavior of many real-world systems and devices can often be expressed with mathematical models. If such a model is implemented using a computer, it can simulate the behavior of the system, and the resulting software is called a simulator. The simulator evaluates behavior of the system using specific values for the input variables and calculates corresponding values for the output variables. For a review of simulation methodology, we refer to [5] and [20]. In industry, simulators are often used instead of small scale models, extensive testing, building the device, or creating the system itself. There are certain reasons to this; ever increasing speed of computing makes the use of simulators more appealing in terms of invested time and money. Running the simulator is usually far cheaper, faster and in some cases also safer than implementing the prototype of a real system or device. Simulation, among other things, allows the designer to try several different designs and to explore advantages and disadvantages of each design.

Different simulators are widely used in industry, ranging from simulation of chemical processes of the paper making line [11] to aircraft flows [66] and internal combustion engine processes [41]. Although the benefits of simulator usage are undisputed, some problems still remain. A simulator merely mimics the behavior of some system, and as such, it gives no information of how exactly that behavior can be improved. We refer to the problem of selection of values for the parameters and variables for the simulation as design problem, which remains in human domain. One of the most prominent problems with simulators is how to manage numerous input variables and their delicate interactions. It is extremely difficult to gain results that, for example, a designer is aiming for by manually adjusting the design variable values. This is where optimization steps in.

Optimization process can be described as a routine where the aim is to find the best possible value for a given quantity by manipulating design (input) variable values. This quantity can be referred to as an objective function, and the optimization routine can be seen as a tool for finding the lowest / highest values of the objective function. Optimization algorithms automatize the search procedure using different means in judging what values for the design variables should produce good objective function values. The designer needs only con-

struct an objective function (which reflects the goodness of a particular design) to meet his/her needs and define the boundaries for the search space, i.e. ranges and other possible constraints for design variables. There are lots of different optimization algorithms available for different types of optimization problems. A more thorough discussion of these can be found for example in [7], [19] and [43].

In real life the designer is rarely so lucky that he/she needs to deal with only a single objective function. Often there are several objectives that should be simultaneously optimized and that usually are conflicting. For example, in the field of engine design, the designer may want to create a powerful engine, while (s)he must keep its fuel consumption as low as possible. These two objectives are obviously conflicting. Problems with multiple objectives are called multiobjective optimization problems. The usual way to solve a multiobjective optimization problem is to convert multiple objective functions into a single objective function using a so-called scalarization function, and finding the optimum for that function by using methods of single objective optimization [43]. Also, especially in the field of engineering, multiobjective optimization problems are often solved using multiobjective evolutionary algorithms.

Simulation based optimization, where the objective function values are derived from the output files of the simulator run, places some special requirements to the optimization algorithm. Objective function is often a so-called black box function (the internal mechanism and structure of the function is unknown, and only input and output characteristics can be employed). The objective function value produced by the simulation software is often the result of a complex sequence of calculations. Due to the complex nature of calculations, relationships between the decision variables and output variables do not necessarily exist in a closed form. Also, derivatives which are commonly used to guide the optimization process are usually unavailable. Derivatives can be estimated using finite differences, but this, in turn, would increase computational load. Objective function evaluations can be extremely expensive computationally, because simulation of, for example, some complex flow model may take hours or days in the worst case. Besides, the objective function itself may have lots of locally optimal values. All the facts above suggest that we are not able to use traditional optimization algorithms such as the steepest descent (gradient) method or the sequential quadratic programming method (SQP), which require gradient information and are local search methods. In other words, these methods will find the nearest optimum from the starting point, which could be quite far from the real global optimum. Facts above suggest that we should use the efficient (in terms of objective function evaluations) global optimization algorithm. This would help us to tackle problems caused by several local optima and the execution time of the optimization algorithm.

There is one more difficulty worth mentioning in simulation based optimization. It is reasonable to distinguish two main parts of the optimization system, namely the optimizer (the optimization algorithm itself) and the part which calculates values for the objective function, in this case, the simulator. With analytic problems, which can be expressed in a closed form, one can often use one

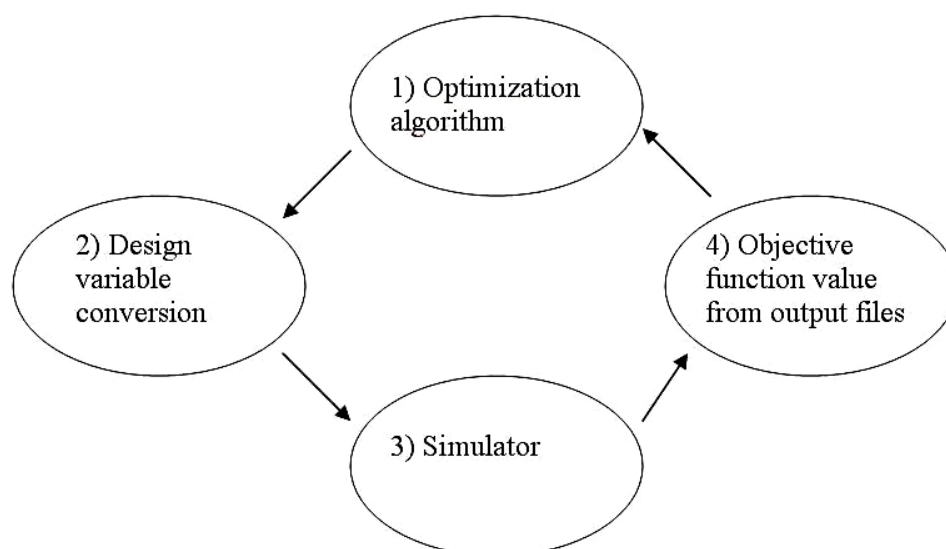


FIGURE 1 Overview of the modules of a heterogeneous optimization system.

homogenous system, i.e. optimization algorithm and objective function calculations are implemented using the same programming language, and they can even reside in the same executable file. While speaking of simulation based optimization the whole system becomes easily very heterogeneous and can consist of several modules implemented using different tools and languages. The whole system can be divided to four separate modules as depicted in Figure 1: the optimizer (1), input interface for the simulator (2), the simulator (3) and the output interface (4) from the simulator to optimizer.

First there is the optimization algorithm module, which guides the whole optimization procedure by deciding what design variable values should be passed forward to the next module. The second module receives the design variable values and generates from them suitable input configuration files for the third module, the simulator. For example, for the engine simulator the engine design variables must be converted to depict the whole internal engine geometry, or some more specific part of the engine which is to be optimized. To the optimization system, the simulator is seen as a "black box": it merely receives design variable values and produces simulation output files which contain detailed information on how the simulated system performed with a given set of design variable values. The simulator itself may be an arbitrarily complex set of calculations and even consist of several software modules. The execution time for a single simulation run may vary from milliseconds to weeks. In the fourth module, the output files of the simulator must be handled to constitute a value for the objective function, which is finally passed back to the optimization algorithm. Then the loop starts all over again, and the optimization algorithm uses information of the objective function value to decide upon new values for the design variables.

As can be seen from Figure 1, the optimization system may be very het-

erogenous: each of the four modules may be implemented using different programming languages and platforms and they may even run on physically separate computers. Regardless of the implementation and structure of the modules they must interface with each others seamlessly.

Within this study, a system for simulation based optimization process is created using the above mentioned coarse module guidelines. As a case study and an example, an optimization system is utilized in the simulator aided internal combustion engine design process. The simulator numerically solves the mathematical model of the engine, and can predict the engine properties such as performance(power and torque), fuel consumption and exhaust gas composition quite reliably, once the values for design variables are selected. The design variables may cover some or all of the basic dimensions of the engine (including bore, stroke, number of valves, sizes, compression ratio, port timing, etc.), shapes and dimensions of the external ducting (intake and exhaust duct) and also secondary variables (such as injection / ignition timing settings, boost pressure, etc.) which do not change the physical essence of the engine but may have a profound effect on the engine output.

The main problem in the engine design process, as in many other design processes, is how to select proper values for the design variables. In practice, there are too many variables for a human to control systemically while pursuing some predefined property for the engine. It is also often impossible to see the delicate interactions between the separate design variables with regard to the pursued predefined property.

The problem of how to find the proper design values has been tackled in several studies. For example in [58], engine emissions and performance are optimized by adjusting the values for injection parameters (timing and pressure) and the six parameters defining piston bowl shape using genetic algorithms.

In [15], NO_x emissions and fuel consumption and their variances (due to manufacturing tolerances etc.) in the engine population produced are optimized. The main difference to [58] is in the consideration of a large engine population instead of one single engine. This is due to the fact, as the authors state, that it is the engines at the extremes of the population which determine business success or failure, not the nominal engine. Among the 15 design variables used a sensitivity analysis is carried out using a Design of Experiment (DoE) method. DoE takes advantage of orthogonality to reduce the number of experimental runs required to characterize the shape of the surface formed by objective function values. The four most significant factors are taken into closer consideration and for these multi-level full-factorial designs are created and executed to create the response surface model (RSM). The optimization is done using the RSM to save expensive objective function evaluation calls.

In [72], the performance of the engine is optimized by adjusting dimensions of the external ducting, where a performance measure (objective function) is merely the mass of the air trapped inside the cylinder after the intake valve closes. Authors consider cases with 3 and 10 variable designs using the Nelder and Mead simplex method, the two membered evolution strategy and the multi-

membered evolution strategy defined in the study. After the maximum of 3500 objective function calls (flow code simulator runs) the optimization algorithm can find the global optimum with some small tolerance.

In [4], the target of the optimization is a little bit different, namely optimization of the sound of an engine by changing muffler and exhaust manifold designs. By using genetic algorithms and DoE based RSM, target performance level was achieved using function evaluations in order of tens of thousands.

In [40], cam shape for the four stroke internal combustion engine is optimized. Cam shape is defined by 40 floating point values assigned to corresponding control points of the B-spline curve. The shape is optimized using a floating point encoded genetic algorithm with a population size of 40 individuals. The objective function to be optimized is a weighted linear combination of several objective functions, such as maximum force peak, dynamic force fluctuation, etc., which are derived from a simulator model of cam-operated mechanisms. Objective function is highly multimodal and non-linear. With 10 000 to 40 000 objective function evaluations the results of the system presented in a paper form suggest that the system can use simulator programs more effectively than an intuitive human trial-and-error experimentation does.

In the studies mentioned earlier, the focus was on 4-stroke engines. In this study the target of the optimization is the performance of a 2-stroke engine. Performance is defined later in a few different ways, but common to all of these is that all objective function values can be derived from the engine power and torque curves which are produced by a simulator. In some cases, also the effect of transmission and gear ratios are taken into account. The optimization is done by changing the dimensions of external ducting, in this case the dimensions of an exhaust pipe (expansion chamber). In this sense, we are solving a shape optimization task, remotely similar to the ones in [40] and [58] where a cam shape and piston bowl shape were optimized. Optimization of the exhaust pipe shape is fertile because the 2-stroke engine is extremely sensitive to exhaust pipe dimension changes and can easily lack in power more than 50% if a properly shaped exhaust pipe is missing [12].

In our study, we want to efficiently solve (using a minimal number of objective function evaluations) a task which is somewhat more complex than in some of the studies mentioned above and which has some practical value. We alter the dimensions and the shape of the exhaust pipe of a 2-stroke engine, and thus affect power output characteristics of that engine. To represent different shapes of the exhaust pipe, we develop several different models with varying number of design variables and also with varying flexibility of the model to represent different shapes. Our objective functions are more complicated than in most of the previous studies, and we address the very important problem (from the perspective of vehicle performance) of fitting the engine properties to gearing and transmission. This study also adheres to the important topic often faced in real world applications, namely solving optimization tasks in case of several conflicting objectives. When solving the multiobjective optimization problem, we use scalarization methods [43], [45] to convert the problem into a single objective opti-

mization problem. Scalarized subproblem is then solved using an efficient global optimization algorithm to assure that the Pareto optimal solution found is not local but global. We select some global single objective optimization algorithms to be used as solvers in our optimization system and compare their properties, such as efficiency and goodness of solution.

Especially in the field of engineering, multiobjective optimization problems are often solved using multiobjective evolutionary algorithms. In our study, these methods are left out of scope, because approximation of a whole Pareto front (set of mathematically speaking equivalent solutions) is computationally expensive, and visualization of the resulting Pareto front is innate only in the case of two objective functions. With three objective functions, visualizations can be produced, for example, as projections to two dimensions, but with four or more objectives intuitive and easily understandable visualization is practically impossible. In our study we wanted to retain the efficiency and expandability of the system to an arbitrary number of objective functions instead of only two or three. We refer to [16] for a full coverage of the multiobjective evolutionary algorithms.

This study is organized as follows. The first chapter defines the problem in general and introduces some related papers. The second chapter gives a short overview to both local and global single objective optimization, and introduces the principles and some methods of these classes. Additionally, some scalarization methods with either local or global optimization algorithms to solve multiobjective optimization problems are introduced. Depending on the optimization algorithm used, the resulting solutions are Pareto optimal either locally or globally.

In the third chapter the basics of both 2- and 4-stroke internal combustion engines are presented. Additionally, design aspects and problems of the internal combustion engine are discussed. Based on the conclusions drawn, this study is focused to optimize properties of an engine by altering the shape of the external ducting of a 2-stroke engine. At the end of the third chapter, a short overview of the working principles of external ducting (exhaust pipe in this case) is given.

At the beginning of the fourth chapter an overview of the optimization system developed is given. After that separate parts of the system such as the engine simulator, different ways to model the engine configuration (exhaust pipe shape), the objective function (both single- and multiobjective formulation), and the scalarization methods are covered more in depth.

The fifth chapter covers the test results of the system created both in the single- and multiobjective case. The sixth chapter briefly discusses the test results and the problems found during the testing, and presents some ideas for the future work to improve the functionality of the optimization system. The seventh chapter presents the conclusions for this study.

2 OPTIMIZATION AND SOME METHODS

We all are familiar with the basic concepts of optimization. In our everyday life we continually come across with optimization problems. For example, while planning a visit to our friend on the other side of the city, we may plan our route so that it is the shortest or fastest possible. While driving on a highway and noticing that we are a little bit short of fuel, we may adjust our way of driving in order to save fuel and to get to the next gas station. In both of these situations we are optimizing some quantity of the system. In the first example we are optimizing our route, and our objective (function) is to minimize either the distance or time to the destination. In the second example we are optimizing our fuel consumption, and the objective is to maximize the length of travel before the fuel runs out.

In both of these cases some single property of the system is identified, and then optimized (either minimized or maximized). This property is referred to as an objective or criterion. When it is quantitative, the numerical value for the goodness of the given solution can be evaluated by using a problem specific objective function. If in the first example our objective is to minimize the length of the route, the objective function value is simply the length of the given route.

Sometimes it may be necessary to apply some constraints to design variable values for the mathematical model to be meaningful and for the optimal design to be implementable. For example, the length of some physical item may not be negative, or the volume of some object must be sufficient to house some specified apparatus.

If there is only one objective to be improved, it is called a single objective optimization problem. However, in real life, we often face problems with several, often conflicting objectives. What if, in the previous highway example, we were short of fuel, and yet in a hurry to a meeting? Driving fast would consume more fuel, driving slow and we probably would not make it to the meeting in time. In this case we face a multiobjective optimization problem with two conflicting objectives. There could be even more objectives to be considered simultaneously.

Optimization problems can be classified in several different ways. Probably the most obvious distinctions in problems involve variations in the mathematical

characters of the objective and constraint functions. The objective function may be very smooth in some cases, or discontinuous in others. It may be given in a closed analytic form, or its value may be evaluated solving a series of several complicated subproblems or running a computationally costly and time consuming simulation. Table 1 given in [19] represents a typical classification scheme based on the nature of the problem functions, where a significant algorithmic advantage (the optimization algorithm is suited to the problem function type) can be taken of each characteristic.

TABLE 1 Typical classification scheme based on the nature of the problem functions as given in [19].

Properties of objective function	Properties of constraint function
Function of a single variable	No constraints
Linear function	Simple bounds
Sum of squares of linear functions	Linear functions
Quadratic function	Sparse linear functions
Sum of squares of nonlinear functions	Smooth nonlinear functions
Smooth nonlinear function	Sparse nonlinear functions
Sparse nonlinear function	Non-smooth nonlinear functions
Non-smooth nonlinear function	

There are also other features to distinguish different optimization problems. One of these is size, or dimensionality (number of design variables) of the problem. The problem size affects both the storage and the computational effort required to obtain a solution. Hence techniques that are effective for a problem with only few decision variables may have a prohibitive cost for problems with tens or hundreds of decision variables. In problems with linear objective functions dimensions can grow to the order of tens of thousands before the problem is considered big i.e. of large scale nature, whereas nonlinear problems can be considered big with essentially lower number of dimensions, depending on the nature of the functions involved.

One very important distinction is to be made as regards the information that is available for the optimization algorithm during the solution process. In some cases it may be possible to compute analytically first (and second) order partial derivatives of the objective function, while in another case only the function values may be available. The latter is usually the case if the objective function value is drawn from some complex external device, such as a simulator. In a multidimensional case the vector consisting of first derivatives is called a gradient. Gradient refers to a rate of change with respect to the distance of a variable quantity in the direction of the maximum change.

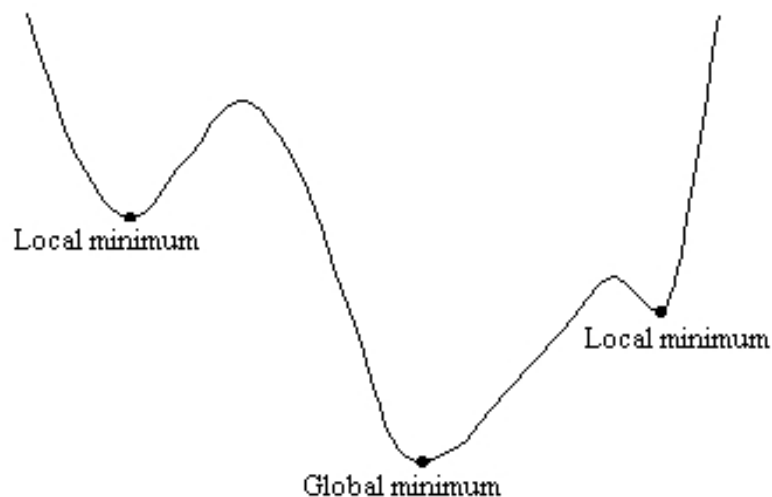


FIGURE 2 An example of a non-convex function with two local and one global minima.

Finally there is one group of objective functions not mentioned in Table 1, which are non-convex. In this case the problem may have a discontinuous feasible region, i.e. allowable area defined by constraints, and there may be multiple locally optimal points. By the type of optima the optimization algorithms are able to find they can be divided into a local and global optimization algorithms. With local optimization algorithms the optimum found is one closest to the given starting point of the algorithm. If the problem is convex, local optimum is also global optimum. See Figure 2 for illustration of non-convex function with two local and one global minima.

Local optimization algorithms are usually quite fast because they exploit gradient information, which requires differentiability of the objective function and constraint functions and depicts precisely the local behavior of objective function. By certain information called optimality conditions, which is based on gradient, local optimization algorithm knows when the local optimum is reached. There are two types of optimality conditions, namely necessary optimality conditions and sufficient optimality conditions. The necessary optimality condition requires that the gradient at that point equals zero. Every local optimum fulfills the necessary optimality condition. On the other hand, there exists points which fulfill the necessary optimality condition, yet they are not local optima. A classic example of this is the saddle point [19, p 64].

A more strict form of optimality conditions, and one which guarantees optimality of the solution, is the sufficient optimality condition, which requires the gradient to equal zero, and the Hessian matrix (the matrix consisting of second order partial derivatives) to be positive definite. A discussion of optimality conditions can be found for example in [19, pp 61-82]. Because gradient points in the direction where objective function values increase most steeply, gradient is useful in optimization. Methods which use gradient information are called gra-

dient based algorithms, and one example of such algorithm is SQP (sequential quadratic programming) [7].

Although gradient information is not always available, a possible strategy is to use a gradient based method also in such cases and estimate the gradient by so-called finite difference approximation. In this technique, the objective function values are calculated along each dimension very near to the original sampling point to create an approximation of the gradient. For example, a forward difference approximation for $f'(x)$ can be calculated as $\frac{f(x+h)-f(x)}{h}$, where $h > 0$ and some small number. This approach is non-trivial in terms of accuracy and increased computational load (objective function must be evaluated at several locations) and implementation also needs some careful consideration [19].

Another method to calculate gradient numerically is so-called automatic differentiation (AD). As stated in [6], AD is a set of techniques for transforming a program that calculates numerical values of a function into a program which calculates numerical values for derivatives of that function with about the same accuracy and efficiency as the function values themselves. The basic process of AD is to take the underlying program which calculates a numerical function value, and to transform it into the transformed program which calculates the desired derivative values. The transformed program carries out these derivative calculations by a repeated use of the chain rule from elementary calculus, but applied to floating point numerical values rather than to symbolic expressions.

The use of automatic differentiation is not always possible with black-box type software, which is the case with simulator in this study. For this reason, AD is not used in this study. We refer to [21] for thorough discussion of the topic.

Gradient free local optimization methods which use only objective function values are called direct search algorithms. One of the most advanced algorithms in this group is the Powell's algorithm [52] (and its modifications), which is on a par with gradient based methods.

Global optimization algorithms are capable of finding a global optimum of the problem, instead of a local one. The global optimum is not necessary unique, since there may exist other optima with the same objective function value. There is no information similar to the gradient information, in case of global optimization, that could be efficiently used: one cannot locally decide where to search next. Nor there are exact rules similar to optimality conditions of the local optimization to go by when the global optimum has been found, unless very limiting assumptions are satisfied [63]. Thus convergence of global optimization algorithms is usually stochastic by nature, that is, the probability of finding the global optimum increases when the optimization process is continued further.

Problems with multiple conflicting objectives are usually solved by converting them into problems with a single objective using scalarization functions discussed in Subsection 2.3.1. After scalarization, multiobjective optimization problems can be solved using methods of single objective optimization on scalarized objective function. Thus we may face same problems with local and global optima as in the case of single objective optimization and, to assure that the solution found is global, we must use a global optimization algorithm.

In the following sections both single- and multiobjective, and local and global optimization algorithms are discussed more in depth. As stated in Section 1, in this study we restrict our interest to computationally efficient single and multiobjective global optimization algorithms. A thorough discussion on optimization in general can be found for example in [7], [19], [43], [63] and [64].

2.1 Local single objective optimization

A general form of the single objective optimization problem is

$$\begin{aligned} & \text{minimize } f(x) \\ & \text{subject to } x \in S. \end{aligned}$$

Function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ to be minimized is called an *objective function*. Without loss of generality we restrict our consideration to minimization because we can always convert maximization problems to minimization problems by changing the sign: maximize f is equal to minimize $-f$. The aim of the minimization process is to find the point with the minimum value of the objective function, i.e. optimum and corresponding design variable values (coordinates). Point x^* is global optimum (minimum), if $f(x^*) \leq f(x)$ with all $x \in S$. If there exists $\delta > 0$ so that $f(x^*) \leq f(x)$ with all $x \in S$, for which is valid $\|x - x^*\| \leq \delta$, point x^* is local optimum.

Minimization of the objective function is done by altering values of the decision or design variables $x \in \mathbb{R}^n$. The decision variables will lie then within the decision space \mathbb{R}^n . Sometimes all decision variable values within decision space are not acceptable for some reason, and the acceptable subset $S \subset \mathbb{R}^n$ is called the feasible region.

If the feasible region is the whole decision space $S = \mathbb{R}^n$, the optimization problem is unconstrained. If, on the other hand, all the decisions are not acceptable, the optimization problem is constrained. The constraints can be bounds for the variables and linear or nonlinear equalities or inequalities, and the feasible region is their intersection. An optimization algorithm tries to locate the optimum within the feasible region. For this reason feasible region is often referred to also as search space. In this work we concentrate on box constrained optimization problems where each of the design variables is bounded between certain minimum and maximum values.

If the objective function is convex, a local optimum is at the same time also a global optimum. In a non-convex case, a local optimization algorithm can not be guaranteed to find the global optimum. For this reason, a plethora of global optimization algorithms have been developed. The basic principles and some examples of global optimization algorithms are discussed in Section 2.2.

Local single objective optimization in general level is out of scope for this study. There are several excellent textbooks on the subject, such as [19]. Global (non-convex) single objective optimization is discussed further in Section 2.2. Only the general principles of gradient free class of local single objective opti-

mization algorithms, namely direct search methods, are given here in a concise manner. Only one gradient free method, the Nelder-Mead simplex algorithm, in the class of local single objective optimization, is presented, because concepts of it are utilized in several global optimization methods.

Direct search methods do not use derivative information of objective and constraint functions, neither do they compute approximations of derivatives. They only need objective function values at specific locations, and this information is used to deduce into what direction search should be directed. Direct search methods are used, for example, when the computation of partial derivatives is impossible (which may be the case with noisy functions or with functions containing unpredictable discontinuities) or otherwise difficult (complexity of functions, computationally costly objective function) or when the objective function is a so-called black box function (internal mechanism and structure of the function is unknown, and only input and output characteristics can be employed) as is often the case with different simulator software.

Common concepts for several local optimization methods are the search direction and the step length. At the current point the search direction and the step length to the search direction are determined, and this process is iteratively executed until some stopping criterion is met. In direct search methods, search direction may be determined for example by selecting decreasing directions from the set of random unit vectors, using each axis as a search direction at a time (univariate search) or testing the behavior of the objective function near the current location and deducing a direction from that information (pattern search). Step length may be constant, or it may be adjusted by the local behavior of the objective function. Two well-known and often referred pattern search methods are the Hooke and Jeeves direct search algorithm [30] and the Powell's algorithm [52]. Another main group of algorithms within direct search methods is the set based algorithms, where set of points are expected to concentrate around the optimum along the search procedure. Probably the best known of these is the Nelder-Mead simplex method, which is discussed in Subsection 2.1.1. For a review of direct search methods, we refer to [38].

2.1.1 Nelder-Mead simplex method

The Nelder-Mead simplex method is a local, gradient free optimization algorithm, and since some global optimization methods to be discussed later borrow its concept of simplex, this method is shortly introduced here. The Nelder-Mead simplex method is a commonly used algorithm for nonlinear optimization. It was presented by Nelder and Mead already in 1965 [49]. The method uses the concept of a simplex, which is a multi dimensional polygon of $n + 1$ vertices in n dimensions, e.g. a line in 1 dimensional case, a triangle in 2 dimensional case, and so forth.

The given starting point is used to construct an initial simplex, a shape with $n + 1$ points, where n is the number of design variables. The objective function values are evaluated at the vertices of the current simplex. The algorithm then

chooses to replace worst (in terms of objective function value) of these evaluated points with a new point. The worst point is thus reflected through the remaining n points considered as a (hyper)plane. In Figure 3 point x_3 is reflected through centroid \bar{x} (centroid is a point located spatially in the mean of the coordinates of the remaining points) and after reflection it is located at x_r . After the reflection, simplex can be expanded, and x_r is moved to location x_e .

In general, after reflection the algorithm uses the following rules:

- If a new point x_r has the best objective function value, simplex is expanded and new reflection is done through the new centroid.
- If a new point x_r has not the best nor the worst objective function value, the simplex is not expanded or contracted, but only reflected.
- If a new point x_r has the worst objective function value, the simplex is contracted and a new reflection is done.

In Figure 4 three different operations, the outside contraction, the inside contraction and the shrink of the simplex are displayed. In outside contraction the worst point is first reflected through the centroid, and then the simplex is contracted. In inside contraction there is no reflection phase. In shrink vertices are brought closer to each other.

The rationale behind the above mentioned rules is that when a better point is found it is reasonable to stretch along to that direction more. On the other hand, if the new point is the worst of all, it is detrimental to continue search along to that direction and thus the simplex is contracted. If the new point is not much better than the previous value then the step should probably be across a valley, so it is reasonable to shrink the simplex towards the best point. Progress of the Nelder-Mead optimization run in a two dimensional case can be visualized as a shape and size changing triangle flipping its way to a local valley.

The usual stopping criteria for the Nelder-Mead algorithm are to stop search when fractional change in objective function values falls below predetermined value, when objective function values at simplex vertices are all alike to some extent, when the diameter of the simplex by some norm is sufficiently small, when the volume of the simplex is sufficiently small or when a predetermined budget for objective function evaluations is exhausted.

2.2 Global single objective optimization

Traditionally most single and multiobjective optimization problems are solved using well known local methods developed for linear and nonlinear optimization. While using these methods, it is often implicitly assumed that the objective function is formulated analytically in a closed form (although this is not necessary) and that it is convex within search space, i.e., it has only a single optimum within the search space. It is important to notice that while using local methods,

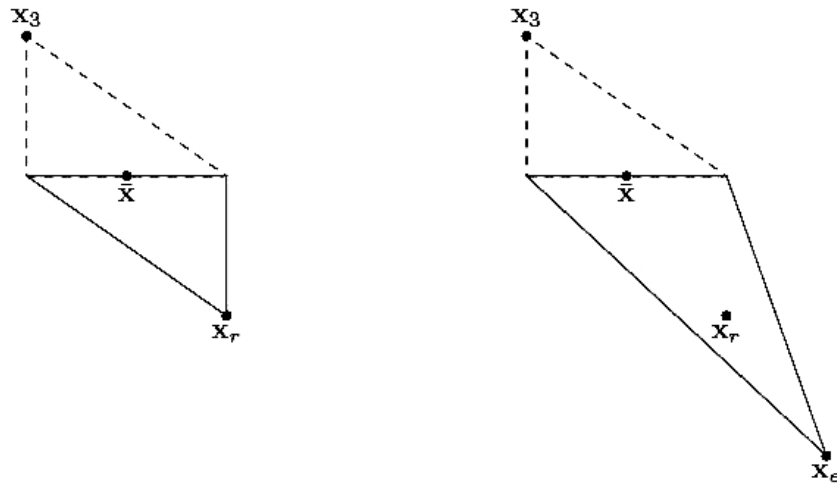


FIGURE 3 Nelder-Mead simplices after a reflection and an expansion step. The original simplex is shown with a dashed line. Figures from [39].

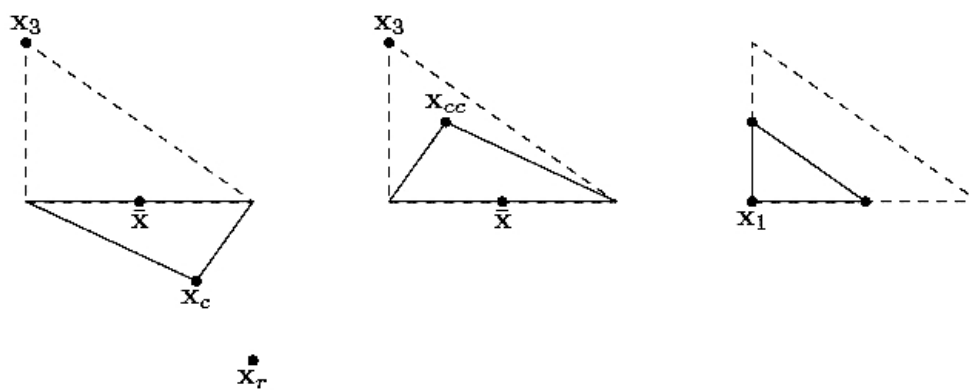


FIGURE 4 Nelder-Mead simplices after an outside contraction, an inside contraction, and a shrink. The original simplex is shown with a dashed line. Figures from [39].

the resulting solution is only locally optimal in both single and multiobjective case.

Several practical optimization applications tend to be of global nature. In these cases the objective function is non-convex and contains several local minima in the search area. Global optimization aims to determine the smallest local minimum among all the local minima in the search area. In the field of local optimization it is easy to judge when optimum is found by checking whether the solution satisfies the optimality conditions or not. Unfortunately, there exist no such general criterion for asserting that the real global minimum has been found.

2.2.1 Classification of methods

Many global optimization methods have been suggested since the early years of the discipline. These methods share common ideas thus making it possible to define classes covering most of the methods. Such a classification is of course not unique, but several classification schemes could be used depending on which features are used in the classification. Below we describe a classification with some examples of methods belonging to the classes as given originally in [63] and updated with more recent ideas in [64]. This classification scheme is based on what kind of a strategy is used to choose the next point to be evaluated. Coarsely, we can speak of local, global and adaptive schemes. Obviously local scheme can not work on its own in the field of global optimization, thus it is commonly used with some global strategy, resulting in a global-local scheme. With global strategy, some technique is used to spread points over the whole search space. With local strategy, it is assumed that it is possible to find a point with a better objective function value at the vicinity of the current point. With adaptive strategy, more effort is put on sampling the search space in the regions where relatively small function values are found during the optimization.

1. Methods with Guarantees (covering methods)

- Bisection Methods
- Interval Methods

2. Search methods

2.1. Global

- Random Search
- Mode-Seeking

2.2. Adaptive

2.2.1. Single Working Point Methods

- Simulated Annealing

2.2.2. Converging Set Methods

- Controlled Random Search
- Genetic Algorithms

2.3. Global-Local

- Multistart
- Clustering Methods

3. Bayesian Methods

- P-Algorithms

Methods with Guarantees

First class, the methods with guaranteed accuracy, are also called covering methods or branch-and-bound methods. These methods guarantee that a solution with a given accuracy is obtained. The price to be paid for this guarantee, however, is that some a priori information of the objective function must be available or some rather restricting mathematical assumptions must be valid.

For bisection methods [63] the objective function must be Lipschitz continuous and the information required is that a Lipschitz constant K is known at the given interval for the objective function. Intuitively, Lipschitz constant K gives a bound to a change rate of the function value, i.e., a line joining two points of the function between given interval can never have slope steeper than constant K . Thus, K is the largest magnitude of the function derivative. When a set of points generated to cover the search space is dense enough, it can then be deduced that the difference between the true global minimum and the best value found (in some of the covering points) will be less than some given threshold.

For Interval Methods [63] the objective function is assumed to be twice differentiable and the first and second partial derivatives are assumed to have a finite number of roots. Using interval arithmetic a union of intervals of decreasing size containing the global minimum is obtained at each step. The algorithm stops when the size reaches some preset limit.

Search Methods

In the second class, search methods with different strategies to choose the next searching point are further divided to the global, adaptive and global-local scheme. In a purely global scheme of Random Search [63] points are sampled uniformly within the whole search space and the best point found is considered as an estimate of the global minimum. This algorithm is mostly used as a part of a more sophisticated algorithm, seldom on its own.

Mode-Seeking [64] is another method based on the global scheme and also one of the first global optimization method using clustering. It uses uniform sampling, and no local optimization procedure is applied. Instead, the algorithm aims at partitioning the region into subregions containing a single minimum. There are also other methods that work on similar principles.

In the Mode-Seeking algorithm the search space is covered by a uniform sample of points. Some small percentage of the best points is retained. These points are expected to cluster around some or the best minima within the search space. The clusters are recognized by using some clustering technique and rated based on their best objective function value. In each of the highest rated clusters (e.g. the 3 best) the retained points define a subregion of the search space for which the same overall procedure is applied again until a global solution is considered to be found.

Adaptive search methods can be divided into single working point and set (or population) based methods. In single working point methods, the behavior to choose the next point is adjusted over time, thus ensuring global search at the early stages of the optimization and convergence in later stages. In set based methods the set of solutions are manipulated by different means and they are expected to concentrate around the global optimum during the optimization run.

Simulated Annealing (SA) [35] employs the adaptive scheme in choosing the search point, and it uses a single working point (only one point is moved at each iteration of the algorithm). SA borrows its name and main principles from physics and more specifically from annealing in metallurgy, a technique which involves heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the metal atoms to become loose from their initial positions and change their positions randomly through higher energy states (unoptimal crystal structure). Slow cooling gives the atoms more chances of finding configurations with lower internal energy than the initial one.

By analogy with the physical annealing process, each step of the SA algorithm replaces the current solution by a random "nearby" solution which may have either a better or worse function value than the current point. A worse point can be selected if it fulfills the so-called Metropolis criterion. Using the Metropolis criterion, the worse point is chosen with a probability that depends on the difference between the corresponding objective function values and on a global temperature parameter T that is gradually decreased during the optimization process. According to the Metropolis criterion, a new point y is selected over the current point x if $e^{(f(x)-f(y))/T} > p$, where p is a uniformly distributed random number between 0 and 1.

The temperature dependency (Metropolis criterion), also referred to as the cooling schedule or acceptance procedure, is such that the current solution changes almost randomly when T is large, but probability to accept "uphill" moves decreases as T approaches zero during the optimization run. The acceptance for "uphill" moves saves the method from getting stuck at local minima.

Controlled Random Search (CRS) [53] is a representative of the adaptive converging set scheme, and as such it uses the population P of points which are expected to concentrate around the global minimum. CRS is purely heuristic with its contraction process where an initial sample of NP points is iteratively contracted by replacing the worst point with a better point. Further discussion about CRS is given in Section 2.2.3.

Genetic Algorithm (GA) [48] is another exemplar of the adaptive converging set scheme. GA is an adoption of biological evolution theories to the global optimization and as such it uses three main operators to evolve the set of solutions (population): selection, crossover and mutation. Before the execution of GA is started, number of solution candidates (population size NP) used in optimization must be defined. Population size has effect on the convergence properties of the algorithm: too small population may lead to a premature convergence to some local optimum, and too large population may slow the convergence of the algorithm to an unacceptable level.

Starting from a randomly generated initial sample S of solution points within the search space, the basic idea of GA is to select a subset of the most fit (in terms of objective function values) individuals for which GA operators' mutation and crossover are used to obtain children, i.e. new sampling points. New sampling points are accepted to the population only if they are fit enough, i.e., have a low enough objective function value. The traditional, binary coded, way to represent each solution as an individual is to encode it as a string of 0's and 1's, by using a 10-bit binary number (suitably scaled) for each design variable. In some cases binary coding may cause problems with accuracy (due to discretization) or by the excessive length of individual representation with large number of design variables. A simple solution to these problems is the use of floating point representation of variables. These GAs are called real-coded GAs, and each individual is coded as a finite-length string of real numbers corresponding to the design variables.

In the GA's selection step some predetermined number of individuals are selected as parents for the next generation. These individuals form the subset Q of individuals, and the composition of this set is biased towards the best individuals. One often used selection method is the roulette wheel selection, where the population is mapped onto a roulette wheel where each individual is represented by a space that proportionally corresponds to the fitness of that particular individual. Individuals to the subset Q are chosen by repeatedly spinning the roulette wheel until all the positions in Q are filled. It is worth mentioning that the selection phase contains a stochastic element due to the randomness of roulette wheel selection, and thus a small proportion of less fit solutions are also selected. This helps to keep the diversity (variation in decision variables) of the population large, which in turn prevents premature convergence to poor local solutions.

The GA's crossover operation (mating) means recombination of two parent individuals. It can be implemented, for example, by taking two individuals from the subset Q , cutting the binary strings at one or more random indices and

exchanging those randomly selected parts between individuals. New child individuals are created as a product. Several different crossover operations exist for real-coded GAs.

Mutation in GA can be implemented by simply flipping a 0 for a 1 or vice versa at some random index in the binary string of the child. Mutations are usually given low occurrence probability per iteration. In real-coded GAs, mutation slightly changes the child individual.

By using selection, crossover, and mutation operators a new generation of individuals is created. This new population shares many of the characteristics of its parent population, but generally the average fitness of the individuals increases since the selection process gravitates towards better individuals. The process of selection, crossover and mutation is iterated over and over again to generate new populations until some stopping condition is met.

One often used method to ensure survival of the best solutions over the generations is to use the elitist selection scheme. In elitism some of the most fit individuals from the current generation are carried unaltered over to the next generation.

Methods in the Global-Local search scheme estimate the global minimum by finding local minima. The starting point for algorithms within this class is the Multistart algorithm [64]. In Multistart, as the name suggests, the local optimization procedure is carried over several times from different starting points. Each of the local optimization runs will converge to some local minimum. Best of these local minima is considered as the global optimum. The multistart algorithm is obviously sensitive to the number and selection of the starting points, which can be purely random or be based on some heuristic.

Multistart may end up into the same local minimum several times. Some clustering method is normally used to avoid this. First, a set of uniformly distributed points within search space is determined. Then the following steps are performed until some stopping condition is found valid.

1. The points are concentrated to the vicinity of some local minima by either leaving out points like in the Mode-Seeking algorithm or by performing few steps of a local optimization algorithm.
2. The clusters are identified by using some cluster analysis technique.
3. Some small percentage of points in each cluster are retained and then the procedure continues from step 1.

Finally a local optimization algorithm is started from the best point in each cluster, and the best result of these local optimization runs is regarded as the global optimum. Several modifications of the original method have been developed by many authors, for example [2] and [65]. A modification where the cluster center is directly identified without any clustering technique (Topographical Global Optimization, TGO) has been developed and applied in [65]. In TGO no actual

clusters are formed but, instead, points that are better than a prespecified number of their nearest neighbours are determined. These points are called topograph minima and they are used as starting points for local optimization.

Bayesian methods

The approach of using Bayesian methods in global optimization aims at producing algorithms that despite having a rather poor efficiency in the worst case analysis can be used to solve average case problems efficiently. These methods are called P-Algorithms [63] and they are based on a meta modelling scheme, where the computationally expensive objective function is replaced with a lower fidelity surrogate model. Surrogate model may be implemented for example by Kriging, artificial neural networks etc. The surrogate is created by sampling the initial set of points within the search space, and after the initial sampling, a stochastic model of the objective function based on all sample points is computed. Then a utility function (known also as an infill sampling criterion, ISC) reflecting the rewards of continuing sampling in a particular region is maximized in order to find the best candidate for a new sample point. The purpose of the utility function is to find a trade-off between sampling in known, promising regions versus sampling in under-explored regions or regions where the variation in function values is high. Due to the large overhead in fitting a sampled dataset to the surrogate model and in selection of sample points, these methods are only suited for problems where the real objective function is very expensive to evaluate. There are many methods employing these ideas, and some of them are discussed in Section 2.2.4.

Other methods

One interesting subclass of methods are hybrid methods, which are not classified in [64]. According to the classification above they may fall into the category of Global-Local scheme. Hybrid methods try to, more or less successfully, combine characteristics of different methods in order to produce a method having the good qualities associated with the original methods without their drawbacks. One commonly seen hybrid method combines the robustness of meta-heuristic methods (such as simulated annealing and genetic algorithms) with the effectiveness of local search methods (producing a coupling of stochastic and deterministic methods).

A meta-heuristic is a method for solving a very general class of computational problems by combining some heuristic procedures. Meta-heuristic methods are considered to be acceptably good solvers for unconstrained optimization problems. They are also robust and can deal successfully with a wide range of problem areas. The drawback of these methods is that they suffer from high computational cost (in terms of objective function evaluations) due to their slow convergence. The main reason to their slow convergence is that these methods explore the search space using more or less random movements without using much of the local information about promising search directions. In contrast

to that, local methods employ local information heavily to determine the most promising search directions, and thus their movements are logical and efficient. And, as has already been mentioned earlier, the general problem with local methods is that they are easily trapped to local minima.

To create more efficient methods meta-heuristic methods can be combined with local search methods. In general, global search (exploration) is responsible for detecting promising areas, whereas local search refines the solutions (exploitation). This is the so-called exploration-exploitation procedure, which results with methods with convergence faster than that of pure meta-heuristic methods. These resulting methods are not easily trapped to local minima because they still have global properties of meta-heuristic methods.

In the following some hybrid methods are represented as examples. In [26] concepts of the local Nelder-Mead simplex method (Nelder-Mead) are combined with a genetic algorithm. The combined method is called Simplex Coding Genetic Algorithm (SCGA), and it is discussed further in Section 2.2.5. In [25] concepts of the local Nelder-Mead simplex method are combined with the simulated annealing method. The resulting method is referred to as Direct Search Simulated Annealing method (DSSA). Further, in [27] a derivative free method is used to produce an approximate descent direction at the current solution. This Approximate Descent Direction method (ADD) is combined with simulated annealing, resulting in a method known as the Simulated Annealing Heuristic Pattern Search method (SAHPS). This method is discussed further in Section 2.2.6.

Within this study methods with guarantees were left out, since there was no information available about the objective function required by those algorithms. Also easily implementable global optimization methods like the genetic algorithms and the simulated annealing (though they have been successfully used in several studies, [4], [58] and [72] to mention a few) were ruled out because of the requirement to solve the optimization problem using a minimal number of objective function evaluations, since they typically require objective function evaluations at least in the order of thousands. In this study the aim was set to the maximum of one to two thousand objective function evaluations in order to carry out the optimization runs within a reasonable time, preferably overnight.

Several methods in the field of global optimization were studied, and a set of powerful algorithms was identified, including DIRECT [51], Multilevel Coordinate Search (MCS) [31], GROPE [17], Differential Evolution (DE) [61], Controlled Random Search [53] (and its variants) [2], Recursive Random Search algorithm [71], Efficient Global Optimization (EGO) [32], a variant of it, ParEGO [37], an enhancement of EGO called SuperEGO [57], hybrid methods Simplex Coding Genetic Algorithm (SCGA) [26] and the Simulated Annealing Heuristic Pattern Search method (SAHPS) [27]. Out of these DE, CRS, SuperEGO, SCGA and SAHPS were selected for further study because of their reported efficiency (CRS [1] and [59], SCGA [26] and SAHPS [27]) and availability of the source code. For benchmarking purposes and to make sure there remain no large unsearched

areas in the search space the Niederreiter random sampling method was selected [42], [50].

Using the classification presented earlier, the selected methods fall into the categories of global search (Niederreiter random sampling), adaptive converging set methods (CRS, DE), Bayesian methods (SuperEGO), and hybrid methods (SCGA, SAHPS).

CRS code was supplied by M.M.Ali, SuperEGO code by M.J.Sasena, SCGA [24] and SAHPS [24] and DE [60] were downloaded from the authors' web pages. The code for Niederreiter random sampling was implemented by Yaohang Li.

2.2.2 Niederreiter random sampling

Strictly speaking, Niederreiter random sampling is not an optimization algorithm as such, but as seen in Section 2.2, it can be classified as a random search method. Niederreiter random sampling was used to gain certain confidence in that there remains no large unexplored areas (due to stagnation of the optimization algorithm to some local optimum) with probably better objective function values than gained by running optimization algorithms. To attain this, the whole search space is covered with (quasi) random points. The points are generated using Niederreiter quasi random sequence generator developed in [42] and [50], and the objective function value is evaluated at every point. The generated points try to maximally avoid each other and cover the search space relatively uniformly, thus, by increasing the number of points, the objective function value of the best solution improves. In Figure 5, 1000 points are displayed in a two dimensional case. In principle, if the number of the points is increased to infinity, the global optimum is found.

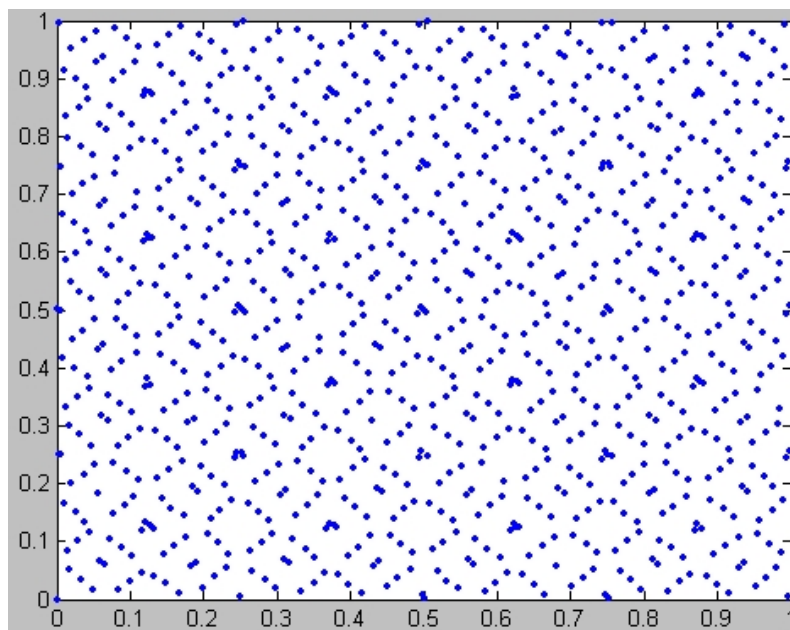


FIGURE 5 Niederreiter random sampling using 1000 points in a two dimensional case.

To create a more realistic optimization algorithm, the best point found by random sampling could be further refined by executing a local search from that particular point.

2.2.3 Controlled random search

The Controlled Random Search (CRS) algorithm was presented originally by W.L. Price [53] already in 1977. Price proposed several versions of the algorithm, the widely cited method is CRS2 [54].

In general CRS is a population based search algorithm. In the Price's original version the search space is randomly sampled to form the population P . Size NP of P is much larger than the number of design variables n , the suggested value for the NP is $10 * n$ [1]. In the next step a simplex is formed by selecting $n + 1$ points randomly from P . A new trial point is generated by selecting one of the points in the simplex which is reflected through the centroid of the remaining points (as in the Nelder and Mead simplex method [49] discussed in Subsection 2.1.1). If the objective function value of the trial point is better than the current worst point in P , the worst point is replaced in the population by the new trial point. The process of forming a new random simplex and generating the trial point is then repeated until some stopping criterion is met. The stopping criteria presented in Subsection 2.1.1 are also valid for the CRS algorithm.

Price himself made the first two improvements to the original CRS algorithm, producing the versions CRS2 and CRS3. In the second version (CRS2) a more sophisticated use is made of the simplices in obtaining new trial points. The difference between CRS1 and CRS2 lies in the way the simplex is formed. In CRS2 the first point of the simplex is always the current best point in the population P (others are randomly chosen), whereas the first point is randomly chosen at CRS1. CRS3 is otherwise similar to CRS2, but it also incorporates a Nelder-Mead type local search from the best $n + 1$ points of the set P .

After Price's initial work the ideas of CRS algorithms have been further extended for example by M.M. Ali and C. Storey [2], who produced the variants called CRS4 and CRS5. Experiments have proved CRS4 to be superior to CRS5 [3]. Both CRS4 and CRS5 employ a local search phase, which is gradient based in CRS5. For this reason, CRS5 is omitted in this study.

Unlike in CRS3, in CRS4 there is no Nelder-Mead type local search. Instead, whenever a new best point x_{min} is found by a manner similar to that in CRS2, it is "rewarded" by an additional search around it by sampling a predetermined number r of points (e.g. $r = 4$) from the beta-distribution using the current x_{min} as its mean and the distance between x_{min} and x_{max} (the worst point in the population) as the standard deviation. This method is reported to be very efficient [1]. In our study we use versions CRS2 and CRS4. A Pascal implementation for both versions of the algorithm was supplied by M.M. Ali.

2.2.4 Efficient Global Optimization

The Efficient Global Optimization (EGO) algorithm to optimize expensive black-box functions was first introduced and described in [32]. Black-box functions are viewed primarily in terms of their input and output characteristics without knowledge of their internal structure. In the field of optimization, a common form of black-box functions is implemented in simulators, the internal structure of which is unknown and which also lack the derivative information.

EGO is not simply an optimization algorithm, it also incorporates a meta modeling scheme. In meta modeling a real and often very expensive objective function is accompanied with a lower fidelity response surface, i.e. meta modelling summarizes how the function typically behaves, and how much the function value tends to change as we move by different amounts in each coordinate direction. It is beneficial if meta model can incorporate estimates of its own accuracy.

In the meta model scheme the real objective function is sampled (i.e. function value is evaluated) only in those points where the meta model suggests that the value of the objective function should improve the most. This scheme should decrease the computational load, because only after a modest number of samples the model should describe the behavior of the true objective function quite accurately. However, it is worth mentioning that fitting the meta model to the existing data may itself be very time consuming as an optimization task. In some cases, this computational load may result in prohibitive costs.

EGO makes use of Kriging [14] to model the search landscape from points sampled during the optimization procedure. More specifically, it exploits a version of the design and analysis of computer experiments (DACE) model of [56], based on Gaussian processes. DACE model has some favorable properties that can be used as building blocks for an efficient search algorithm: the likelihood of the sample has a simple closed form expression from which it is possible to compute the maximum likelihood model, and the error in the expected objective function value of a solution also has a simple closed form expression. Thus, model estimates its own uncertainty in predicting objective function values. Knowledge of the possible error in the response surface is a useful property while trying to locate a minimum on the objective function landscape, and EGO makes use of this property explicitly, as described in the following paragraphs.

The EGO algorithm begins by first generating a number of sampling points (approximately ten times the number of design variables [32]) within the search space. Locations for these points are determined using a Latin hypercube (i.e. space filling) [67] design. A square grid containing sample point positions is a Latin square if, and only if, there is only one sample point in each row and each column of the grid. In a two dimensional case it means that there can not be two sample points with the same x- or y-coordinate values. Latin hypercube is a generalization of the Latin square concept to an arbitrary number of dimensions. By this arrangement sample points are distributed at different locations along each axis instead of lumping them together, and as a result of that a maximum

amount of information about the objective function surface is extracted using a minimum amount of sample points.

In the next step, DACE model is fitted to the points obtained by Latin hypercube sampling. In this phase some optimization algorithm is needed to fit the DACE model parameters to the existing data so that the model maximizes the likelihood of the sample.

To generate a new sample point to evaluate, EGO uses a procedure referred to as the infill sampling criterion (ISC) to decide what point in the search space should be included in the sample next. ISC of the EGO algorithm searches (and this is another optimization task within EGO itself in addition to the meta model fitting task) for the point that maximizes what is called in [32] "the expected improvement". The idea of the expected improvement is illustrated in Figure 6 where (at the point $x = 8$) a normal density function is drawn with the mean and standard deviation suggested by the DACE model. If the function's value at the point $x = 8$ is treated as a realization of the random variable Y with the density function shown in the figure, then there is a probability that the function's value at $x = 8$ is better than the currently known best function value f_{min} . This can be seen from the fact that the tail of the density function shown in Figure 6 extends below the line $y = f_{min}$. Different amounts of possible improvement are associated with different density values. When all these possible improvements are weighted by the associated density value, the result is a value for the expected improvement.

The use of the expected improvement effectively means that EGO weighs up both the predicted value of solutions and the error in this prediction in order to find the point that has the greatest potential to improve the minimum objective function value. EGO does not just choose the solution that the model predicts would minimize the objective function value. Rather, it automatically balances exploitation and exploration (global and local search). A solution which has a low predicted objective function value and low error may not be as desirable as a solution whose predicted objective function value is higher but whose error of prediction is also higher.

Every time a new sampling point has been chosen and evaluated using the true expensive objective function (e.g. simulator run) the DACE model is updated with this new information and the next sampling point is chosen from the location that maximizes the expected improvement using the updated DACE model. The steps of the EGO algorithm as given in [57] are the following

1. Use a space-filling design of experiments to obtain an initial sample of the true expensive objective function.
2. Fit a meta model (Kriging, DACE) to data of points sampled so far.
3. Numerically maximize an infill sampling criterion (ISC) known as the expected improvement function to determine where to sample the next point.
4. Evaluate the value of the true expensive objective function at the point(s) of interest and update the meta model.

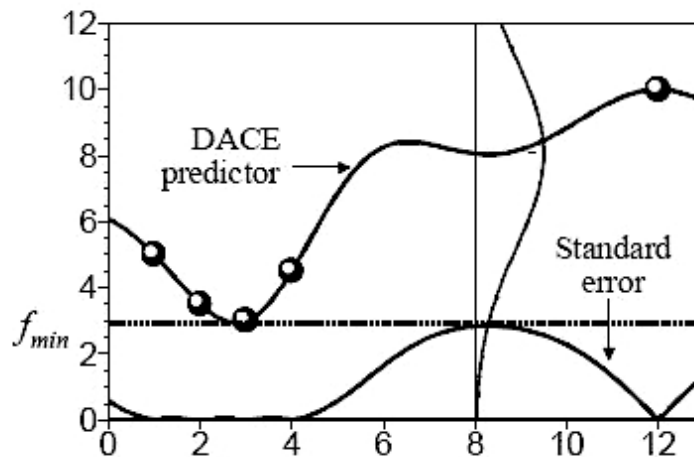


FIGURE 6 Uncertainty about the function's value at a point (e.g. $x = 8$) can be treated as if the value there were a realization of a normal random variable with mean and standard deviation given by the DACE predictor and its standard error. Figure from [32].

5. Stop if the expected improvement function has become sufficiently small, otherwise return to 2.

For a thorough discussion of the EGO algorithm, see [32] and [57].

SuperEGO [57] by M. Sasena is an extension of the EGO algorithm, which aims to advance the efficiency and flexibility of the original algorithm. The author selected the name, *superEGO*, to signify that the extended algorithm can solve a superset of the problems EGO was originally designed to solve. This is done, as the author of [57] outlines, by implementing some improvements. We present two of them in the following:

1. Improving the Kriging modeling in Step 2 of the EGO algorithm. The DACE procedure for determining the Kriging model parameters may occasionally lead to poorly fitting models. A simple strategy for detecting a specific type of modeling error and a method for correcting the situation is employed by replacing Gaussian model with a covariance model and introducing an additional parameter into the covariance model. This should both improve the stability of the Kriging system and to enable the non-interpolating Kriging models.
2. Using alternative ISC in Step 3. EGO uses a criterion known as the expected improvement criterion to select the next sampling point. Other Bayesian analysis methods use different criteria, some search for the minimum of the approximate model, some for the location of maximum uncertainty of the model, and some compromise between the two. Methods vary in the quantification of these ideas. The ability to use an alternative ISC in *superEGO* allows a new degree of flexibility that broadens the application domain of the algorithm.

2.2.5 Simplex Coding Genetic Algorithm

This section is based on [26]. Simplex Coding Genetic Algorithm (SCGA) is a genetic algorithm borrowing concepts from the Nelder-Mead simplex method (see Section 2.1.1). SCGA uses the main functions of the genetic algorithm; selection, crossover and mutation, on a population of simplices to explore the search space and to locate promising areas. Moreover, SCGA exploits the implicit information of promising areas gained by exploration by trying to improve the initial individuals and new children by applying a local search.

At the final stages of the optimization procedure, SCGA applies the Nelder-Mead method on the best point reached by the previous exploration-exploitation procedure. The purpose of this local search is to accelerate the final stages of the GA procedure. That should be an effective strategy because the GA may have a difficulty (due to stochastic components) in obtaining some required accuracy, although the GA may quickly approach the neighborhood of the global minimum.

SCGA starts by generating the initial population P that consists of NP individuals, each individual being a simplex consisting of $n + 1$ vertices, where n is the number of design variables. The vertices in each simplex are arranged in an ascending order so that the vertex with the best objective function value becomes the first. Fitness for each individual (simplex) is defined by the objective function value of its first vertex.

In the next step, a small number of Nelder-Mead simplex method iterations are applied to improve each individual in the initial population. Then the set Q of parents is selected for mating. The size of Q is the same as the size of P , but more fit members from the population P are selected with higher probability to be included in set Q , using the roulette wheel selection introduced in the context of genetic algorithms in Section 2.2. Individuals to the set Q are chosen by repeatedly spinning the roulette wheel until all the positions in Q are filled.

In the crossover phase a random number of parents from the set Q are chosen to mate together to produce children, until each parent in the set Q has mated at least once. Thus the number of children may vary. In crossover, the first baseline child simplex is produced by calculating the average of the vertices of all the parents. In Figure 7(a) a dotted simplex represents a baseline child of the parent simplices S^1 , S^2 and S^3 . Actual children (as many as parents), are generated by randomly moving a baseline child within the circular area defined by the radius of d (d is the maximum distance between pairs of parents) and the centre located at the baseline child (see Figure 7(b) where children C^1 , C^2 and C^3 are produced).

After the mating (crossover) process some children are selected for mutation by choosing a random number from an unit interval $[0,1]$ for each child, and if this number is smaller than the predetermined mutation probability, then that particular child is mutated. Mutation is executed by randomly selecting one of the vertices. This vertex is reflected as a mutation. In child population, the original child is replaced by its mutated counterpart. After mutation process a small number of Nelder-Mead simplex method iterations are taken with all child simplices for improvement. The population for the next generation consists of the

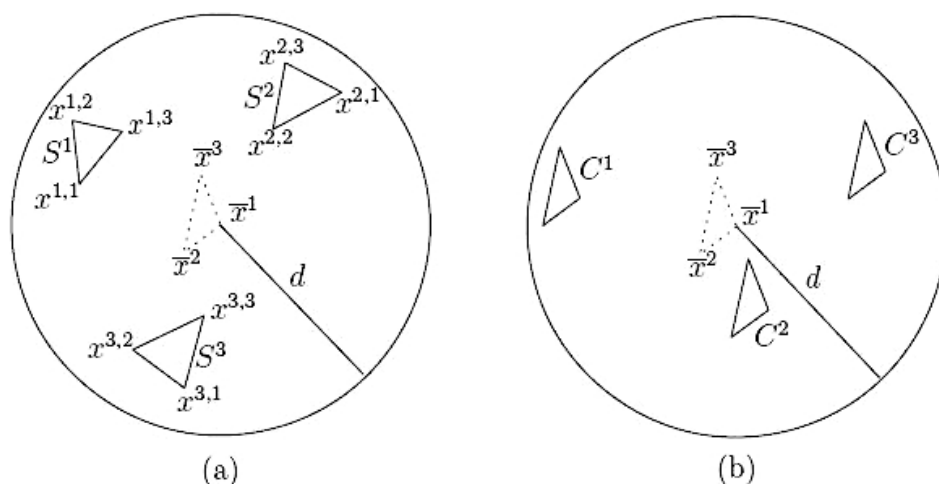


FIGURE 7 An example of SCGA crossover in two dimensions. Figure from [26].

NP best individuals which are selected from the original population P and from all the children produced in the crossover and mutation phases.

After each predetermined number of generations, some of the worst members are removed from the population P . The GA loop is terminated when the objective function values at all vertices of the simplex that contains the best point come sufficiently close to each other (i.e. $f(x^{1,n+1}) - f(x^{1,1}) \leq \epsilon$, where ϵ is a small positive number and set to, for example, 10^{-8} , and first upper index refers to a simplex number, and second upper index refers to a vertex number), or the number of generations exceeds the predetermined number which can be set equal to $\min(10n, 100)$. After the GA loop termination, the final stage is reached and the process is accelerated by constructing a small simplex from the best points obtained by the above procedures. Then, Kelley's modification [33] of the Nelder-Mead simplex method is applied on this simplex to obtain the final solution.

By numerical experiments presented in [26] SCGA seems to work more successfully on some well known test functions (e.g. Branin, Goldstein and Price, Shekel, Rosenbrock and de Jong) than some other meta-heuristic methods do. The number of function evaluations seems to remain roughly in the region of 100-500 times the number of design variables, which is acceptable for our purposes.

2.2.6 Simulated Annealing Heuristic Pattern Search, SAHPS

This section is based on [27]. It is stated in [27] that simulated annealing (SA) [35] (see also Section 2.2) is one of the most effective metaheuristics not only for combinatorial optimization (for which it is traditionally employed) but also for continuous global optimization. However, compared to gradient based optimization algorithms SA uses objective function calls profiligately.

In continuous optimization, hybridizing SA with direct search methods is a practical remedy to overcome the slow convergence of SA. The authors in [27] introduce their own derivative free heuristic method to produce an approximate

descent direction at the current solution, referred to as the Approximate Descent Direction (ADD) method. In ADD, the approximate descent direction v is obtained by randomly generating m points close to the current point, and calculating the direction by using function values and mutual locations of the points. The ADD method is used as a building block for a new pattern search method Heuristic Pattern Search (HPS) that the authors present in the same paper. In the HPS method, the ADD method is used to obtain an approximate descent direction v at the current point. If a better function value along the direction v with a certain step length is obtained, this new point is accepted. On the contrary, if no improvement is obtained along the direction v , then a mesh of points is generated around the current point. To avoid searching randomly in all these directions, directions which are similar to the direction v by some degree (which is controlled by the pruning control parameter β) are removed (pruned), as that direction is already known to be futile.

Finally, SA and HPS are hybridized to construct a global search method, known as the Simulated Annealing Heuristic Pattern Search (SAHPS) method. The SAHPS tries to get better movements through the SA acceptance procedure or by using the HPS procedure. First a new circular neighborhood is created around the current point with radius ϵ . Within this neighborhood m_1 trial points are generated. If more than m_{ac} out of m_1 trials are accepted by the SA acceptance procedure the next major iteration of SAHPS is taken. Otherwise, within the same SAHPS iteration, HPS is executed for m_2 times. The authors of [27] call the procedure of creating trial points as diversification, and the local HPS procedure as intensification. In the early stage of the search diversification is needed more than intensification, while the contrary is true at the final stage of the search. For this reason, m_2 is initialized with a moderate value and increased while the search progresses.

When the cooling schedule is completed (temperature T of SA falls below a predetermined minimum) or the function values of two consecutive trials are sufficiently close each other or the number of iterations exceed 50 times the number of design variables, the execution of the main loop of SAHPS is considered terminated. As the final stage of the search the best point obtained so far is further refined using the Kelley's modification [33] of the Nelder-Mead simplex method.

According to numerical experiments presented in [27] the SAHPS seems to work successfully on some well known test functions compared to two other SA based hybrid optimization methods, especially with slightly higher ($n > 5$) problem dimensions. The number of function evaluations seems to remain roughly in the region of 100-500 times the number of design variables, which is acceptable for our purposes.

2.2.7 Differential Evolution

This section is based on [8] and [61], which both give a detailed description of the Differential Evolution (DE) algorithm. The Differential Evolution algorithm is a member in the family of the evolutionary algorithms, and, to be more accurate, a

form of the evolution strategy (ES) [9]. As such, differential evolution is a simple, population based stochastic optimization algorithm. It was successfully applied to the optimization of some well known nonlinear, non-differentiable and non-convex functions by Storn and Price [61] in 1997. DE combines simple arithmetic operators with the genetic operators (familiar from genetic algorithms) of selection, crossover and mutation. A randomly generated starting population evolves to a final population, from which the individual with the best objective function value is picked up as the final solution when the search procedure terminates.

GA and ES have both similarities and differences. The crossover (also known as mating or recombination) processes of both methods are similar in that they facilitate the search process by mixing the successful information contained in more fit members of the population to create new members. In GA the crossover step is the main search step, while ES uses it as a secondary operator, or not at all.

In GA, the mutation operator ensures that the genetic material (different design variable values) contained within a population between successive generations is sufficiently diverse to prevent premature convergence to some local optimum. In ES, mutation is the main search step, and was originally implemented as a Gaussian-distributed move away from the current solution vector. This technique is effective when the average mutation step length, i.e. amount of change in design variables, away from the current solution is comparable to the standard deviation of the actual distribution of the design variable values in the population. However, according to [8] that approach is computationally expensive to implement.

Ideally, the mutation step length is the function of the design variable (range of variable values) in question and the state of the evolutionary process. DE avoids the problem of selecting a proper mutation step length explicitly by using difference vectors formed from design variable values in the evolving population as a convenient and appropriately scaled source of perturbations. Therefore, as the part of the search space which is occupied by current population contracts and expands over generations, the random step length in each dimension adapts accordingly. This crucial idea differs from the idea of a mutation operator as used by traditional ES in which predetermined probability distribution functions determine vector perturbations.

As the execution of the DE algorithm starts, the initial population P of DE is formed and consists of NP individuals (vectors), each with n components. NP does not change during the optimization process. The initial population of vectors (of real coded design variables) is chosen randomly and should cover the entire search space to encompass sufficient diversity. If some already known good design is available, the initial population might be generated by disturbing its coordinates by adding normally distributed random deviations to them.

After the initialization phase, wherein the first parent population is created, the main loop of the DE algorithm is started with its three distinctive operators, mutation, crossover and selection. In the mutation phase at step 1 in Figure 8 DE generates the same number of mutated vectors as there are members in the current population. These mutated vectors are later used in the crossover phase

as mates for each member in the parent population. The mutation process begins by choosing three vectors randomly from the current population, each with a uniform selection probability. The first selected vector forms the base value for the mutated vector. The other two vectors are paired to create a difference vector, whose components represent a random mutation step length for each dimension. The difference vector is multiplied by a scalar F (this step is not visible in Figure 8, where F is 1) ranging in $[0,2]$ to create the mutation step length vector. Finally, a mutated vector is created when the first selected base vector is added to the step length vector. The whole mutation process is repeated in each DE iteration NP times so that a new mate for each member in the parent population is created.

The rationale of the mutation process above is that the length of the mutation step in each dimension will evolve proportionally over time, taking small steps when the variation in the values of a given design variable within a population is small, and large steps when that variation is large.

In the crossover phase (parameter mixing), at step 2 in Figure 8, the design variable values from the mutated vector are mixed with the design variable values from another predetermined vector from the parent population, the target vector, to yield a so-called trial vector. In DE, each member of the parent population serves as a target vector one after another. Thus, each parent is allowed to undergo recombination exactly once per iteration of DE by mating with a mutated vector. The crossover process in DE thus creates a child population of the same size as the parent population. Each design variable of the parent is recombined in a series of Bernoulli trials, where each of the design variable values assigned to the child comes from either the original parent or the mutated vector. Each of the design variables is evaluated one after another. The crossover constant, CR , is used to control the rate at which crossover occurs within a single recombination cycle. CR can be loosely interpreted as the probability that a given design variable value will come from the mutated vector rather than the original parent vector during the Bernoulli trials. If $CR = 1$, then the trial vector is identical to the mutated vector. The loose interpretation for Bernoulli trials stems from the requirement that if all previous Bernoulli trials yield design variable values from the original parent vector, the final design variable value is deterministically chosen to come from the mutated vector. Thus, the child vector will virtually always differ from its parent vector by at least one design variable value.

In the selection phase at step 3 in Figure 8, each vector in the child population is evaluated for fitness, on a competitive basis, against the fitness of its parent vector. The one with a better objective function value of the two survives into the next generation. Thus, the trial vector replaces the target vector in the following generation, if the trial vector yields a lower objective function value than the target vector. The primary benefit of this scheme is that it resists loss of diversity by forbidding both the parent vector and its respective child vector to survive. Parent and child vectors have some identical variable values, and if they both survive, the population may be driven to a homogenous state, where the diversity of individuals will be low and, thus, the DE will be unable to continue the search.

After the selection phase the new population becomes a new parent population and the evolutionary process of mutation, crossover and selection continues until the fitness of the best vector in the population converges to some specified value, some termination criteria is valid, or the predetermined budget for generations or function evaluations is exhausted.

The above scheme is not the only variant of DE. Other DE strategies can be classified by the notation $DE/x/y/z$, where x specifies how the vector to be mutated is selected (randomly from a population or the best vector in a population), y is the number of difference vectors used, and z denotes the crossover scheme. Using this notation, the basic DE strategy described above can be expressed as $DE/rand/2/bin$. "Bin" means that a crossover is made using independent binomial experiments, as explained earlier.

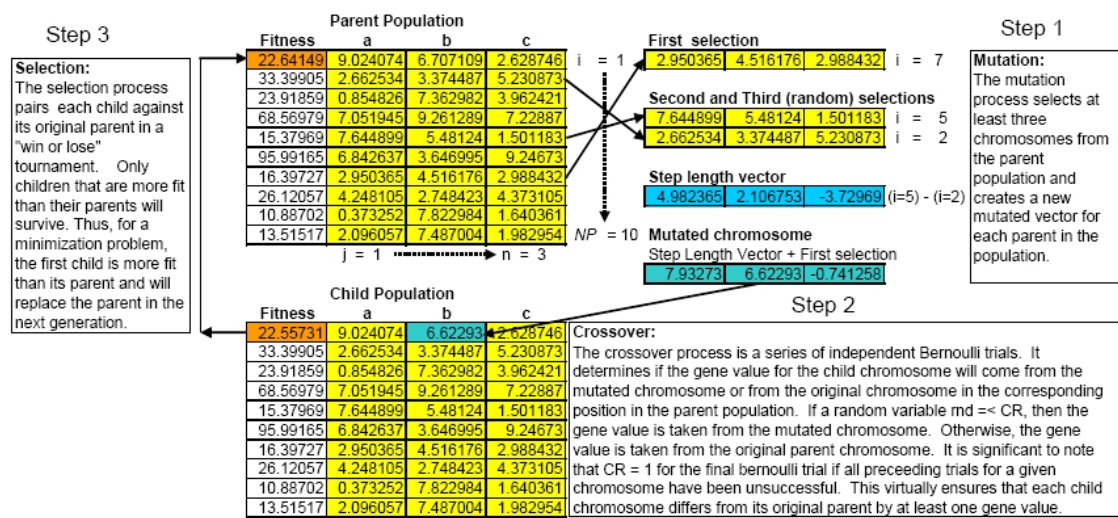


FIGURE 8 Steps of the main loop of the DE algorithm. Notice that a chromosome equals to an individual, and a gene equals to a design variable in this figure. Figure from [8].

2.3 Multiobjective optimization

This section is based on [43, Chapter 2]. Multiobjective optimization is needed whenever there are more than one objective function to be optimized simultaneously, as is the case often with real world applications. A general form of the multiobjective optimization problem is

$$\begin{aligned} & \text{minimize } \{f_1(x), f_2(x), \dots, f_k(x)\} \\ & \text{subject to } x \in S, \end{aligned}$$

where

$f_i(x)$ is objective function ($i = 1, \dots, k$), whose value depends on design variable vector x ,

$k (\geq 2)$ is the number of conflicting objective functions.

A vector consisting of the values of all the objective functions, $z_i = f_i(x)$, is called

objective vector $z \in \mathbb{R}^k$. The space is \mathbb{R}^k where objective function values are called the objective space.

In multiobjective optimization the concept of optimality is not as straightforward and unambiguous as it is in the single objective case. In the multiobjective case we are trying to minimize the values of all the conflicting objectives at the same time, but usually there exists no single point within the feasible region where all the objectives reach their minima simultaneously. Instead, a multiobjective optimization problem has a set of optimal compromise solutions, which cannot be mathematically ordered, and we need a human designer (known as the decision maker in the multiobjective optimization literature) to select one from the set as the final solution. A set of optimal solutions is called the Pareto optimal set, and this set can be visualized as a Pareto front (see bold lines in Figure 9). A solution belongs to the Pareto optimal set if none of the objective function values can be improved without degrading the value of at least one objective. Mathematically, the point $x^* \in S$ is globally Pareto optimal if there exists no other feasible point $x \in S$ so that $f_i(x) \leq f_i(x^*)$ with all $i = 1, \dots, k$ and $f_j(x) < f_j(x^*)$ for at least one j . The objective vector z^* is Pareto optimal, if the corresponding point $x^* \in S$ in feasible region is Pareto optimal. Local Pareto optimality can be defined similarly to the single objective case in the vicinity of the current solution.

A usual way to solve multiobjective optimization problems is to convert them into single objective optimization problems using the method of scalarization. The resulting single objective subproblem is then solved using an appropriate single objective solver. It is important to emphasize that depending on whether the solver is local or global, the resulting solutions are either locally or globally Pareto optimal.

For an ideal scalarization method there are two requirements: it must be able to find any Pareto optimal solution, and every solution it gives must be Pareto optimal [43, p 62]. Due to the nature of multiobjective optimization, some sort of preference information over mathematically equivalent solutions is needed. Typically we assume that we have a human decision maker available who can give preference information, and we can then define a Pareto optimal solution that satisfies the decision maker most as a final solution. In our desert driving example the driver / decision maker may prefer having the certainty of arriving to the destination to being there on time. Another driver, under the same circumstances, might be anxious to get to the destination on time while taking the risk of not getting there at all.

Especially in the field of engineering, multiobjective optimization problems are often solved using multiobjective evolutionary algorithms. In our study, these methods are left out of scope, because approximation of a whole Pareto front is computationally expensive, and visualization of the resulting Pareto front is innate only in the case of two objective functions. With three objective functions visualizations can be produced, for example, as projections to two dimensions, but with four or more objectives intuitive and easily understandable visualization is practically impossible. In our study we wanted to retain the efficiency and expandability of the system to an arbitrary number of objective functions instead

of only two or three. For the above mentioned reasons we concentrate purely on different scalarization methods in our study. We refer to [16] for full coverage of multiobjective evolutionary algorithms.

In this study six different scalarization methods are employed. They fall into three categories of scalarization methods, as discussed in Subsection 2.3.1. First we use a method producing a neutral compromise solution without any preference information. In the second method we use different weighting for each objective function and thus alter their relative emphasis. Then we have three methods which use a reference point, a vector consisting of aspiration levels (function values that are satisfactory or desirable) for each objective function. Our last method is interactive, based on decision maker's continuous involvement in the search process via classification of objective functions.

For many scalarization methods some information about the ranges of points in the Pareto optimal set is needed. The lower bounds are defined by an ideal objective vector z^* , whose components are obtained by minimizing each of the objective functions individually. With conflicting objectives the ideal objective vector is not reachable, but it can be considered as a reference point, something to go for, and it is better than any Pareto optimal solution. A vector strictly better than z^* might be called a utopian objective vector z^{**} .

The upper bounds of the Pareto optimal set are much more difficult to obtain. A vector containing upper bounds is called a nadir objective vector z^{nad} , and its components can be estimated from a payoff table, which is formed by using information obtained when calculating the ideal objective vector. The row i of the payoff table displays the values of all the objective functions calculated at the point where objective function f_i obtains its minimal value. Hence, the components of the ideal objective vector are at the main diagonal of the table. The maximal value of the column i in the payoff table can be selected as an estimate of the upper bound of the objective function f_i . This estimated nadir objective vector may not be very good in all cases. For further details, see [43] and the references therein.

In Figure 9 there are two objective functions z_1 and z_2 and the feasible objective region Z (the set of all possible solutions of an optimization problem in the objective function space). The ideal objective vectors are represented by black points, the grey points represent nadir objective vectors, and the bold line is the Pareto optimal set.

2.3.1 Scalarization methods

In the case of multiple objectives an optimization problem must be typically scalarized, i.e. converted into a single objective problem. The single objective optimization problems are then solved using appropriate single objective solvers. Depending on whether the solver is local or global, the resulting Pareto optimal solutions are also either local or global. There are several different methods for scalarization. These methods can be classified into different categories according to different criteria. Here classification given in [43] and based on the participa-

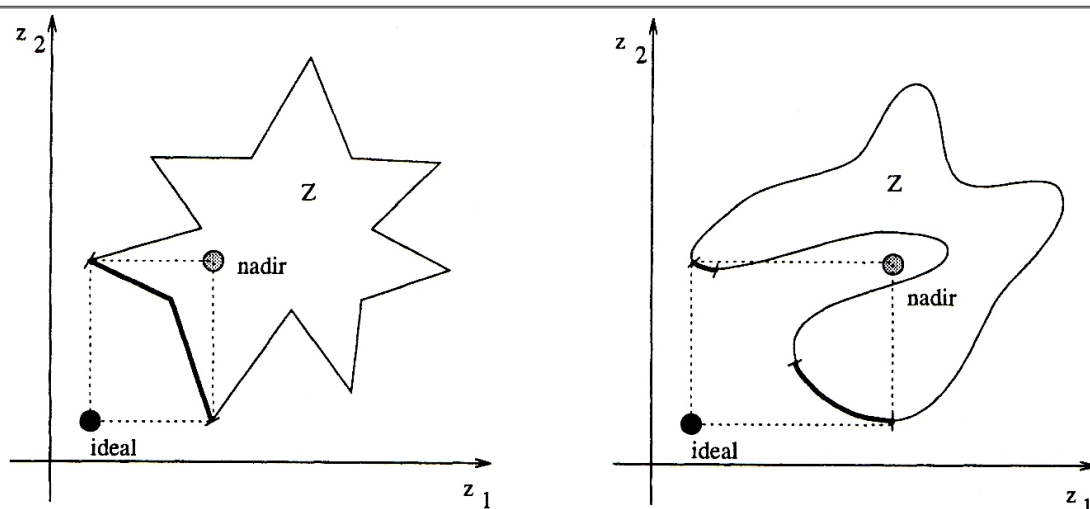


FIGURE 9 Pareto optimal sets, ideal and nadir objective vectors in two cases. Figures from [43].

tion of the designer / decision maker in the solution process is presented:

1. Methods where no articulation of preference information is used (no-preference methods).
2. Methods where a posteriori articulation of preference information is used (a posteriori methods).
3. Methods where a priori articulation of preference information is used (a priori methods).
4. Methods where progressive articulation of preference information is used (interactive methods).

In no-preference methods the opinions of the decision maker are not taken into consideration. The decision maker may either accept or reject the solution. It seems reasonable to assume that the solution best satisfying the decision maker can not be found using methods in this category. This is why these methods are suitable only for situations where no decision maker is available or the decision maker does not have any special expectations for the solution and he/she is satisfied simply with some Pareto optimal solution.

In a posteriori methods the Pareto optimal set or at least part of it is generated and presented to the decision maker, who selects the one that pleases him/her most. Probably the best-known representative of a posteriori methods is the weighting method [43, p. 78], where the idea is to associate each objective function with a weighting coefficient and minimize the weighted sum of the objectives. A set of different Pareto optimal solutions is produced by creating a set of different weighting coefficients and solving the corresponding problems.

One of the problems with the weighting method is the weak stability of the system: an evenly distributed set of weighting vectors does not necessarily

produce an evenly distributed representation of the Pareto optimal set. Another problem of the weighting method is that it does not work with non-convex problems.

Also, in general, there are some problems with methods of the a posteriori class. Methods in this class require lots of objective function evaluations and they are thus computationally expensive. Hence they are only of little value for problems where the evaluation of objective function value is time consuming, and computational cost may grow prohibitively. Additionally, for a decision maker it is difficult to select the final solution among a bulk of the Pareto optimal solutions. However, it is worth mentioning that the weighting method can be used also as an a priori method if the decision maker specifies a weighting vector representing his/her preference information.

In a priori methods the decision maker must specify his/her preferences before the solution process where a satisfying solution is to be searched. The difficulty here is that the decision maker does not necessarily know beforehand what kinds of results it is possible to attain or how realistic his/her expectations are. A priori methods are often based on some kind of ordering or classifying the objective functions (value function method, lexicographic ordering, weighting method) or setting goals or optimistic aspiration levels for the objective functions. Deviations from these aspiration levels are minimized.

In interactive methods a solution pattern is formed and repeated iteratively, overcoming many weak points of the three classes above. Only part of the Pareto optimal set has to be generated and evaluated, and based on this data the decision maker can further adjust his/her preferences as the solution process continues. In contrast to other classes, the decision maker can lack knowledge about the global preference structure. Due to the interactive solution process he/she will learn about the nature of the problem and will probably have more confidence in the final solution.

In this study we consider some different scalarization methods. One of the methods (neutral compromise solution) belongs to the class where no articulation of preference information is used. The next four methods belong to the class where a priori articulation of preference information is given, one of these uses weights to manifest the preference information (weighted and scaled Chebyhev scalarizing function), other three (ACH, GUESS, STOM) are based on the reference point. Finally we consider the interactive NIMBUS method.

All these methods are selected to give a short overview of the different methods, and three reference point based methods are used because different scalarization functions tend to produce slightly different solutions. These three methods were selected for consideration based on the results of [44] where 15 scalarizing functions are numerically and theoretically compared.

The first of our scalarization functions belongs to the class of methods with no-preference information. It produces a neutral compromise solution (NCS) [68], which is Pareto optimal, by solving the problem

$$\begin{aligned} & \text{minimize } \max_{1 \leq i \leq k} \left[\frac{f_i(x) - \bar{z}_{i,mid}}{z_i^{nad} - z_i^{**}} \right] + \rho \sum_{i=1}^k \frac{f_i(x) - \bar{z}_{i,mid}}{z_i^{nad} - z_i^{**}}, & (1) \\ & \text{subject to } x \in S, \end{aligned}$$

where for each $i = 1, \dots, k$

$$\bar{z}_{i,mid} = \frac{z_i^{**} + z_i^{nad}}{2}$$

$f_i(x)$ = value of i :th objective function at x

z_i^{nad} = approximated nadir objective vector component

z_i^{**} = approximated utopian objective vector component

ρ = some small positive value.

A solution gained by this method should be located somewhere in the middle of the Pareto optimal set because the reference point $\bar{z}_{i,mid}$ is located precisely in the middle of the ranges between the upper and lower bounds i.e. the nadir and ideal objective values. Thus the solution of (1) is a neutral compromise between conflicting objectives, as its name suggests. It is stated in [68] that neutral solutions like the one defined above might serve as a starting point for interaction with the decision maker. From that point on the decision maker may balance between different objectives, and emphasize the ones he/she wishes.

To gain control over different preferences for each objective, the weighted and scaled Chebychev scalarizing function (WSC) from the a priori class can be used. With this function the designer manifests his/her preference information as weights for each objective. The problem with weights is that one objective with a long range may have a major effect over a shorter ranged objective regardless of the weight values. Because of the different ranges of the objective function values, the effect of the weights is a little bit fuzzy. For this reason, the original weighted Chebychev function which minimizes the weighted distance between the Pareto optimal set and the approximated ideal vector (given in [43, p. 97]) is modified to take the different ranges of the objective function into account. It is formulated as:

$$\text{minimize } \max_{i=1,\dots,k} \left[w_i \left(\frac{f_i(x) - z_i^{**}}{z_i^{nad} - z_i^{**}} \right) \right] \quad (2)$$

subject to $x \in S$,

assuming $z_i^{**} \neq 0$ for all $i=1,\dots,k$.

This is essentially the same as the original weighted Chebychev method, but the expression in parenthesis is scaled to a similar range between different objective function values. It is possible to add a so-called augmentation term into (2) similarly to (1) and (3) to guarantee Pareto optimality of the solution.

In general, scalarization functions with augmentation terms produce properly Pareto optimal solutions with bounded trade-offs [43].

With weight based scalarization functions there is a difficulty in seeing how weights are related mutually and to actual objectives. For this reason, the result of the current weighting in terms of separate objective function values can be evaluated only after the whole optimization run has been completed. A more straightforward way to manifest preference information is the use of a reference point, a vector consisting of aspiration levels (objective function values that are satisfactory or desirable to the decision maker) for each objective function. With a reference point, preference information can be given in a straightforward way and without guesswork (in contrast to e.g. weights) using real objective function values, which are readily familiar and understandable to the designer / decision maker.

One method relying on designer supplied reference point information is the achievement scalarizing function (ACH) [45] from the a priori class. It finds the Pareto optimal solutions closest to the reference point, and different Pareto optimal solutions can be obtained by adjusting the reference point accordingly. The problem to be solved is similar to (1) (only the reference point is different), and it is formulated as:

$$\begin{aligned} & \text{minimize } \max_{i=1,\dots,k} \left[\frac{f_i(x) - \bar{z}_i}{z_i^{nad} - z_i^{**}} \right] + \rho \sum_{i=1}^k \frac{f_i(x)}{z_i^{nad} - z_i^{**}}, & (3) \\ & \text{subject to } x \in S. \end{aligned}$$

Our second scalarization function exploiting the reference point information comes from the satisficing trade-off method (STOM), and here we present the formulation given in [45]

$$\begin{aligned} & \text{minimize } \max_{i=1,\dots,k} \left[\frac{f_i(x) - z_i^{**}}{\bar{z}_i - z_i^{**}} \right] + \rho \sum_{i=1}^k \frac{f_i(x)}{\bar{z}_i - z_i^{**}}, & (4) \\ & \text{subject to } x \in S. \end{aligned}$$

The third scalarization function which uses reference point is related to the one used originally in the GUESS method and slightly modified in [45]. An augmentation term that was not used in the original formulation is included in order to guarantee Pareto optimality, and we assume that $\bar{z}_i < z_i^{nad}$ for all $i = 1, \dots, k$

$$\begin{aligned} & \text{minimize } \max_{i=1,\dots,k} \left[\frac{f_i(x) - z_i^{nad}}{z_i^{nad} - \bar{z}_i} \right] + \rho \sum_{i=1}^k \frac{f_i(x)}{z_i^{nad} - \bar{z}_i}, & (5) \\ & \text{subject to } x \in S. \end{aligned}$$

Solutions produced by all the previous scalarization functions are Pareto optimal.

NIMBUS method

NIMBUS (Nondifferentiable Interactive Multiobjective BUndle-based optimization System) [43, S. 5.12.] is an interactive multiobjective optimization method designed especially for efficient handling of nonlinear functions. For that reason it is capable of solving complicated real-world problems.

In the NIMBUS method the interaction phase has been aimed to be comparatively simple and easy to understand for the decision maker. At each iteration the NIMBUS method offers flexible ways to direct the search according to the designer's wishes by means of classification including aspiration levels and upper bounds. Aspiration levels are used because they do not require consistency from the decision maker and they reflect his/her wishes well. The use of classification with aspiration levels avoids using other difficult and artificial concepts for preference information extraction.

The classification of the objective functions means that the decision maker indicates what kinds of improvements are desirable and what kinds of impairments are tolerable. The basic idea in classification is that the decision maker contemplates the current Pareto optimal objective function values $f_i(x^c)$ at each iteration of the NIMBUS method and assigns each of the objective functions f_i into one of the following five classes depending on his/her preferences:

1. I^{imp} , function value should be improved as much as possible.
2. I^{asp} , function value should be improved to a certain aspiration level.
3. I^{sat} , function value is satisfactory at the moment.
4. I^{bound} , function value is allowed to impair to a certain upper bound.
5. I^{free} , function value is temporarily allowed to change freely.

After the decision maker has classified the objective functions using the above classes, and specified aspiration levels and upper bounds (if required) the original multiobjective optimization problem is transformed into a single objective optimization subproblem, which is then solved. The resulting Pareto optimal solution reflects the classification as well as possible. Then a new iteration of method execution starts. In the synchronous version of the NIMBUS method, several scalarizing functions leading to different subproblems may be utilized using same preference information. In the synchronous version the decision maker must define how many (one to four) different scalarizations he/she wishes to use at each step. Standard NIMBUS scalarization (STD) given in [45] produces Pareto optimal solutions and is formulated as

$$\text{minimize } \max_{\substack{i \in I^< \\ j \in I^{\leq}}} \left[\frac{f_i(x) - z_i^*}{z_i^{nad} - z_i^{**}}, \frac{f_j(x) - \hat{z}_j}{z_j^{nad} - z_j^{**}} \right] + \rho \sum_{i=1}^k \frac{f_i(x)}{z_i^{nad} - z_i^{**}} \quad (6)$$

$$\begin{aligned} \text{subject to } & f_i(x) \leq f_i(x^c) \text{ for all } i \in I^< \cup I^{\leq} \cup I^=, \\ & f_i(x) \leq \varepsilon_i \text{ for all } i \in I^{\geq}, \\ & x \in S. \end{aligned}$$

Other scalarization functions used in NIMBUS, namely ACH, STOM and GUESS are given in Equations 3, 4 and 5, respectively. Use of several scalarizations synchronously results with a set of slightly different Pareto optimal solutions from which the decision maker can choose the best as a starting point for a new classification. The decision maker can also generate an arbitrary number of intermediate solutions between any two Pareto optimal solutions found so far, and use them as a base for a new classification, if desired.

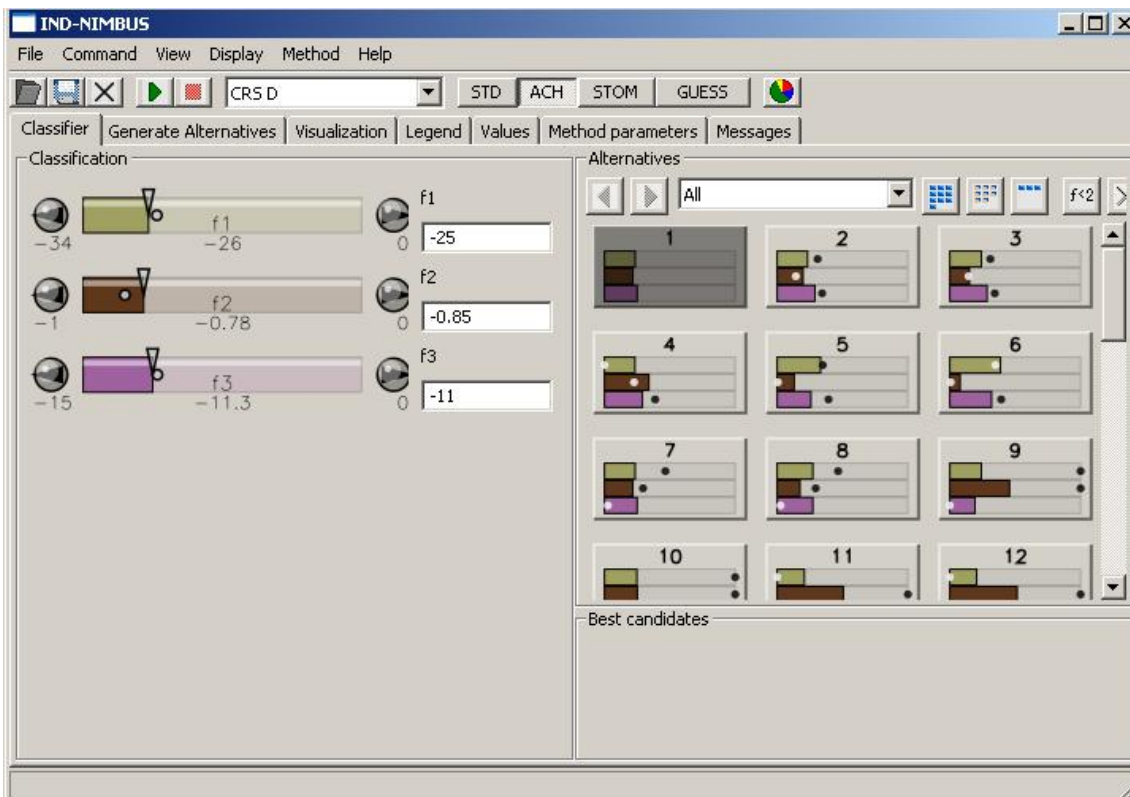


FIGURE 10 Sample screenshot of NIMBUS software.

In Figure 10 we present an example screenshot of the IND-NIMBUS software (which implements the NIMBUS method) tackling a problem with three objective functions to be minimized. On the left side of the screen, the solution to be classified is displayed in the form of bars. The end points of the bars represent the ranges of each objective function in a set of Pareto optimal solutions. The length of the bar represents the current value of the corresponding objective function (the less coloured the area is, the better the value).

Classification can be adjusted either by entering the desirable objective function values to fields f_1 , f_2 and f_3 , or by clicking the bar with the mouse to indicate a desirable value. Clicking the current value means that it is considered

satisfactory and clicking the end point on the left means that the corresponding function should get as good values as possible. Clicking the right end point or leaving the function unclassified means that it can change freely for a while. The right side of the panel contains already found solutions, and anyone of them can be freely chosen as a base for a new classification. When a solution from the right side is selected, it is shaded on the right side, and details of it are displayed on the left side. On the bottom right corner there is an area labelled "Best candidates". This is an area where the designer can drag and drop solutions for latter inspection or to be used as a starting point for a new classification.

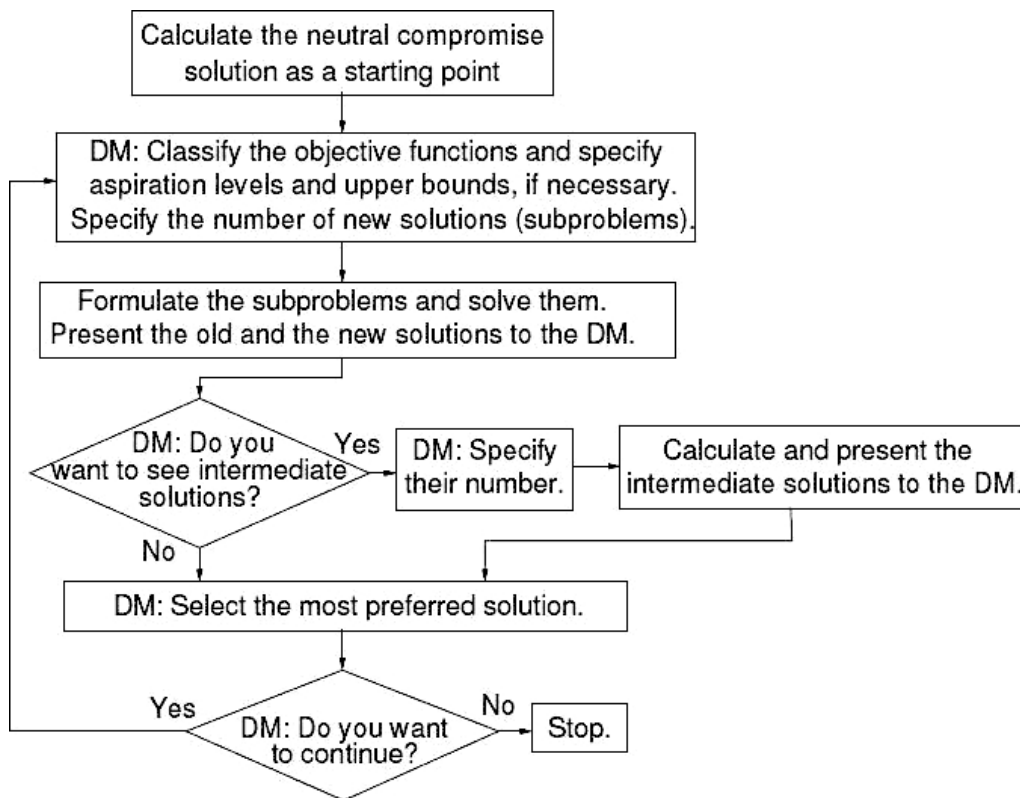


FIGURE 11 A flowchart of the NIMBUS algorithm. Figure from [22].

In Figure 11 a flowchart of the NIMBUS algorithm is given to illustrate the role and possibilities of the decision maker (DM) in the solution process. For further details and more recent advances in the NIMBUS method, see [43] and [45]. An implementation of NIMBUS operating on the Internet, called WWW-NIMBUS, can be found at [46]. WWW-NIMBUS offers both local and global underlying solvers for scalarized subproblems. For academic teaching and research purposes WWW-NIMBUS is free to use.

In this chapter we have provided some perspectives to the field of optimization in general, and we have also discussed topics of local and global optimization. Further, we have discussed differences between single and multiobjective approaches. Some aspects related to this study, especially global and multiobjective optimization, were discussed in a more detailed level. In the next chapter we discuss the basic principles of operation of internal combustion engine, and from

that basis we focus our study to a specified part of engine design problem.

3 BASIC OPERATION OF INTERNAL COMBUSTION ENGINES

Internal combustion (IC) engine converts chemical energy of fuel into mechanical energy. This conversion is done by burning the fuel, which converts chemical energy into thermal energy. This in turn raises the temperature and the pressure of the gases inside the cylinder. The pressure is turned into mechanical energy by means of linkage systems which are connected to a rotating shaft, which, in turn, is used to take power out of the engine. Usually this shaft is connected to a transmission or power train which directs the energy to its desired final use.

A vast majority of the IC engines used in vehicles (such as cars, motorcycles, mopeds, snowmobiles, water jets, etc) for personal transportation and recreational use fall into two main categories, namely spark ignited 2-stroke and 4-stroke engines. In the following, we present the basic operation principles of these two main types of IC engines.

In addition to the two main types, there exists also other types of IC engines, for example compression ignited (Diesel) engines, and rotating piston (Wankel) engines. In Section 3.3 we restrict our consideration to 2-stroke engines for the rest of this study. This chapter is loosely based on [55].

3.1 4-stroke engine cycle

In a 4-stroke engine, one full working cycle consists of two engine shaft rotations, resulting in four upward and downward piston strokes, hence the name 4-stroke engine.

In Figure 12, the working principle of a 4-stroke engine is illustrated with four images, each of them representing one stroke. Explanations for the strokes are given below.

1. First stroke: intake stroke (Figure 12, a). The piston travels down while the intake valve is open and the exhaust valve is closed. Free volume in

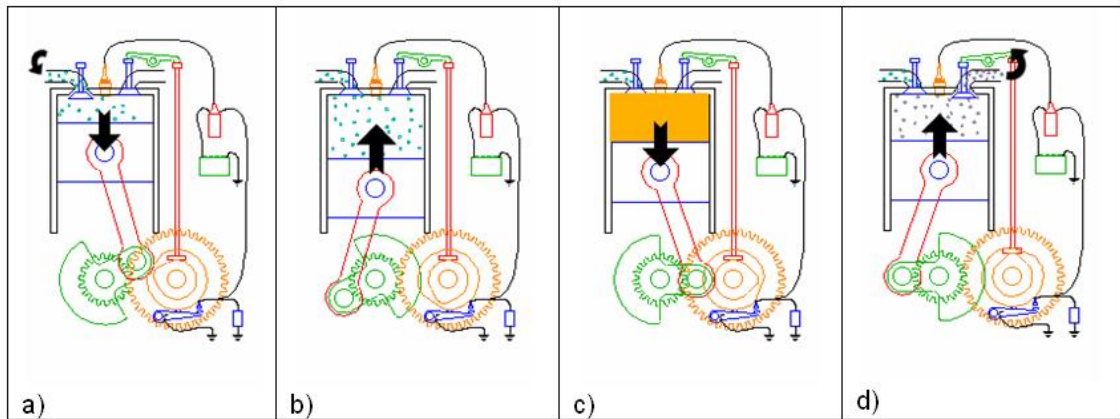


FIGURE 12 4-stroke engine cycle. Figures from [34].

the cylinder increases, and this creates a pressure difference between the cylinder and the atmosphere. As a result of that, fresh air charge flows into the cylinder and fuel is injected along the way to create a burnable mixture.

2. Second stroke: compression (Figure 12, b). When the piston reaches a bottom dead centre (BDC), the intake valve is closed (as is also the exhaust valve), and the piston starts to travel up. The volume in the cylinder decreases, and the pressure and the temperature of the air-fuel charge increases.
3. Combustion (Figure 12, c). Combustion of the compressed air-fuel charge occurs within a relatively short time period around the top dead centre (TDC). Combustion starts when the charge is ignited by a spark just before the compression stroke ends and continues a while to power stroke while gases expand. At the combustion phase the pressure and the temperature inside the cylinder rise drastically.
4. Third stroke: power stroke (Figure 12, c). The pressure created by the combustion pushes the piston down, and this produces the power which is seen as the engine output power. The gases inside the cylinder expand, while the pressure and the temperature decrease.
5. Fourth stroke: exhaust stroke (Figure 12, d). Before the piston reaches the bottom dead center (BDC), the exhaust valve is opened and the high pressure exhaust gas flows out. This is called an exhaust blowdown period. After the blowdown there still remain exhaust gases inside the cylinder, and while the piston moves up, it pushes the exhaust gases out via the exhaust valve. This completes the working cycle, and the process continues with a new intake stroke, etc.

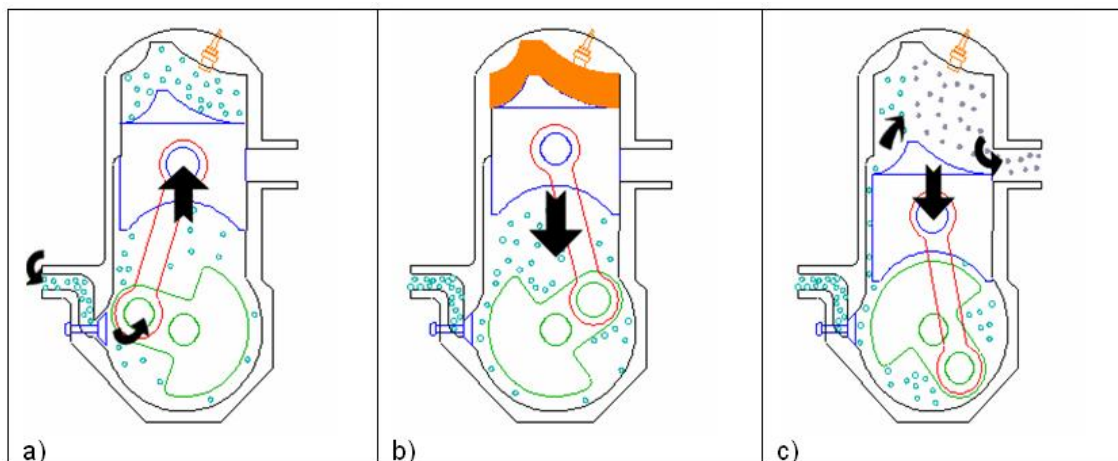


FIGURE 13 2-stroke engine cycle. Figures from [34].

3.2 2-stroke engine cycle

In a 2-stroke engine one full working cycle consists of one engine shaft rotation, resulting in two upward and downward piston strokes, hence the name 2-stroke engine.

In Figure 13, the working principle of a 2-stroke engine is illustrated with three images, of which the first two ones represent two actual strokes. The third image represents a scavenging phase. Explanations for the strokes are given below.

1. First stroke: compression (Figure 13, a). With all the valves (or to be more accurate in case of the 2-stroke engine, the ports which are carved to the cylinder wall) closed, the piston travels up and compresses the air/fuel charge into a higher pressure and temperature.
2. Combustion (Figure 13, b). Same as in the case of 4-stroke.
3. Second stroke (Figure 13, b): Power stroke, same as in the case of 4-stroke.
4. Exhaust blowdown (not clearly visible in the figures). In a 2-stroke engine the power stroke ends when the exhaust port opens and the piston is approximately half way down. This starts an exhaust blowdown period, after which the cylinder still remains filled with the lower pressure exhaust gas.
5. Scavenging (Figure 13, c). When the piston is approximately 3/4 down of the stroke length from TDC, the scavenging ports are opened. While the piston has travelled down, the volume under the piston (crankcase) has decreased, and the pressure there is higher than in the cylinder which is now open to the atmosphere via the exhaust port. A flow via the scavenging ports sweeps the burned charge out of the cylinder and fills it with the

fresh air-fuel charge. The scavenging process continues until the scavenging ports are closed and the piston is approximately $1/4$ of the stroke length up after the bottom dead centre (BDC).

6. Intake (Figure 13, c). While the piston goes up, the volume in the crankcase under the piston is increased and the pressure drops. After the piston is over halfway up towards the top dead centre (TDC), the intake port opens, and the fresh charge flows to the crankcase.

As we can see, in the 2-stroke engine, the volume below the piston (crankcase) is used efficiently as a pump unlike in the 4-stroke, where only the volume above the piston is effective, and different strokes are needed to draw the fresh charge in and push the burnt charge out. Dual action in the 2-stroke engine allows reduction of the strokes from four to two. Thus, the power strokes appear two times more frequently, which is the main reason for the high power output of 2-stroke engines, when compared to their 4-stroke counterparts.

3.3 Focus of this study

In this section, we set the goal of this study in general, and then focus it to a particular engine design. Although it would be beneficial to solve the engine design problem from scratch in its entirety, this approach is very demanding because the problem would be computationally very demanding (high dimensionality) to solve. So, it is more reasonable to solve the distinct subproblems of the whole problem separately. Also, in practice, while some fundamental parts of the engine may remain unchanged when some minor mechanical modifications are done, these modifications may have profound effects on the engine behavior, as discussed in following paragraphs. In the future, an increase in the computational power of computers may enable handling of more complex entities, and also give more profound understanding of phenomena itself. Before that happens, however, it is crucial to understand the functionality of each separate subproblem.

For the reasons above we focus this study on the optimization of a selected part of the engine design process. This focusing is not done on the expense of generality; the results can also be generalized to other similar problems.

Design aspects and problems of IC engines

There is a wide variety of applications where a different engine may be needed, ranging from small 30 cubic centimetres and 1 horsepower chainsaw engines to huge marine diesels capable of producing some 100 000 horsepower and a displacement of some 22 000 litres! It is self-evident that many properties of those engines are totally different. Properties of one particular engine are defined by several design factors such as displacement (cylinder volume), cylinder bore and piston stroke, size, number and location of valves or ports, camshaft tim-

ing, compression ratio, inlet and exhaust ducting (with their resonance effects on engine breathing), ignition timing, fuel injection timing, etc. Values for all these factors should be given so that the given needs are met. For further details on engine design factors, see [55].

Today it is not necessary to actually build the designed engine to evaluate its performance. Instead, there exists a variety of engine simulator packages capable of producing, for example, power, torque and emission outputs with acceptable accuracy (assuming that the underlying mathematical models are sound) for the desired engine, once the design parameter values are given. From the simulator outputs it is possible to derive the "goodness" of that particular engine design. Use of simulators greatly reduces the need of manufacturing different costly prototypes. However, it is important to notice, that in the final phase of the design, the results must be verified using real mechanical prototypes.

Although simulators can greatly reduce the time needed to attain results related to any given design, the task of selecting proper values for the design variables for the application in hand remains still in human domain. This is problematic, since it is almost impossible to systematically control dozens of variables and their delicate interactions. Therefore, the need for a tool to automatically select values or to assist in the selection of values for the design variables is evident.

Goal of this study

As described above, there is a need for a tool to aid in selecting proper values for the design variables to produce acceptable or even optimal designs. Using the trial and error method mentioned in the first chapter, a result is considered good if it is acceptable, i.e., it fulfills some given performance requirements. As a result of this, a "good" design may or may not be close to the real optimal design (the best possible design with regard to the given performance indicators) which would be possible to achieve only if the designer could somehow select proper values for the design variables.

The goal of this study is to create a system where the design process of an engine is shifted to a more abstract level, and the design is actually optimal or at least very close to the optimal with regard to the given requirements. This is achieved by using tools and methods of global, multiobjective and interactive optimization.

With this system, designing is no longer done by adjusting the values of separate design variables and trying to see their effect on the end result, but rather describing the desired end result in a higher level. This description can be done, for example, by giving graphically desired power or torque curves (power or torque plotted against engine rpm), or defining some mathematical formula to characterize the goodness of the engine, and using this formula as the target of the optimization process.

Practical design variables

In real life, where production and design costs must be taken strictly into account, there is a tendency to minimize the variety of different components needed for a new engine. Thus, no new engine is designed from scratch for every application, but instead, some base engine is modified to meet the standards currently posed by altering some of the auxiliary parts.

Easily adjustable auxiliary parts are, for example, the inlet manifold and the exhaust ducting, carburation, fuel injection and ignition timing changes. If even more changes in the engine performance are needed, it is possible to alter some easily changeable parts of the engine, like the cylinder head, camshaft, or valves. This reasoning holds true especially as regards recreational vehicles, such as snowmobiles, where one basic engine is used in a variety of models.

To allow possible empirical tests in the future, this study is limited to adjust those design variables that do not require altering of the basic engine concept, that is, internals of the engine. As were seen above, this limitation is justified.

Practical engine concept

In this study, we want to alter the behavior of the current engine design without altering its internal design. 2-stroke engines are known to be very sensitive to changes in the external ducting. For example, the lack of a proper exhaust pipe may easily drop engine performance by more than 50% [12], and different exhaust pipes can also alter the engine performance dramatically.

Although 4-stroke engines are widely used in car industry, this is not the case with recreational vehicles, where emphasis is strictly on performance, weight and simplicity. By these preferences, 2-stroke engines are superior to their 4-stroke counterparts. Furthermore, the exhaust pipe of a 2-stroke engine consists of a series of converging and diverging cones, and as such it is a rather simple and cheap device to fabricate (keeping in mind possible empirical tests in future).

For its sensitivity to changes in external ducting the 2-stroke engine is a very suitable target for engine optimization, and it is therefore used as the example in this study. More accurately, the shape and the dimensions of the exhaust pipes are altered to optimize the performance of a given engine.

3.4 How external ducting works

The usability of 2-stroke engines still in modern days is largely owing to the use of properly designed exhaust pipes, or expansion chambers as they are also called. Without the use of exhaust pipes, 2-stroke engines would be inferior to their 4-stroke counterparts, in terms of performance and economy.

On a general level, the exhaust pipe is designed as a four section device (see Figure 14):

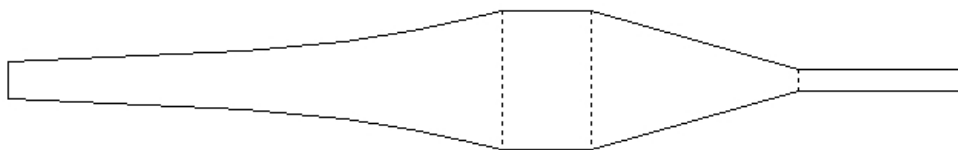


FIGURE 14 Basic pipe shape consisting of four sections: diffuser (aggregated with header pipe), belly, baffle cone and stinger.

- the first section is a horn shaped section, known as a diffuser. A factor defining how trumpet-like the diffuser is, is called a horn coefficient. Sometimes the diffuser is represented as an aggregate of a header pipe and the diffuser. In this case, the header pipe refers to the first mildly diverging part of the diffuser. In real life, the diffuser often consists of a series of diverging cones (for example below in Figure 15 the diffuser consists of 3 cones), or the cross section of the diffuser may implement the continuous shape of the exponential curve seen in Figure 14.
- the second section is a parallel walled part, known as a belly section (this section may also be missing if its length is 0).
- the third section is a converging cone, known as a baffle cone.
- after the baffle cone there is fourth part, a gas outlet, that is known as a stinger. It functions merely as a bleed resistor to keep a certain pressure inside the exhaust pipe.

An exhaust pipe harnesses energy of erupting pressure wave first when the exhaust port opens to aid scavenging and evacuation of the burned mixture out of the cylinder. Also just before the exhaust port closes, a reflected pressure wave pushes the escaped fresh charge back to the cylinder. Thus, the compression phase starts with the excess of the air/fuel mixture and the volumetric efficiency of the engine is increased. Because of this the use of the exhaust pipe can be seen as some sort of supercharging.

The functional operation of the exhaust pipe is based on the basic principles of the reflection of waves [12] in parallel walled finite pipes in the following way:

- when a positive amplitude (pressure) wave travels along the pipe, and reaches the open end of this pipe, it is reflected back, and its sign is negated. This means that the pressure wave is reflected back as a rarefaction or a suction wave.
- when the wave travels along the pipe, and reaches the closed end of this pipe, it is reflected back, but it keeps its original sign. Thus the pressure wave is reflected back as a pressure wave.

Figure 15 composed of seven images illustrates the propagation of the original pressure wave released by the exhaust port opening and the reflected waves. The

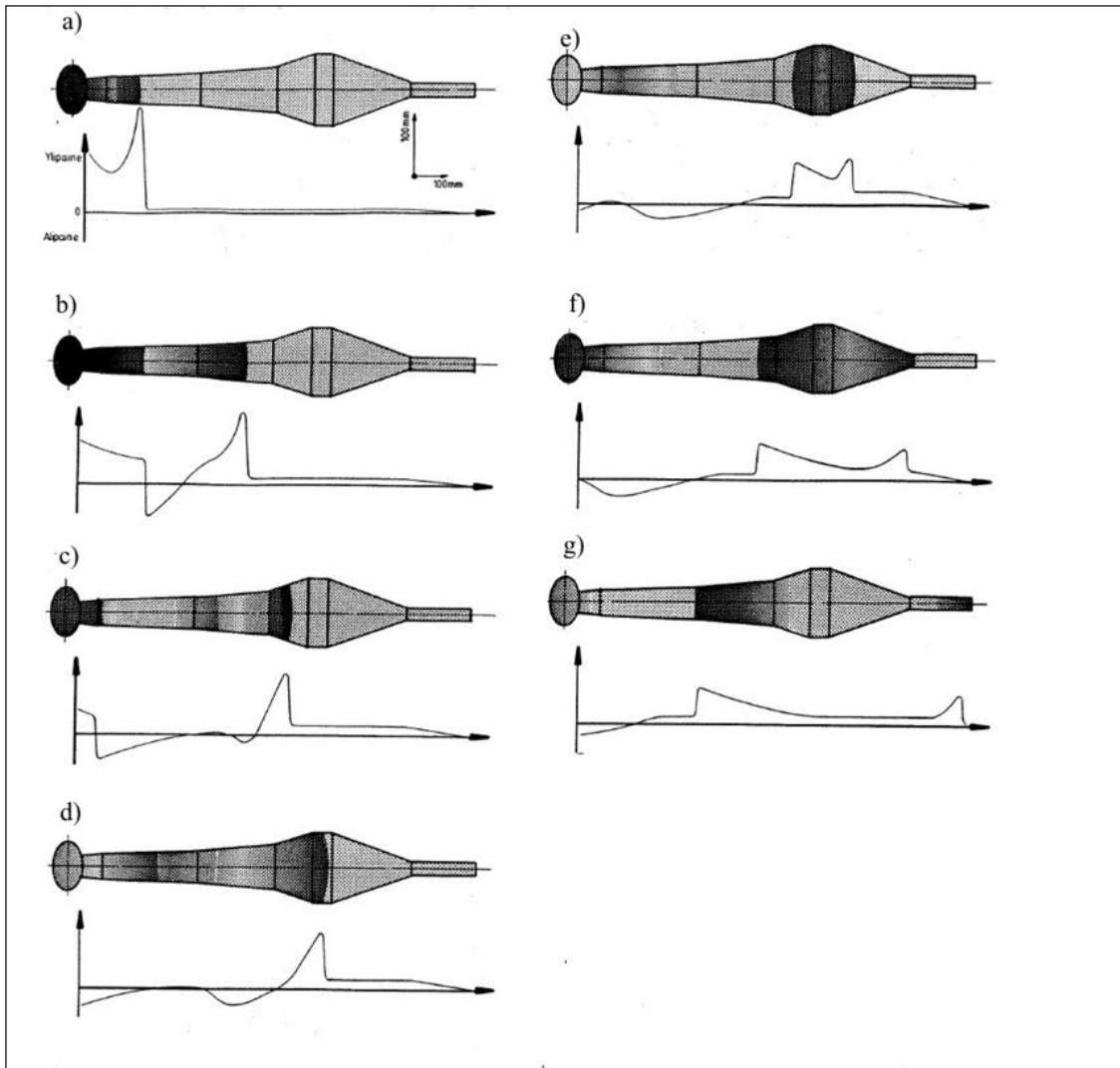


FIGURE 15 Waves and pressure distribution inside the exhaust pipe. Figures from [62].

tone value represents the pressure inside the pipe, the darker area represents high pressure whereas the lighter area represents a low pressure i.e. suction. The diagram below each image shows pressure distribution along the pipe length at the current moment, while the reference pressure level (atmospheric pressure) is the same as the x-axis. Below is a short explanation for each image.

In Figure 15 a), the exhaust port has just opened and a very powerful positive pressure wave is released to the exhaust pipe. The wave front travels with the speed of sound, which is in ambient circumstances inside the exhaust pipe (pressure, temperature, gas composition) approximately 500 meters per second.

In Figures 15 b) and c), the wave front propagates along the first parts of the expansion chamber, the header pipe and the diffuser. While the positive pressure wave proceeds along the diffuser, it reflects continuously a negative rarefaction wave backwards, and can be considered as the continuous open end of the pipe. The amplitude of this wave is dictated by the steepness of the diffuser at each location.

The shape of the diffuser is designed to produce a reflected rarefaction wave, which causes suction to the cylinder and thus helps the scavenging process to drag a fresh air-fuel charge from the crankcase up to the cylinder. The steepness of the first part of the pipe (header) is rather gentle, and it only lets the exhaust blowdown happen with a minimal resistance. After the blowdown, the scavenging ports are opened and in this phase suction can help the scavenging process. To create a consistent suction the diffuser wall is made to open more and more steeply until the scavenging process is finished. The diffuser cannot be made infinitely steep in the hope of creating enormous suction, because the original pressure wave contains only some limited amount of energy to be dissipated in the process of scavenging and "supercharging" the cylinder.

In Figure 15 d), the wave travels along the parallel walled and thickest section of the exhaust pipe, the belly section. The length of the diffuser is designed so that the suction ceases when the scavenging phase of the cylinder is over (when transfer ports are closed) and there is no longer need for the suction. As a by-product of the scavenging process some fresh air-fuel charge is spilled out of the cylinder to the exhaust passage. In this phase the wave experiences no changes, and the belly section is used merely to maintain a proper length for the exhaust pipe. The length of the pipe is a crucial factor since it dictates the timing of all the wave actions which must be in harmony with the engine port geometry and the desired running speed of the engine.

In Figures 15 e) and f), after the belly, the pressure wave hits the converging cone, the baffle cone, which acts analogously to the continuous closed end of the pipe. The amplitude of the wave has slightly decreased, since it has lost energy in the diffuser. The pressure wave does not change its sign, and it is reflected back from the baffle cone as a pressure wave. The duration and amplitude of the reflected pressure wave is dictated by the length and steepness of the converging cone.

In Figure 15 g) the reflected pressure wave travels backwards towards the cylinder. In this phase the scavenging ports are closed, and the exhaust port remains still open. The piston is moving up and the volume in the cylinder is decreasing. This further assists spilling of the fresh charge out of the cylinder, which could degrade the performance in several ways. With the use of a proper exhaust pipe, spilling of the charge is not harmful, since the reflected pressure wave arrives and pushes the fresh charge back to the cylinder while the upward moving piston closes the exhaust port. An excess of the air-fuel charge is trapped inside the cylinder, and this improves the volumetric efficiency of the engine. Power, economy and emissions are improved.

In general steepnesses and lengths of the cones determine the amplitudes and durations of the reflected waves. The total length of the exhaust pipe is selected to work best at some operating speed of the engine. To change this speed it is possible to scale all the lengths of the sections within the pipe proportionally to achieve the desired speed.

The steepness and length of the diffuser and the baffle cone must be selected carefully to maintain harmony: the diffuser cannot waste too much energy to help

the scavenging, because in that case there is no energy left to stuff the spilled charge back to the cylinder. And, on the other hand, stuffing the charge ragingly back would be of no benefit if there is no spilled charge in the first place.

In this chapter we have discussed the basic principles of both 2- and 4-stroke internal combustion engines. We have also focused our design problem to a particular subproblem, namely we control a 2-stroke engine behavior with changes in external ducting (i.e., exhaust pipe). In the next chapter we first discuss the layout of the optimization system as a whole, after which we concentrate on the separate modules of the system. We discuss engine simulator, introduce some models to represent exhaust pipe shapes, and finally introduce several different objective functions, i.e. ways to calculate goodness of a particular engine design from simulator output files.

4 OPTIMIZATION TASK OF AN INTERNAL COMBUSTION ENGINE

For optimization one must formulate an optimization problem including an objective function depending on design variables, and to solve it one needs an optimization algorithm. The optimization algorithm explores the surface formed by the objective function values (in the search space) to identify the deepest valley, which is a global minimum, and also the design with optimal values.

In our case the general layout for the optimization system given in Figure 1 is implemented as follows. Objective function values are derived from the output files of a simulator, which is software mimicking behavior of a specified engine configuration. The optimization algorithm selects and passes on the design variables defining the current engine configuration. The design variables are converted into a format understandable to the simulator. The simulator produces output files that contain, for example, power, torque and fuel consumption values for that particular engine design at several different running speeds. From this data, objective function values are derived using methods to be discussed in the following sections.

Within this study the engine design problem refers to the selection of the dimensions for the exhaust pipe for a 2-stroke engine. These dimensions cover all the lengths, diameters and angles for the cones the exhaust pipe consists of. Varying amounts of design variables are used to control the pipe models, which depict the true shape of the exhaust pipe. Different pipe models and how they are converted to simulator input files are discussed in Section 4.3.

4.1 Framework of the optimization system

To implement a functional simulation based optimization system for optimization of internal combustion engine, the following components are required:

- a simulator which models the actions of the engine and is capable of producing the data from which the goodness of the current design can be derived

from.

- software to calculate goodness (objective function values) of the current design using output data of the simulator.
- software to convert the design parameter values given by the optimization algorithm to a format which is readable for the simulator.
- an effective (by means of objective function evaluations) optimization algorithm.
- in case of a multiobjective optimization problem method to convert multiple objectives into the single objective.
- software to link all the above together to create a functional optimization system.

All of these components are discussed more in detail in the following sections.

4.2 The engine simulator

The base of an engine simulator is a bunch of mathematical models (e.g. flow model, combustion model, scavenging model) depicting different physical and chemical phenomena taking place inside the engine. By solving numerically these mathematical models the engine simulator imitates physical (gas exchange flows, compression, heat conduction) and chemical (burn period, emissions) processes in the engine.

The purpose of the simulator is to give information about the engine without a need to actually build one. Simulators are rapid and rather cheap tools to inspect properties of an engine before creating a prototype of it. For the simulator, the physical geometry of the engine must be depicted with a high accuracy. This includes all the basic dimensions of the engine such as bore, stroke, cylinder volume, crankcase volume, head volume, connecting rod length, sizes and shapes of gas exchange ports and measures of the external ducting. Operating conditions of the engine must also be defined, for example, ambient air pressure, ignition timing, fuel characteristics, air/fuel ratio etc. From that input information the simulator generates the output data describing the engine behavior, for example at different running speeds. The output data contains information such as power, torque, fuel consumption and exhaust gas composition.

For this study, the simulator was selected keeping in mind the fact that for the planned end use it was necessary to consider especially the time consumed during a single simulation run and the possibility to link the simulator with the optimization tool. MOTA 6.1 by J. van Leersum [41] and Ian Williams Tuning [69] was selected as a simulator. It is widely used among the 2-stroke enthusiasts and the semi-professional engine designers. Execution time for one single simulation run is typically few minutes using a modern PC (AMD Athlon 2.09 GHz,

1.0 GB of RAM), depending on the complexity of the pipe design and the grid granularity (total amount of rpm steps) for end results. MOTA is also based on sound scientific principles [41]. This cannot be guaranteed in the case of some less known simulator.

The equations in MOTA describing conservation of heat, momentum, and internal energy of quasi one-dimensional compressible flow in a duct of varying cross-section area are known as the Euler equations. The numerical method chosen to solve those equations uses a flux conservative Lax-Wendroff type algorithm modified by the use of flux splitting and flux limiters. This enables the model to cope with specific very high output engines, that is having high engine power with relation to the engine displacement [41]. These types of engines are the ones that will benefit the most of optimization, because their performance is already close to the limit and not so easily treatable by other means.

The MOTA simulator model consists of several submodels depicting the main processes which occur in a 2-stroke engine, including changes in gas properties, duct flow model with flux splitting and flux limiters, the purity calculation of the gas inside the cylinder, the scavenging model and the combustion model. Because a more detailed discussion of these complex submodels is out of the scope of this study, and the simulator is used as a black box from the perspective of the optimization system, we refer to [41] for a full coverage of the topic.

In order to use the simulator successfully in the context of optimization, the simulator itself should produce results with sufficiently good agreement with real life experimental data. Otherwise, results of the optimization system are not applicable to real world problems. As shown in one of the examples in [41], the MOTA simulation model predicts the position and magnitude of the peak power for a real Parilla T75 100 cc karting engine well, indicating that the wave speed modelling mechanism is very good. The slight discrepancies seen in Figure 16 between the experimental data and the model predictions away from the position of the peak power could be due to the dynamometer error, which was in the order of under 8 % during a number of different runs, and uncertainties in the following parameters used in the MOTA simulation model:

- the parameters used in the combustion model,
- the scavenging parameters describing zones of fresh charge, mixed charge and short circuiting of fresh charge to the exhaust stream,
- the combustion efficiency,
- the air-fuel ratio,
- the assumed discharge coefficients for various ports of the engine.

Considering the fact that all the parameters listed above, except the last one, vary to some extent with the engine speed (and in [41] estimated values for these parameters were taken from the literature, and no attempt was made to "tune" these

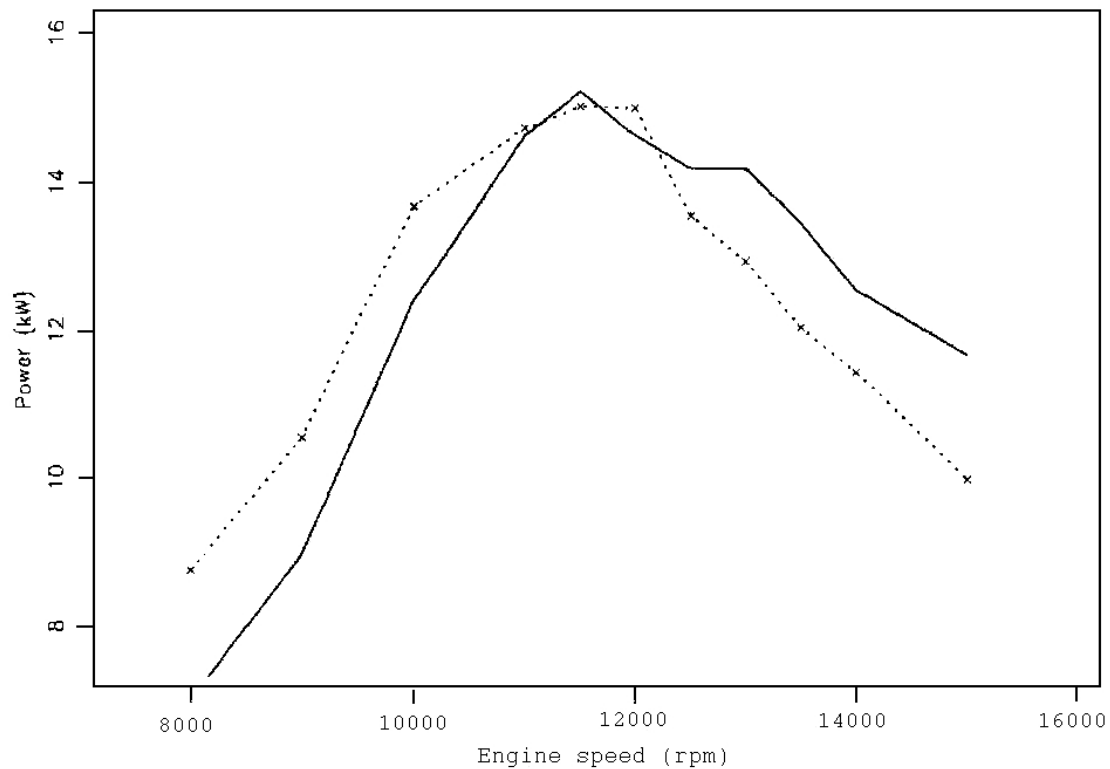


FIGURE 16 Simulated and experimental dynamometer results for the Parilla engine. Crosses: simulated, solid line: experiment. Figure based on [41].

values to match the experimental results) the agreement between the dynamometer runs and the results predicted by the model is quite good, as seen also in Figure 16. This is regarded sufficient for our purposes. It remains as a concern of a further study to experimentally evaluate the quality of the solutions produced within this study.

Engine configuration is given to the simulator as a single plain text input file in ASCII-format. The input file contains information about mechanical engine configuration (bore, stroke, port dimensions in cylinder wall, exhaust and intake channel lengths, carburetor size, exhaust pipe dimensions, etc.) and its ambient operating conditions (air pressure and temperature, ignition timing, air/fuel ratio). The input file contains also information about user specified simulator run parameters, such as the number of engine speeds (for which the values are calculated), initial engine speed and speed increment step size. Exhaust pipe dimensions for each design are inserted to the input file, which otherwise remains intact. These dimensions are given as a list of lengths for each separate section, and as another list containing start/end diameters for each section.

Output of the simulation run is stored in plain ASCII-files, which can easily be parsed for objective function evaluation purposes. Output of the simulator contains all the necessary information such as power, torque, bmep and fuel consumption for all rpm steps that were used during the simulation run.

4.3 Modeling the engine configuration

In this study engine configuration modeling refers to a dual technique for the exhaust pipe shape representation and it should not be mixed with the simulator engine model. On the one hand engine configuration model is used to produce general and modifiable exhaust pipe shape using as few design variables as possible (to lower the dimension of the optimization problem). On the other hand, the exhaust pipe shape which is created using as few design variables as possible must be converted to a form which is usable for the simulator (list of lengths and diameters). For example, a diffuser shape may be represented by a continuous curve. This continuous shape must be converted, for the simulator, to separate sections whose start diameter, end diameter, and length is known.

Exhaust pipe (see Figure 14 and Section 3.4) consists basically only of few parts: header, diffuser, belly, baffle cone and stinger. Often when the header and the diffuser are aggregated, this aggregate is referred to, somewhat confusingly, as diffuser. The stinger is the last part of the pipe, although, in real life applications, a muffler is often fitted after the stinger to absorb noise. If properly constructed, the muffler has no effect whatsoever on the exhaust pipe, thus it is excluded from this study.

Earlier, expansion chambers often were so-called "two-cone chambers" meaning that there was, after the parallel walled header pipe, only one shallow cone which functioned as a diffuser and another steeper converging cone, namely a baffle cone. Additionally, between the two cones there was often a parallel walled belly section. This sort of pipe is detrimental to gas flow because it causes turbulence at the sharp joint of the header and the diffuser, and thus wastes wave energy. This is the reason why today header pipe and diffuser form a single gradually steepening cone, or to be more accurate, a horn. The baffle cone may have straight or outcurved (convex) walls.

Depending on the manufacturing technology (pressing, hammering, welding out of cones) the expansion chamber may have a smooth overall shape or it may be an approximation of the original shape made by series of cones and cylinders rolled out of sheet metal. For the manufacturing and other purposes the original continuous pipe shape is often discretized to suitable pieces. By discretization we refer to the process where the continuous shape of the pipe is divided into a discrete number of pieces. For example, the diffuser section in Figure 14 could be divided into five sections in a manner that most efficiently retains the original shape of the diffuser (see Figure 18).

Likewise, for the simulator the exhaust pipe shape must be represented as a series of sections for which the start and end diameters and lengths must be given. For some models below, one needs to carefully consider how continuous shape can be discretized in such a way as to represent the original shape with sufficient fidelity.

From the optimization point of view, flexibility of the model used is of importance. Even the best of the optimization algorithms is capable of finding only

as good a solution as the model in question is capable of presenting. As a simplified example of this we can consider exhaust pipe shape that is modelled using only cylindrical shape, with two design variables, length and diameter. In this case the result would be extremely poor compared to properly shaped exhaust pipes. As seen in Chapter 5, some of the models seem to lack sufficient flexibility.

Let us next discuss several possibilities for modeling engine configuration i.e. the exhaust pipe shape.

4.3.1 Blair model

Blair model given in [12] is a natural starting point for modeling the exhaust pipe shape since it is widely used among the 2-stroke enthusiasts for calculating the dimensions of the exhaust pipe for a given engine. While designing an exhaust pipe, the total length of the exhaust pipe (called tuned length) is the most significant factor which determines the location of the power peak (maximum power) of the engine [12]. The shape of the pipe is used to adjust the shape of the power curve. Because of this it is reasonable to link the lengths of the different pipe sections to the total length of the pipe. Unlike in Figure 14, the diffuser is not represented as a continuous curve, but it is replaced with far coarser representation of the horn function consisting only of three pieces (see Figure 17, L2-L4) with predefined length ratios. In addition, lengths for the belly section (L5) and the baffle cone (L6) are implicitly given as ratios of the tuned length of the whole pipe, so the designer cannot adjust these explicitly.

TABLE 2 Design variables and their bounds for Blair model.

Design variable	Bounds
L_t , total length of pipe (mm)	Bounds depend on the application's possible rpm scale and are calculated from a simple formula which converts rpm to length [12].
k_1 , initial pipe diameter coefficient	Bounded between 1.05 - 1.125.
k_2 , mid section diameter coefficient	Bounded between 2.125 - 3.125 times of header diameter at cylinder flange.
k_3 , tail pipe diameter coefficient	Bounded between 0.5 - 0.7 times of header diameter at cylinder flange.
k_h , horn coefficient	Bounded between 1.25 - 2.00.

The Blair model has five design variables as given in Table 2. The bounds for design variable values are given in [12]. The first of the variables is total length of the pipe L_t . The other four variables are control parameters for formulae given in [12] and also presented at the end of this section. Initial pipe diameter coefficient k_1 determines how conical the first part of the pipe is (see Figure 17, part L1). Mid section diameter coefficient k_2 determines the maximum diameter of the pipe, i.e., diameter of the section L5. The tail pipe diameter coefficient k_3 determines the

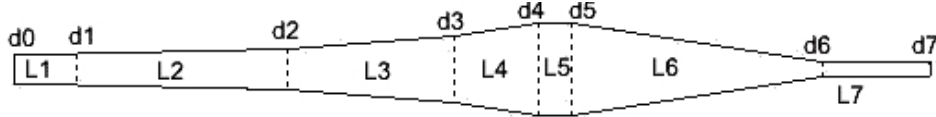


FIGURE 17 Examples of exhaust pipe shape produced using the Blair model. Shape is presented to simulator using seven pieces, L1-L7.

diameter of the end pipe, stinger L7. Horn coefficient k_h determines how hornlike the diffuser is. Thus diameters for sections L1, L2, L3, L4 and L5 are determined indirectly by coefficients k_1 , k_2 and k_h . Pipe dimensions are determined using the formulae given in [12]. In this study these are slightly modified. In the original formulae the total length of the pipe is determined using information such as desired running speed of the engine, specific power output of the engine and the port layout of the cylinder. Within this context calculus for the length is omitted, because the length itself is one design variable and thus value for it is given by the optimization algorithm, and it is bounded between the designer specified interval. Thus, the designer must decide the minimum and maximum values for the pipe length using a simple formula which converts rpm to pipe length. The formula is given in [12].

When using the Blair model, the model itself represents the pipe as separate sections. Thus the form of the Blair model can be presented similarly to the simulator and there is no discretization error between the engine configuration model and simulator input shape.

Symbols in the formulae below refer to diameters (d_i) and lengths (L_i) of the pieces in Figure 17, and variable values are the ones mentioned in Table 2. Constants are based on empirical values established in [12]. Lengths for the first pipe and diffuser are given as

$$L_1 = 0.1 \cdot L_t, \quad L_2 = 0.275 \cdot L_t, \quad L_3 = 0.183 \cdot L_t, \quad L_4 = 0.092 \cdot L_t \quad (7)$$

and lengths for midsection, rear cone and tail pipe are given as

$$L_5 = 0.11 \cdot L_t, \quad L_6 = 0.24 \cdot L_t, \quad L_7 = L_6. \quad (8)$$

It should be noted that the length L_t is comprised of lengths L_1, \dots, L_6 and, thus, the length of the tail pipe L_7 is not included in L_t .

Diameters for all sections except diffuser are

$$d_1 = k_1 \cdot d_0, \quad d_4 = k_2 \cdot d_0, \quad d_7 = k_3 \cdot d_0. \quad (9)$$

Diameters for diffuser sections are given as

$$d_2 = d_1 \cdot e^{X_{12}}, \quad d_3 = d_1 \cdot e^{X_{13}} \quad (10)$$

where

$$X_{12} = \left(\frac{L_2}{L_2 + L_3 + L_4} \right)^{k_h} \cdot \ln \left(\frac{d_4}{d_1} \right) \quad (11)$$

and

$$X_{13} = \left(\frac{L_2 + L_3}{L_2 + L_3 + L_4} \right)^{k_h} \cdot \ln \left(\frac{d_4}{d_1} \right). \quad (12)$$

Although the Blair model is widely used among the 2-stroke enthusiasts to calculate the dimensions for the exhaust pipe, it has drawbacks. Lengths L_i are readily given, and they can not be adjusted. Also the shape of the diffuser is rather coarse, thus it cannot implement a preferred horn shape with high fidelity. For these reasons, chances to alter engine properties are somewhat limited. Thus, in the following subsections we introduce different, more flexible models to represent exhaust pipe shapes.

4.3.2 Horn model

We have developed the Horn model as a generalization of the Blair model. In contrast to the Blair model, the Horn model implements a continuous diffuser section using Equations (13) and (14) given later in this section, a parallel walled belly section and a single piece baffle cone. The total length of the pipe is one design variable, as is the case with the Blair model, but the lengths for the diffuser, belly and baffle cone are given as fractions of the total length of the pipe. With this procedure one can also give the bounds for the design variables of the optimization algorithm as fractions of the total length of the exhaust pipe, instead of giving the bounds as the real lengths, which would vary from case to case. Thus, the only length which is given as an absolute value is the total length of the exhaust pipe, whose bounds can be determined relatively easily by using simple exhaust pipe formulas, found for example in [12].

When the exhaust pipe is designed as a three-section device as described above, six design variables are needed to fully determine all its physical dimensions. The design variables and their bounds are given in Table 3. The bounds for the variables are based on empirical values from [12] with the ranges slightly extended.

Handling length ratios of the diffuser and the belly/baffle is a little bit tricky, because the sum of the lengths of all the three pieces must equal the total length of the pipe. This is solved by giving, to each section, a minimum and maximum length as a proportion of the total length of the pipe, while all the proportional lengths must be equal to one (the real length of the pipe is scaled to equal 1). Design variables modulate the lengths of sections between minimum and maximum values.

Below we give the ranges for the proportions for each section of the exhaust pipe. The extremes of these ranges are later referred to as minimum and maximum values. Ranges are based on common sense, and they are expanded so that with the extreme values a pipe shape produced can be readily condemned unsatisfactory in a visual inspection. The sum of all proportional lengths must be equal to one.

- the diffuser may be 0.4 to 0.85 times the total length

TABLE 3 Design variables and their bounds.

Design variable	Bounds
x_1 , total length of pipe (mm)	The bounds depend on the application's possible rpm scale and are calculated from a simple formula which converts rpm to length [12].
x_2 , diameter of belly (fraction)	Bounded to 2 - 4 times of exhaust port flange diameter at cylinder.
x_3 , diameter of stinger (fraction)	Bounded to 0.5 - 0.7 times of exhaust port flange diameter at cylinder.
x_4 , length of diffuser (fraction)	Bounded between 0 - 1. See below how this is converted to a real length.
x_5 , ratio of lengths of belly and baffle cone (fraction)	Bounded between 0 - 1. See below how this is converted to a real length.
x_6 , horn coefficient for diffuser	Bounded between 0 - 2. Zero gives a conical diffuser, larger values result in an increasingly hornlike diffuser.

- the belly may be 0.0 to 0.45 times the total length
- the baffle cone may be 0.15 to 0.6 times the total length

From the design variable values the proportional length of the diffuser is determined first. Next we calculate what is left out for the belly when the diffuser and the baffle cone are in place. In some cases this causes the belly to disappear totally (when the value is 0). The sum of the proportional lengths of all the pieces must equal to 1. This is then scaled to equal the real length of the pipe.

Let us, as an example, consider a pipe with the total length of 1000 mm, while the diffuser proportional length = 0.55 and belly/baffle ratio = 0.6. Now the real length of the diffuser can be calculated as $\text{diffuser min.len.} + (\text{diffuser max.len.} - \text{diffuser min.len.}) \cdot \text{diffuser length variable value} \cdot \text{total length of pipe}$, which gives $(0.4 + (0.85-0.4) \cdot 0.55 \cdot 1000 = 647.5 \text{ mm})$. This is the same as the value of the diffuser length variable 0.55 scaled between the diffuser minimum and maximum lengths multiplied by the total length of the pipe.

Now there is total length - diffuser length ($1000-647.5 = 352.5 \text{ mm}$) left for the belly and the baffle together. This is divided by the value of 0.6 as the belly/baffle ratio indicates, resulting with the belly length of $352.5 \cdot 0.4 = 141 \text{ mm}$ and the baffle cone length of $352.5 \cdot 0.6 = 211.5 \text{ mm}$.

The Horn shape for the diffuser is calculated using Equations 13 and 14. These are generalized for the continuous case by the author of this text from the ones given in [12] and discussed also in Section 4.3.1.

For calculating all the diameters for the pipe, we must first calculate the diameter D_e of the thickest part of the pipe, where the diffuser ends and the pipe continues as a belly. It is calculated by multiplying the cylinder exhaust port

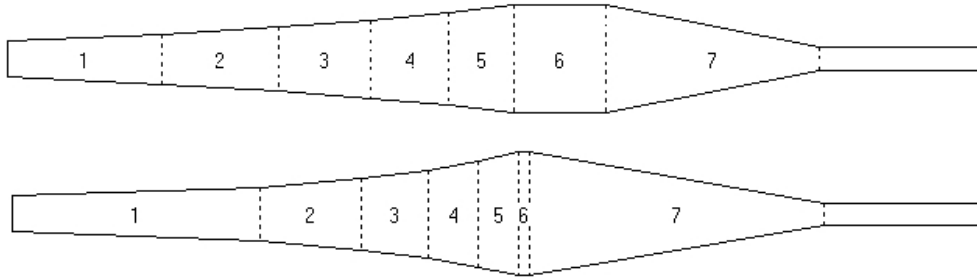


FIGURE 18 Two examples of exhaust pipe shapes produced using horn model. Shape is presented to simulator using seven pieces.

diameter at flange D_s (constant defined by physical engine) by design variable x_2 . Diameters for a horn shaped diffuser are calculated using the formula

$$D_l(l) = D_s \cdot e^X, \quad (13)$$

where

$$X = \left(\frac{l}{L} \right) \cdot \ln \frac{D_e}{D_s} \quad (14)$$

D_l = the diffuser diameter at location l

D_s = the start diameter of the diffuser (defined by engine configuration)

D_e = the end diameter of the diffuser, calculated using variable x_2

L = the length of the diffuser

k = the horn coefficient.

With this model, for a representation suitable for the simulator the diffuser is selected to be divided into five subsections to give sufficient fidelity to the horn shape. Each of these sections will have an equal difference in their diameters, and thus different lengths depending on the horn coefficient. This is a real world approximation of the exponential horn curve, given in Equations (13) and (14).

With the five subsections the diffuser follows the diffuser horn curve closely enough. This is also a usable form for the simulator, producing the whole pipe shape using seven different pieces. The first diameter of the pipe is constant, because it is always the same as the diameter of the exhaust channel at the joint of the cylinder and the exhaust pipe.

In Figure 18 two examples of different designs are presented. The effect of the horn coefficient is clearly visible in the five diffuser segments. The pipe on top has a small value for the horn coefficient whereas the pipe at the bottom has an essentially higher value.

The horn model has one drawback with its flexibility. Although the horn model is more flexible than the Blair model, it may still not be able to represent all the possible pipe shapes seen in real world applications. For example, the quite simple shape depicted in Figure 19 is not reachable with the horn model because of the multisection outcurved (convex) baffle cone. This drawback is real, because for example convex baffle cones are frequently seen in real life designs. The diffuser may also contain more complex shapes than the horn.

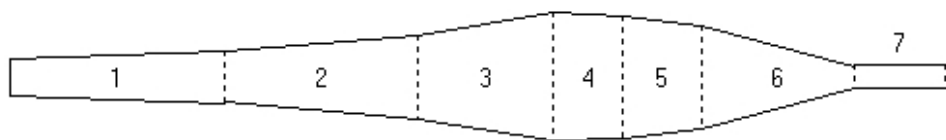


FIGURE 19 Inflexibility of the horn model. This shape of the exhaust pipe is not reachable using horn model.

4.3.3 Free Form Shape model

To overcome shortcomings of the horn model discussed in Section 4.3.2 we next develop a Free Form Shape (FFS) model. The most severe of the shortcomings of the horn model is its stiffness, i.e. it always uses a horn shaped diffuser (there cannot exist a shallower section just before the belly section for example), and the baffle cone is restricted to be a single cone without any outcurved (convex) or incurved (concave) shape.

In real life, baffle cones may sometimes be either convex or concave. Free form shape (FFS) restricts neither the convexity nor the concavity of the baffle. The FFS model consists of an envelope (see Figure 20) which is divided into ten sections of equal length. The envelope can be seen as a region wherein the pipe shape can change freely. Diameters for each section are defined with a factor between 0 and 1 which modulates the diameter inside the envelope of a given shape. Thus, in the upper part of Figure 20 the pipe shape with direct lines is caused by setting all the shape variables to zero (the minimum side of the envelope), whereas a bulky shape is caused by setting all the shape variables to one (the maximum side of the envelope). Minimum values for the envelope are produced by drawing a direct line from the exhaust flange to the outer edge of the stinger having a diameter of 0.55 times of the header. Maximum values are determined by setting proportionate lengths of the diffuser, belly, and baffle cones to 0.4, 0.4 and 0.2 times of total pipe length, respectively. The maximum diameter is 3.55 times the exhaust flange diameter and the maximum value for the stinger diameter is 0.7 times the exhaust flange diameter. All the previous diameter factors are based on empirical values given in [12] with the ranges slightly expanded.

In FFS there is no explicitly given diffuser, belly or baffle cone sections, all of these are implicitly defined by ten design variables. For this reason there are some restrictions to this model: adjacent diameters should increase up to the section where the maximum diameter of the pipe is reached, after that the section diameters must decrease. From the optimization point of view that kind of shape rule makes search space very fragmented. This fragmentation is illustrated by the fact that the shape rule filters out a massive amount ($> 99.5\%$) of randomly created design variable points. Shape rule is embedded into the FFS model and thus it is invisible from the perspective of optimization algorithm. The FFS model returns some predefined sufficiently poor objective function value in case there is a shape rule violation.

FFS model has eleven design variables given in Table 4. The first of these

TABLE 4 Design variables and their bounds for FFS model.

Design variable	Bounds
x_1 , total length of pipe (mm)	The bounds depend on the applications' possible rpm scale and are calculated from a simple formula which converts rpm to length [12].
x_2, \dots, x_{11} , diameters for each locations within envelope (fraction)	Bounded to 0 - 1 times of distance between upper and lower envelope diameters.

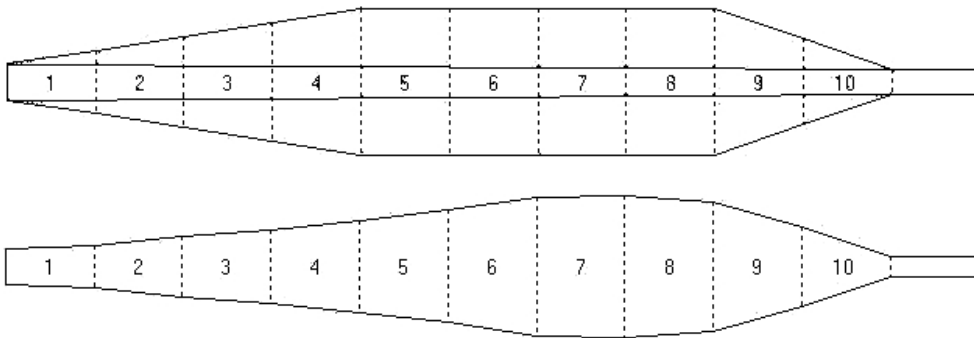


FIGURE 20 The shape envelope for the FFS model and one resulting shape. The shape is presented to the simulator using ten sections.

determines the total length of the pipe, and the bounds for it are calculated in a manner similar to that in the Blair and horn models. The other ten design variables are so-called shape variables that merely adjust the diameter of the pipe at each location of the shape envelope. The difference between the minimum and maximum envelope values is simply multiplied by the value of that particular design variable. Lengths for the pieces are not explicitly needed, because the total length of the pipe is divided into ten equally long sections. The first diameter of the pipe is constant, because it is always the same as the diameter of the exhaust channel at the joint of the cylinder and the exhaust pipe.

The exhaust pipe shape created using the FFS model is inherently suitable for the simulator, thus no separate conversion procedure is needed.

4.3.4 Bezier model

To combine the flexibility of the FFS model with the low design variable dimensionality of the horn and Blair models we next develop one more model, the Bezier model, which is capable of representing very dissimilar pipe shapes using a Bezier curve and only a few design variables.

In shape optimization literature, for example in [23], the shape is often represented (or parametrized) using either Bezier curves, or B-spline curves. The Bezier curve was formally presented in [10] by Pierre Bezier who was trying to

formulate representations for smooth surfaces suitable for modelling the car chassis. After that, Bezier curves have been a very common way to display smooth curves, both in computer graphics and mathematics. The degree of the polynomial curve defining a Bezier curve increases as the number of control points increases. To avoid excessively high order polynomials, a single Bezier curve with several control points can be divided into several low order Bezier curves, which is called a B-spline curve. As such, the B-spline curve is an extended version of the Bezier curve that consists of segments, and each of these segments can be viewed as an individual Bezier curve.

The main difference of the Bezier and B-spline curves is that no matter the degree of the Bezier curve, there is no local control property in a single Bezier curve [23]. Thus, all control points affect the whole curve, and this can be a disadvantage when designing smooth shapes with Bezier curves. However, in our study, this does not seem to be the problem, as seen in Figures 21, 22 and 23, and it can, in fact, be beneficial that a single Bezier curve with only four control points allows no sudden changes, which would be detrimental to the pressure / suction wave propagation, in exhaust pipe shape.

In the Bezier model the pipe shape is defined by the Bezier curve having unadjustable start $(c1_x, c1_y)$ and end $(c4_x, c4_y)$ control points at both ends of the pipe (header pipe upper edge and stinger start point upper edge) and two adjustable control points $(c2_x, c2_y)$ and $(c3_x, c3_y)$ (see Figure 21). By moving the control points, the pipe shape can be altered radically (see Figures 22 and 23). The control points may move only within their specific regions: C2 is allowed to move in the region marked with a double line, and C3 in the region marked with a single line. Region sizes are based on the authors' experiments, and they provide sufficient diversity for reachable pipe shapes. Regions are defined using total length of the pipe L_t as a practical unit of measure. The region for the control point C2 starts from the location $0.5 \cdot L_t$, the width is $0.7 \cdot L_t$, and the height $0.15 \cdot L_t$. The region for the control point C3 starts from the location $0.3 \cdot L_t$, the width is $0.5 \cdot L_t$, and the height $0.2 \cdot L_t$. The location of the control point inside a region is modulated via the design variables given in Table 5.

The difference to the FFS model is that Bezier produces always somewhat smooth shapes, which are generally appealing, but in some rare cases it could be important to be able to have sudden slightly thicker part for example in the header pipe to create a momentary stronger vacuum. However, it must be emphasized that these are kinds of forms that are extremely rare in real life, whereas smooth shapes are de-facto in almost all exhaust pipes.

The coordinates, that is, x and y values for a Bezier curve representing the pipe shape at location t are calculated using the following equations given in [29]:

$$\begin{aligned}
 x(t) = & (c1_x + t \cdot (-c1_x \cdot 3 + t \cdot (3 \cdot c1_x - c1_x \cdot t))) \\
 & + t \cdot (3 \cdot c2_x + t \cdot (-6 \cdot c2_x + c2_x \cdot 3 \cdot t)) \\
 & + t \cdot t \cdot (c3_x \cdot 3 - c3_x \cdot 3 \cdot t) + c4_x \cdot t^3
 \end{aligned} \tag{15}$$

and:

$$\begin{aligned}
 y(t) = & (c1_y + t \cdot (-c1_y \cdot 3 + t \cdot (3 \cdot c1_y - c1_y \cdot t))) \\
 & + t \cdot (3 \cdot c2_y + t \cdot (-6 \cdot c2_y + c2_y \cdot 3 \cdot t)) \\
 & + t \cdot t \cdot (c3_y \cdot 3 - c3_y \cdot 3 \cdot t) + c4_y \cdot t^3
 \end{aligned} \tag{16}$$

where

cn_x = x coordinate of control point n

cn_y = y coordinate of control point n

$0 \leq t \leq 1$

TABLE 5 Design variables and their bounds for the Bezier model.

Design variable	Bounds
L_t , total length of pipe (mm)	The bounds depend on the applications' possible rpm scale and are calculated from a simple formula which converts rpm to length [12].
$c2_x$, x -location of control point 1	Bounded between 0 - 1.
$c2_y$, y -location of control point 1	Bounded between 0 - 1.
$c3_x$, x -location of control point 2	Bounded between 0 - 1.
$c3_y$, y -location of control point 2	Bounded between 0 - 1.
k_t , tail pipe diameter coefficient	Bounded between 0.5 - 0.7 times of the header diameter at the cylinder flange.

The Bezier model has six design variables as given in Table 5. The first of them is the total length of the pipe, which is defined similarly to the models previously introduced. Four next variables are x and y coordinate locations for the two control points of the Bezier curve. The last design variable, tail pipe diameter coefficient determines the diameter of the end pipe, stinger.

Continuous Bezier curve shape is discretized to a piecewise representation for the simulator using six sections for the diffuser and three sections for the baffle cone. There is no separate belly section. The lengths for the diffuser sections are found by requiring that all the adjacent diffuser sections must have the same difference in their diameter. For example, if the header diameter at the exhaust flange is 30 mm and the diameter of the belly is 90 mm, then the total difference in their diameter is 60 mm. If the diffuser is divided into five sections, the diameter difference for the adjacent sections is 12 mm, and thus the set of diameters from the header to the belly is: 30, 42, 54, 66, 78, 90.

To retain sufficient fidelity to the original Bezier curve, the lengths for the baffle cone sections are determined in a somewhat more complex way. Here differences in diameters are not the same between each section, they are ratios of

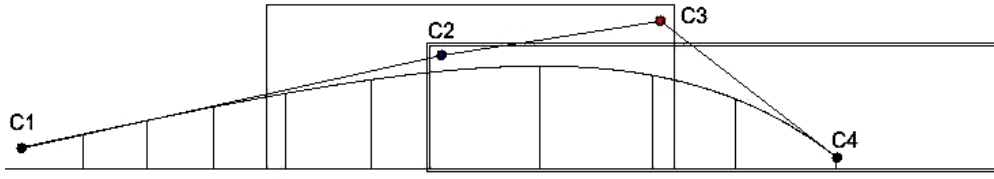


FIGURE 21 Bezier curve with two adjustable control points inside their respective boundary boxes.

sums of a geometric series. The sum S_r of the geometric series to the element r can be calculated (as any textbook of calculus would indicate) as

$$S_r = \frac{a \cdot (1 - q^r)}{1 - q} \quad (17)$$

where

a = first term, here 1.0,

q = ratio between adjacent elements, here 2.5,

r = ordinal number for the element where to sum is calculated, $r=1,2,3$.

Diameters for each of the baffle cone sections are calculated as

$$d_r = d_{max} - \frac{S_r}{S_R} \cdot (d_{max} - d_{min}) \quad (18)$$

where

d_r = diameter for baffle cone section r ,

d_{max} = maximum diameter of baffle (next to belly),

d_{min} = minimum diameter of baffle (next to stinger),

S_r = sum of geometric series to element r ,

S_R = sum of whole geometric series, where total number of baffle sections $R = 3$.

As an example, baffle cone with $d_{max} = 90$ mm and $d_{min} = 20$ mm results a in a descending set of diameters of 90, 83, 65 and 20 mm. Diameter differences for the baffle cone sections are calculated using (17) and (18) to produce lengthwise shortening series of baffles and to follow Bezier curve closely by having smaller diameter differences near the belly section. Lengthwise distribution of sections can be altered by changing the values of a , q and R . Here these values are based on the author's experience. See Figure 22 to see the arrangement of baffle cone sections.

4.4 Defining the objective function

This section is loosely based on [13], [18], [47] and [70]. For the optimization, one or more properties of the system to be improved must be identified to be improved. In this section we present five different properties of the engine or

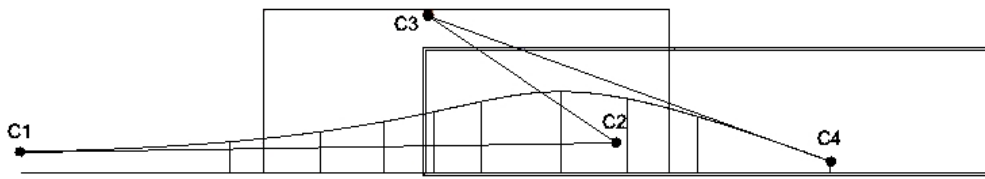


FIGURE 22 Example of smooth exhaust pipe shape created using Bezier model.

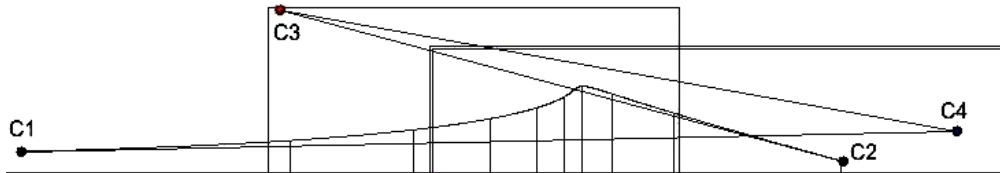


FIGURE 23 Example of flexibility of Bezier model: approximation of horn shape.

engine / vehicle combination, and some of them we use later as our objective functions. Four of these properties are derivatives of the engine performance only, namely *maximum power*, *integrated power*, *location emphasized integrated power* and *bmep* (efficiency). The fifth one, namely *coverage*, reflects compatibility of the engine properties with gearing / transmission from the perspective of vehicle performance. All of these are discussed in the following.

While speaking of engines, the easiest and probably most common way to compare engines is by finding out which one of them has the highest maximum power output, P_{max} . Unfortunately, this type of comparison is potentially very misleading. An engine with an extremely high power output may have a razor-sharp power band and may prove inferior in terms of usable power to its low power counterpart, which may have a nicely shaped and widespread power band. Power band is a somewhat vague term, but usually it means the location, width (both in rpm scale) and shape of the power curve in the interval where the engine produces its maximum power (peak power). For example, in Figure 24 we could say that the power band begins at 11000 or 11500 rpm and continues to give good power up to 13500 rpm.

One possible way to compare power curve with power distribution is to calculate integrated power P_{int} in some predefined interval. Integrated power is merely the area under the power curve in some predefined interval.

Integrated power does not take the location of the power peak into consideration. Sometimes the designer may want to set the power peak at a desired location, for example, to restrict mechanical stresses to engine. For this reason, we introduce *location emphasized integrated power*, and refer to it as P_{leip} . While calculating P_{leip} we must give the desired location in rpm scale for the power peak, and define the rpm interval before and after peak power. If the actual power peak is not co-located with the desired peak, the true area under the curve between a given interval is reduced by a factor of differences between real peak power and power at a desired peak. Let us consider an example where the desired peak is

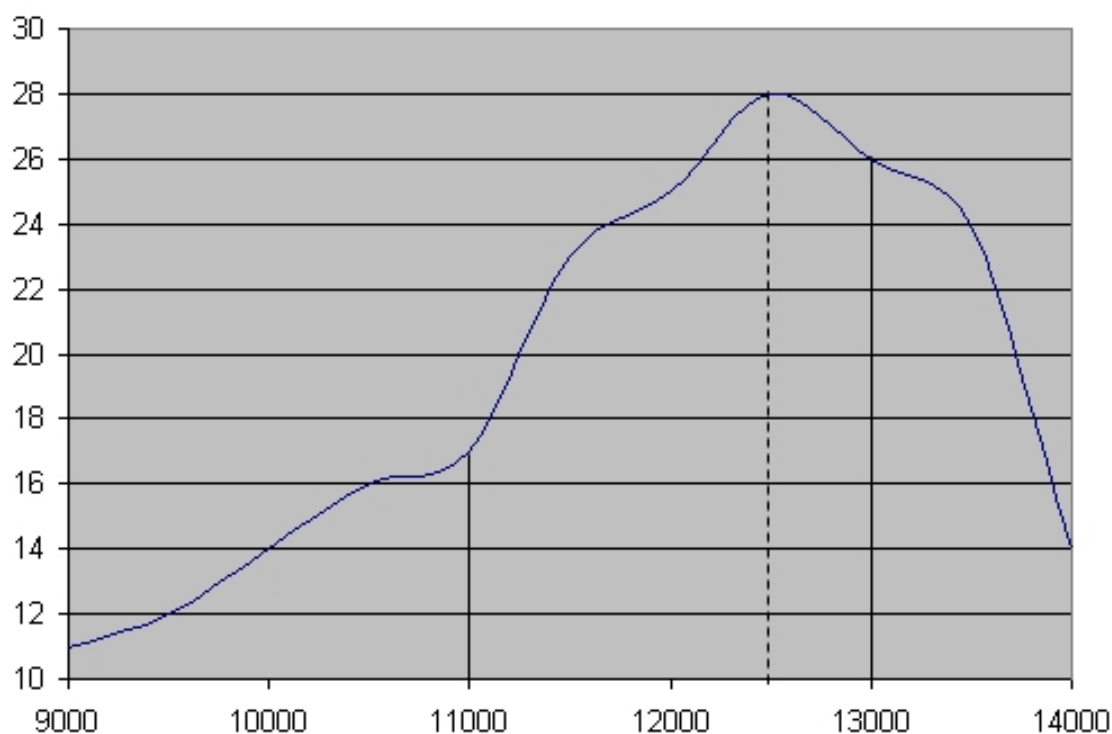


FIGURE 24 Three possible properties for optimization: maximum power, integrated power and location emphasized integrated power.

at 12000 rpm, the rpm interval is 2000 rpm before the peak and 1000 rpm after the peak (the resulting band being 3000 rpm between 10000 to 13000 rpm). Now, if the integrated area between 10000-13000 is 400, and the real peak is located at 13000 rpm with a peak power of 26 hp, and power at the desired peak (12000 rpm) is 24 hp, the true integrated area will be reduced by a factor of $24/26 = 0.923$, and the resulting function value P_{leip} is $0.923 \cdot 400 = 369$. Thus better function values are produced when the real power peak matches with the desired power peak.

In Figure 24 an imaginary power curve can be seen together with examples of the maximum power (a vertical dashed line at 12500 rpm), the integrated power in a given interval (the area between vertical solid lines at 11000 and 13000 and under the power curve resulting band of 2000 rpm) and location emphasized integrated power (the desired peak at 12500, the interval before the peak is 1500 and the interval after the peak is 500).

In addition to the maximum power, to the integrated power and to the location emphasized integrated power, there is also a fourth factor which can be used to restrict the location of the peak power. In general, there are only two ways to improve the maximum power of an engine: by increasing the operating speed of the engine or by increasing the pressure inside the cylinder during the power stroke (volumetric efficiency). The latter is to be preferred, since increasing the operating speed of the engine results in increased mechanical stresses, decreased mechanical efficiency due to increased friction, and exposes the engine to a failure.

A factor to describe the pressure inside the cylinder during the power stroke is known as *b MEP* (brake mean effective pressure), and it can be understood as a measure of effectiveness of the engine: the higher the *b MEP* at a given rpm, the higher the efficiency. Here, *B MEP* is calculated as

$$b MEP(P, rs, cc) = \frac{P \cdot 10000}{rs \cdot cc}, \quad (19)$$

where

P = power of the engine at the speed rpm, kW

rs = speed of the engine at the maximum power, revolutions/second

cc = engine displacement (constant), cubic centimeters

As seen in the formula, *b MEP* can be calculated at whatever running speed of the engine, but we are only interested in the special case of the maximum power and the corresponding running speed of the engine. In the given formula for the *b MEP* we can see that if there are two identical engines in terms of displacement and maximum power, the one which produces maximum power at a lower running speed has a higher *b MEP*. The purpose of using *b MEP* is to find a preferred engine design which gives the maximum power at some reasonable running speed.

In this study we do not only examine the goodness of the engine alone (i.e., with the four properties above), the goodness is also seen from a somewhat wider perspective in which the engine is used to motorize some vehicle. In this case, the goodness of the engine design is affected, in addition to the characteristics of power curve, also by the compatibility of the power curve and the type and implementation of a power transmission and gearbox design (gear ratios). A vehicle must be able to operate at very different speeds, for example, a regular car must cover all speeds from a slow walking speed to fast freeway speeds up to 200 km/h. No engine is capable to such flexibility, and thus some sort of gearing is needed for adjusting the ratio between the rotational speeds of the engine and the tyres of a vehicle. In general, the gearbox can contain a set of paired pinions, where two pinions form one gear ratio. Usually the vehicle has four to six gears.

Another type of gearing is a continuously variable transmission (CVT), which is implemented using a belt drive and pulleys which are capable of changing their diameter [28] and thus the gear ratio. In principle the CVT can fully use the potential of the engine, but because of mechanical losses and durability issues the normal pinion gearbox is generally used. The pinion gearbox cannot use the full potential of the engine, but ratios of the different gears with regard to the engine's behaviour and power band define how well the engine-gearbox combination suits to a particular engine.

The goodness of the engine-gearbox design may be regarded as a strict science only in some limited cases, because different users normally have different preferences; their driving styles and skills may be totally different, and even the purpose of the vehicle may be very different. In general, application of the engine must be taken into account. Below there are some factors as examples to be

considered:

- wide gearbox ratios require a wide power band.
- for everyday use the engine must be "elastic" and "smooth" (flat power curve without sudden humps and dips) at the expense of the maximum power. This reduces the need to change the gears all the time and makes driving flexible.
- for racing purposes usability is sacrificed in quest of the maximum power, because a close ratio gearbox (one having at least six gears, which are by ratio close to each other) allows utilization of the razor sharp power band.

As a general rule of thumb, the power band of the engine should be wide enough to allow switching from gear to gear without dropping out of the power band. It is possible to examine the tendency to drop out of the power band by plotting a cascade of torque curves acquired using all gears against a so-called ultimate driving force curve (UDFC), see Figure 25. The UDFC is the imagined optimal case where the transmission is continuously variable, and thus the vehicle can utilize maximum power of the engine at all road speeds, which produces for example fastest acceleration of the vehicle. Ultimate driving force F_{udf} in Newtons is calculated using the formula

$$F_{udf}(V) = \frac{3600 \cdot P}{V}, \quad (20)$$

where

P = maximum power of the engine, kW

V = road speed, km/h.

The torque cascades for all gears are calculated similarly to the UDFC, but this time maximum power of the engine is not used. Instead, we use for each road speed power that the engine produces at the chosen gear at that particular speed. The real driving force F_{rdf} is calculated for each gear using the formula

$$F_{rdf}(g, V) = \frac{3600 \cdot P_{vg}}{V}, \quad (21)$$

where

P_{vg} = power of engine at gear g running at speed v , kW

V = road speed, km/h.

To see how well a particular engine suits to a particular gearing, we calculate the value of coverage, which is derived from the UDFC and the set of torque cascades. The coverage c is the ratio between the areas of the torque cascades and the UDFC, and is calculated as

$$c = \frac{A_{tc}}{A_{udfc}}, \quad (22)$$

where

A_{tc} = area under the torque cascade lines

A_{udfc} = area under the ultimate driving force line.

If full potential (power) of the engine were usable at all road speeds, the coverage would be one, and this is the optimum case. In Figure 25, the ultimate driving force is plotted as a dashed line against road speed of the vehicle. Torque cascades are plotted as solid lines. The coverage is the ratio of the areas under the ultimate force line and the torque cascades.

In real life, consideration of the coverage only is not enough. It is possible to have an engine-gearbox combination with a very high coverage, while the peak power of the engine is low. This is obviously not a good solution. Therefore, the coverage and the maximum power are conflicting objectives and they must be considered in unison. The overall performance of an engine/vehicle combination can be determined and controlled by three factors: coverage, maximum power and bmep. When considering the goodness of a particular engine-gearbox combination all three of them must be taken into account. It is possible to consider all of them simultaneously as three objective function values (coverage, power and bmep) and formulate a multiobjective optimization problem.

For example, one engine may have a high maximum power and a poor coverage, and another one may have a low maximum power and a good coverage. There is no mathematically justified means to decide which one of these designs is actually better, and thus the decision must be left to the engine designer. The situation described above is common in the field of multiobjective optimization, and it is discussed in Section 2.3.

4.4.1 On the implementation

Our system for simulation based optimization consists of several heterogeneous software modules. The most prominent of these are the engine simulator and several different optimization algorithms. All the software runs on Windows XP environment and the optimization algorithms are either stand-alone applications or Matlab scripts.

MOTA simulator is a stand-alone application which consists of a graphical user interface and a separate executable, which forms the calculational core of the simulator. As input, the simulator reads the engine configuration file, which contains full information about geometrical properties of the engine and external ducting and operating conditions of the engine. The input file also contains simulation run specific data, such as the number of rpm steps, initial rpm, and rpm step size. After a simulation run, the simulator produces an output file, which contains lots of data, but the most important for us are the power and torque values for all given rpm steps. The values for objective functions are constructed from power, torque and rpm values.

To put all the pieces together we wrote some additional Java code, which conceals the complex configuration file structure of the simulator. The simulator

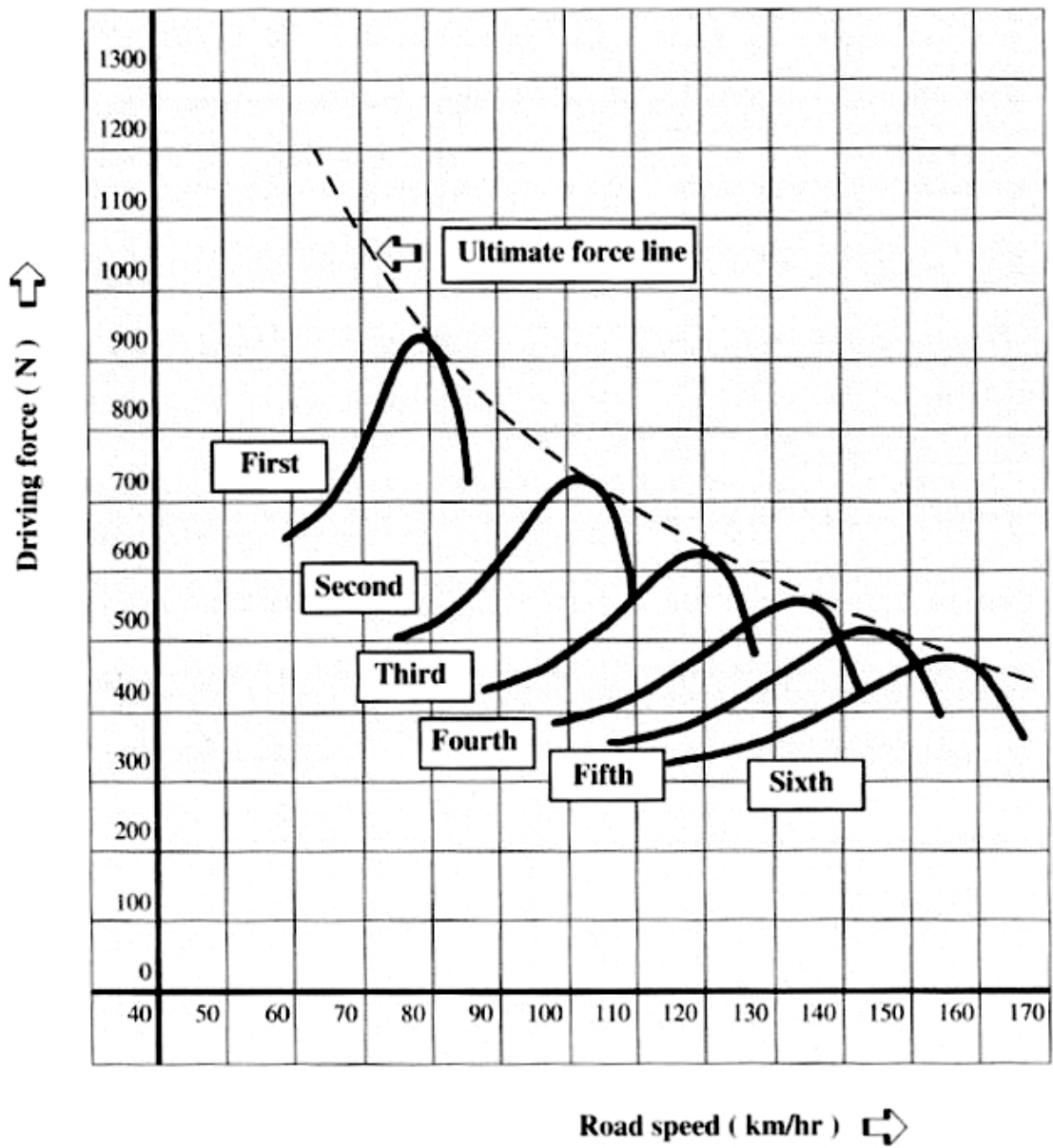


FIGURE 25 Ultimate driving force curve (UDFC) and torque cascades. Figure from [13].

can be called via a simple Java function call. The Java code takes care of converting design variables into simulator configuration files (measures of external ducting are inserted in the configuration file, which is otherwise left intact) and executing the calculation core of the simulator with a given engine configuration. The code also calculates the objective function values from the simulator output files.

Control flow starts from the optimization algorithm. In the case of Matlab scripts, algorithms can call Java code directly via the Matlab's own Java interface. While using stand-alone optimization algorithms, direct call to Java code is not always possible. In this case, calling Java code is done in a somewhat more complex way. Design variable values are written into an external ASCII file, after which Java execution is invoked via a command file call in the optimization algorithm. An objective function value is once again written to the external ASCII file by Java code after the completion of the simulator run. Finally the objective function value is read from the file by the optimization algorithm.

Use of Java implemented interfacing code in our optimization system virtually enables use of any optimization algorithm (assuming that the source file is provided) and simulator combination.

4.5 Formulation of an optimization problem

4.5.1 Single objective case

Although performance studies of a particular engine should be done related to the planned use of the engine and also to the gearing of the vehicle, these are often omitted, and performance study is based merely on power and torque curves. For this reason, performance consideration can be done simply by evaluating the properties of the power curve of that particular engine and ignoring the effect of the gearing and the gear ratios as described in Section 4.4.

Because the use of maximum power has only limited value in estimating engine performance, and we want some control over peak power location, we prefer to maximize location emphasized integrated power (like depicted in Figure 24) in this study. In this case the single objective optimization problem is of the form

$$\begin{aligned} & \text{maximize } F(x), \\ & \text{subject to } x \in S, \end{aligned} \tag{23}$$

where

$F(x_1, \dots, x_n) = P_{leip}$, location emphasized integrated power (area under curve).

The feasible design variable space S is limited by the upper and lower bounds for each variable. The variable can have any value between its bounds, and thus

the problem has box constraints. The design variables were discussed more in depth in Section 4.3. One specific design variable common to all models is tuned length, and in our study it is bounded from 700 to 800 mm. Note that the numbers vary depending on which model is used to depict the exhaust pipe shape.

4.5.2 Multiobjective case

If gearing is taken into account while judging the goodness of some particular engine / vehicle combination, the problem at hand is a multiobjective optimization problem of the form

$$\begin{aligned} & \text{maximize } \{F_1(x), F_2(x), F_3(x)\} \\ & \text{subject to } x \in S, \end{aligned} \tag{24}$$

where

$$\begin{aligned} F_1(x_1, \dots, x_n) &= P_{max}, \text{ maximum power of the engine} \\ F_2(x_1, \dots, x_n) &= c, \text{ coverage} \\ F_3(x_1, \dots, x_n) &= bmep, \text{ effectiveness of the engine.} \end{aligned}$$

The feasible design variable space S is limited similarly to a single objective case by upper and lower bounds for each variable.

In this chapter we have discussed the layout of the optimization system as a whole, and also some details of its modules, exhaust pipe models and different ways to calculate objective function values. In the next chapter we outline how we compared different exhaust pipe models and different optimization algorithms both in a single and in a multiobjective case. We also report results obtained by different scalarization methods.

5 RESULTS

We have tested our simulation based optimization system using as a target engine an air-cooled high rpm Yamaha YZ-80 motocross engine with 6 gears. This particular engine was chosen because it is a representative of high output, high rpm 2-stroke engine and the configuration for it was given as a sample file in the MOTA distribution package. For calculation of coverage, the following gear ratios were used: 1st = 2.545, 2nd = 1.933, 3rd = 1.571, 4th = 1.333, 5th = 1.166, 6th = 1.045.

The simulator was set to calculate engine output values in a rpm range from 9000 to 14000 rpm, with a step of 250 rpm, resulting in 21 different engine speeds. This data was used in production of objective function values.

Our test procedure was divided into a two somewhat distinct phases, namely a single and a multiobjective phase. In the single objective phase our intention was to employ a simple problem formulation, and compare the performance of different global optimization algorithms and different ways to model exhaust pipe shape, and select the best of those to be used in the more complex multiobjective phase. In the multiobjective phase our aim was to compare different scalarization methods from three separate scalarization method classes.

In the single objective phase different exhaust pipe shape models were compared using both the CRS2 algorithm and the Niederreiter random sampling to decide which model has most potential. By this comparison only one model, the Bezier model, was selected for further study to avoid the need to examine all possible shape model / optimization algorithm combinations. Finally, several global optimization algorithms, CRS2, CRS4, SAHPS, SCGA, DE and SuperEGO, were tested upon the Bezier model to find out which optimization algorithm had the best potential in this particular problem.

In the multiobjective phase the exhaust pipe shape model and the optimization algorithm selected in the single objective phase were used. To convert the multiobjective optimization problem into a single objective one, five different scalarization functions and the interactive NIMBUS method were employed, and their features were examined. One of the scalarization functions employed no preference information (neutral compromise solution), another employed prefer-

ence information given by weights (weighted and scaled Chebychev function), and the rest three relied on reference points. Effects of specifying different reference points on solutions obtained are shown in four sample cases with three different scalarization functions. Finally we tested the interactive NIMBUS method, and we were able to reduce the total number of function evaluations with the help of intermediate solutions.

5.1 Single objective case

In the single objective case our aim was to compare different models to represent exhaust pipe shape and also different optimization algorithms, and then choose one model and one optimization algorithm to be used with multiobjective optimization.

Our single objective optimization problem was formulated as given in Equation 23. For our objective function we selected location emphasized integrated power P_{leip} , because it has most practical potential of the objective functions discussed in Section 4.4, and by using it we could have good control over power peak location. P_{leip} was calculated with the desired peak location set at 12500 rpm, the interval before the peak at 2000 rpm and the interval after the peak at 500 rpm.

5.1.1 Comparison of different shape models

Different models were initially compared using the Niederreiter random sampling to get a coarse impression of the capabilities of each exhaust pipe model, and also of the search space structure and the distribution of good solutions within search space. We assumed that more potent and flexible models should produce better solutions than their more rigid counterparts if search space is sampled with equal density of covering. A maximum of 2695 objective function evaluations were executed for each of the Bezier, horn, Blair and FFS models, and the best objective function values found before 500, 1000, 1500, 2000 and 2500 evaluations were recorded. Special attention was paid also to the best objective function value reached before $245n$ iterations to compare the behavior in relation to design variable number n . This data is presented in Table 6, where in each cell the best objective function value before a given number of evaluations is presented together with the evaluation number, which produced that value, in parenthesis.

With fewer than 500 and 1000 function evaluations, the Blair model produced the best objective function values, probably due its low number of design variables. With more function evaluations, the Bezier model dominated, and it also produced the best objective function value with respect to $245n$ iterations. FFS was far behind the other models, probably because of its higher dimensionality. When comparing with the Niederreiter random sampling, the Bezier and Blair models seemed to be more effective. Nevertheless, this kind of comparison

TABLE 6 Results using Niederreiter random sampling. The best function value gained before a given number of evaluations in each cell is on top, and respective evaluation number is below in parenthesis.

Model	n	Bef. 500	Bef. 1000	Bef. 1500	Bef. 2000	Bef. 2500	Bef. 245n	245n
Bezier	6	388.941 (193)	389.812 (534)	400.786 (1222)	400.786 (1222)	400.786 (1222)	400.786 (1222)	1470
Horn	6	393.012 (450)	393.012 (450)	393.012 (450)	393.643 (1525)	393.643 (1525)	393.012 (450)	1470
Blair	5	396.909 (123)	399.925 (731)	399.925 (731)	399.925 (731)	399.925 (731)	399.925 (731)	1225
FFS	11	382.755 (263)	385.500 (672)	385.500 (672)	385.500 (672)	385.758 (2099)	385.758 (2099)	2695

only gives a very rough estimate of the behavior of the model, and thus we further studied the models using the CRS2 algorithm. CRS2 was selected for our initial comparison because it is a widely cited and well-known algorithm, and some studies, such as [1] and [59], also indicate that it is efficient. Population size NP of CRS2 was set for each model to $5n$ and a maximum of 2695 objective function evaluations were executed. Data similar to the Niederreiter random sampling was recorded. The results of the CRS2 run are presented in Table 7.

TABLE 7 Results using CRS2 and population size $5n$. The best function value gained before a given number of evaluations in each cell is on top, and respective evaluation number is below in parenthesis.

Model	NP	Bef. 500	Bef. 1000	Bef. 1500	Bef. 2000	Bef. 2500	Bef. 245n	245n
Bezier	30	404.761 (463)	406.160 (997)	406.515 (1288)	406.523 (1994)	406.623 (2005)	406.515 (1288)	1470
Horn	30	403.896 (455)	405.642 (974)	405.888 (1451)	406.083 (1607)	406.083 (1607)	405.888 (1451)	1470
Blair	25	399.026 (493)	399.226 (968)	399.331 (1385)	399.375 (1949)	399.508 (2145)	399.266 (1057)	1225
FFS	55	337.903 (15)	385.427 (992)	396.514 (1494)	401.545 (1957)	403.378 (2409)	404.180 (2615)	2695

As can be seen in Table 7, the Bezier model dominated comprehensively, and the horn model surpassed the Blair model, as was expected due to the better flexibility of the horn model. As the number of function evaluation counts increased, the FFS model produced better and better solutions, but FFS obviously suffered of high dimensionality, and for this reason the absolute number of function evaluations grew to an intolerably high level.

From the results of Tables 6 and 7 we can speculate that the greater the flexibility of the model with respect to the number of design variables, the better

the performance of that model. The Bezier model with its high flexibility and its modest number of six design variables performed better than its more rigid counterparts, the horn and Blair models. The Blair model reached an objective function value level of approximately 399 at very early evaluations, and could not improve it essentially, probably due to the rigidity of the model. The FFS model is the most flexible of all four models, but at the same time it suffers of high dimensionality. We can assume that if we had used function evaluations profligately, FFS would eventually have performed very well.

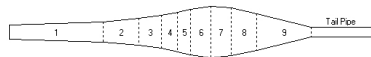
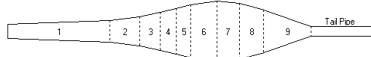
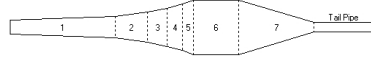
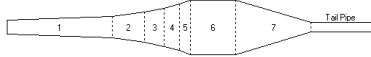


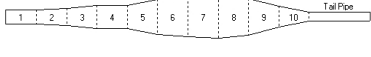
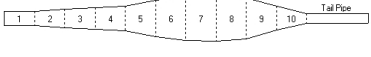
Method	Over 400	Best
Bezier	401.001 (154) 	406.623 (2005) 
Horn	400.594 (259) 	406.083 (1607) 
Blair	399.508 (2145) 	399.508 (2145) 
FFS	400.00 (1807) 	404.180 (2615) 

FIGURE 26 Examples of actual pipe shapes using all four models in a CRS2 run. On top of each shell, the objective function value and the respective evaluation number in parentheses.

In Figure 26, we present two pipe shapes for each model gained during a CRS2 run. The first shape is from a situation where the objective function value passed a level of 400 for the first time, and the second shape corresponds to the best objective function value of the whole optimization run. Above the pipe shape, the objective function value and the respective evaluation number are given in parentheses. Images in Figure 26 depict the pipe shape represented to the simulator, and thus differences in shape models (sections and their dimensions) can be compared. The Bezier and Horn models produce a very similar looking diffuser and baffle cone sections, although the general outline of the pipe with the horn model is more chiselled. Also the resulting objective function values with both of the shapes are very close to each other. The Blair model produces a somewhat thinner exhaust pipe, and the diffuser is a far coarser representation of the horn shape as is the case with the Bezier and horn models, due to the rigidity of the Blair model. Probably for these reasons performance remains lower than with the Bezier and horn models. The FFS model produced visible irregularity to the diffuser section, which is probably detrimental to performance, and it is slighter after more function evaluations. Probably with even more function evaluations the FFS model would have reached the performance level of the Bezier and horn models.

5.1.2 Comparison of different optimization methods

For good results, flexibility and a modest number of design variables we selected the Bezier model for comparisons between different global optimization methods, namely CRS2, CRS4, SAHPS, SCGA, DE and SuperEGO. For these comparisons we tightened our demands, and only allowed a maximum of 1000 objective function evaluations for each algorithm. This number of function evaluations corresponds roughly to amount of calculation that can be executed outside of office hours during night-time. A thousand evaluations was considered a sufficient number to adequately demonstrate the capabilities of each algorithm, while allowing the total calculation time for all of the algorithm comparisons to be kept at a reasonable level. Nevertheless, it took a few weeks calculational time to produce the data displayed in Table 8.

With CRS2, CRS4 and DE, the effect of the size of the population was also examined. With DE, the effect of crossover parameter CR and mutation parameter F was also tested with the best population size. For each test case, the best function values gained are reported before 100, 200, 300, 400, 500 and 1000 function evaluations. The results of the optimization algorithm comparison are presented in Table 8. In the first column a method name is given with a population size with regard to the problem size n (where applicable). The second column indicates population size NP , and the rest of the columns are the best function values gained before a defined number of evaluations. With a function value, the corresponding evaluation number is given in parenthesis.

Overall, CRS variants seemed to work most efficiently, especially CRS2. Surprisingly, a moderate population size of $5n$ produced the best performance. At the beginning, SuperEGO produced the second best function values after CRS2, but as the number of function evaluations grew, the computational overhead of the SuperEGO algorithm itself became prohibitive. While other algorithms had negligible overheads and could be executed at a constant pace of approximately 1200 iterations per day (24 h), SuperEGO managed to go through a mere 300 iterations per day, with ever decreasing rate, meaning that SuperEGO was the slowest algorithm of all, regardless of the number of function evaluations.

DE performed best with even smaller population ($3n$) than CRS2, a majority of the DE runs used crossover parameter $CR = 0.9$ and mutation parameter $F = 0.8$ as suggested in [60], except in the case marked with an asterisk in Table 8, where $CR = 0.8$ and $F = 0.5$, and in the case marked with a double asterisk where $CR = 0.7$ and $F = 0.2$. The effect of the mutation parameter is clearly visible in results, as expected. With the smaller values of F , search is expected to converge faster with the expense of higher risk to get stuck on some local optimum. With $F = 0.5$ search was somewhat faster than with $F = 0.8$, but reducing F further down to 0.3 obviously lead to a premature convergence to some local optimum. The majority of the DE runs were executed using the DE/best/2/bin strategy, three runs referred to as DE2 were executed using the DE/rand/1/bin strategy, with $CR = 0.9$ and $F = 0.8$. DE2 did not perform nearly as well as DE, thus no further parameter tweaking was studied.

TABLE 8 Comparison of different global optimization algorithms. Best function values gained before given number of evaluations in each cell are on top, and respective evaluation number is below in parenthesis.

Method	NP	Bef. 100	Bef. 200	Bef. 300	Bef. 400	Bef. 500	Bef. 1000
CRS2 3n	18	382.124 (93)	396.699 (173)	400.086 (218)	401.280 (399)	401.713 (490)	402.619 (918)
CRS2 5n	30	387.714 (61)	401.001 (154)	401.951 (295)	403.684 (382)	404.761 (496)	406.160 (997)
CRS2 6n	42	381.333 (89)	388.295 (123)	397.683 (280)	401.083 (353)	403.143 (497)	405.735 (984)
CRS2 10n	50	364.191 (76)	395.376 (144)	398.086 (242)	403.259 (371)	404.655 (491)	405.189 (861)
CRS4 3n	18	347.396 (98)	354.826 (174)	355.334 (298)	356.156 (399)	356.653 (450)	357.088 (956)
CRS4 5n	30	371.240 (84)	388.614 (191)	401.206 (288)	402.136 (499)	402.476 (495)	404.489 (997)
CRS4 10n	60	344.700 (89)	387.489 (164)	387.489 (164)	390.318 (367)	394.401 (494)	402.966 (997)
SAHPS	-	156.531 (21)	165.465 (196)	232.328 (221)	232.3278 (221)	232.328 (221)	279.829 (671)
SCGA	-	154.392 (16)	164.202 (196)	175.129 (295)	222.200 (329)	222.200 (329)	228.124 (882)
DE 3n	18	389.534 (60)	395.559 (113)	398.586 (221)	398.700 (349)	402.301 (421)	405.170 (924)
DE 3n *	18	397.113 (68)	399.166 (166)	400.931 (251)	402.590 (347)	404.135 (490)	405.645 (918)
DE 3n **	18	388.483 (95)	393.528 (193)	394.026 (279)	394.250 (379)	394.391 (488)	394.486 (794)
DE 5n	30	382.180 (99)	389.109 (183)	398.273 (249)	398.273 (249)	398.273 (249)	403.866 (980)
DE 10n	60	369.928 (25)	374.911 (164)	374.911 (164)	374.911 (164)	391.96 (494)	401.856 (821)
DE2 3n	18	379.015 (97)	394.292 (137)	398.83 (261)	398.83 (261)	399.7 (438)	403.483 (967)
DE2 5n	30	387.798 (87)	396.819 (128)	396.819 (128)	399.284 (367)	401.416 (486)	404.08 (819)
DE2 10n	60	344.317 (72)	375.905 (116)	378.876 (214)	389.728 (394)	395.896 (488)	398.331 (676)
Super EGO	-	379.820 (90)	398.555 (168)	401.556 (289)	401.566 (329)	-	-

With DE and SAHPS implementations used there was a problem with design variable bounds, because both algorithms tried to proceed searching out of the feasible region, i.e. intervals for design variables. In some cases this is not acceptable, because, for example, it may result non physical solutions, such as

sections having negative length etc. For this reason, the objective function was modified to give worse values as bounds were broken and summing the absolute values of those design variables that were violated. Thus search procedure was not encouraged to proceed out of the feasible region. SAHPS and SCGA both performed very poorly, probably because of their complicated structure, and both of them produced quite poor objective function values with regard to other methods.

TABLE 9 Ordinal comparison of different global optimization algorithms. Algorithms for each function evaluation level are ordered columnwise.

Bef. 100	Bef. 200	Bef. 300	Bef. 400	Bef. 500	Bef. 1000
DE 3n *	CRS2 5n	CRS2 5n	CRS2 5n	CRS2 5n	CRS2 5n
DE 3n	DE 3n *	SuperEGO	CRS2 10n	CRS2 10n	CRS2 6n
DE 3n **	SuperEGO	CRS4 5n	DE 3n *	DE 3n *	DE 3n *
DE2 5n	DE2 5n	DE 3n *	CRS4 5n	CRS2 6n	CRS2 10n
CRS2 5n	CRS2 3n	CRS2 3n	SuperEGO	CRS4 5n	DE 3n
DE 5n	DE 3n	DE2 3n	CRS2 3n	DE 3n	CRS4 5n
CRS2 3n	CRS2 10n	DE 3n	CRS2 6n	CRS2 3n	DE2 5n
CRS2 6n	DE2 3n	DE 5n	DE2 5n	DE2 5n	DE 5n
SuperEGO	DE 3n **	CRS2 10n	DE2 3n	DE2 3n	DE2 3n
DE2 3n	DE 5n	CRS2 6n	DE 3n	DE 5n	CRS4 10n
CRS4 5n	CRS4 5n	DE2 5n	DE 5n	DE2 10n	CRS2 3n
DE 10n	CRS2 6n	DE 3n **	DE 3n **	CRS4 10n	DE 10n
CRS2 10n	CRS4 10n	CRS4 10n	CRS4 10n	DE 3n **	DE2 10n
CRS4 3n	DE2 10n	DE2 10n	DE2 10n	DE 10n	DE 3n **
CRS4 10n	DE 10n	DE 10n	DE 10n	CRS4 3n	CRS4 3n
DE2 10n	CRS4 3n	CRS4 3n	CRS4 3n	SAHPS	SAHPS
SAHPS	SAHPS	SAHPS	SAHPS	SCGA	SCGA
SCGA	SCGA	SCGA	SCGA	SuperEGO	SuperEGO

In Table 9 ordinal comparison of algorithms is presented. In each column methods are in the order of superiority by the best function value achieved before 100, 200, 300, 400, 500 and 1000 function evaluations, the best one at the top. From this data we can extend implications made from the data in Table 8, and see that CRS2 was overall working most efficiently with the current problem setting. DE produced almost as good results as CRS2, but as there are many critical parameters to set (population size, values for crossover and mutation parameters and selection of DE-strategy) which affect the performance of the algorithm, DE is more complicated to use and leaves certain amount of uncertainty for the designer to be dealt with. In fact, selection of parameters for DE is an optimization problem itself. For these reasons we chose to exclusively use the CRS2 method, with a population size of $5n$, as the optimization algorithm in the multiobjective case, and depict pipe shape using the Bezier model.

5.2 Multiobjective case

In the multiobjective case we had three conflicting objectives, maximum power, coverage, and bmep, as defined in Section 4.4. Simulator setup was similar to that in the single objective case, and exhaust pipe shape was modelled using the Bezier model. Different scalarization methods were used to convert the problem into a single objective one, and the resulting single objective problem was solved using the CRS2 algorithm with a population size of $5n$.

5.2.1 Comparison of different scalarization functions

First we made a rough comparison of methods between different classes (no preference information, weight based preference information and reference point based preference information) allowing a maximum of 200 objective function evaluations. Some of the best methods were then further tested with an ever-decreasing number of objective function calls to evaluate how this affects the quality of the solution.

For normalizing purposes, our scalarization functions require information about ideal and nadir objective function values. In our study, ideal and nadir points were not calculated explicitly, because it would have required lots of computing time. Instead, the value 0 was used for all the components of the nadir objective vector. For the components of the ideal objective vector, the maximum power was set to 34 hp, the coverage was set to 1, and bmep to 15. All these three values are sufficiently good and cannot be reached and can, thus, serve as components of the ideal objective vector.

In Table 10 we show the results of the neutral compromise solution (NCS) (no preference method) and weighted and scaled Chebychev (WSC) scalarization (weight based preference information) functions with four different sets of weights. The neutral compromise solution shows quite acceptable objective function values, considering the fact that there was no preference information available. If the designer lacks expertise of a specific domain, (s)he can get some usable solution using the NCS function.

TABLE 10 Multiobjective optimization results and weights using neutral compromise solution (NCS) and weighted and scaled Chebychev (WSC) scalarization functions.

	NCS	WSC Case 1		WSC Case 2		WSC Case 3		WSC Case 4	
	Values	Wgts	Vals	Wgts	Vals	Wgts	Vals	Wgts	Vals
Power	25.32	0.33	18.69	0.25	18.05	0.18	17.83	0.13	25.19
Coverage	0.85	0.33	0.96	0.50	0.97	0.65	0.97	0.80	0.55
Bmep	11.17	0.33	8.96	0.25	8.66	0.18	8.37	0.07	12.08

With the WSC function we first defined identical weights for all the objectives in Case 1. This resulted in a very high coverage, and in quite poor power

and bmep values. In Case 2 we wanted to gain more power and bmep and less coverage, thus the weights were adjusted accordingly. Surprisingly, this caused no changes in the objective function values. In Case 3, the weights were further adjusted to gain more power and bmep at the expense of coverage, but we still failed to see any significant changes in the objective function values. In Case 4, we further sacrificed coverage, and emphasized bmep over power. This resulted in a dramatic increase in both power and bmep, and in an equally dramatic decrease in coverage. This erratic behavior probably implies that the task at hand is somewhat demanding, and that a weight based approach is not appropriate for solving it.

Data in Table 10 clearly demonstrates the weakness of the weight based scalarization function: there is no direct relationship between the weight values and the resulting objective function values. In our case, weight adjustment in the middle range did not produce significant changes, if any, but changes near the extreme values had a remarkable effect. Thus, selecting values for weights is a kind of guesswork, and the results can be seen only after a time consuming completion of the optimization process when the designer is able to assess whether the selected weights led to the desired output or not.

Problems of weight based scalarization can be overcome using reference point based scalarization. In reference point based scalarization, preference information is given as a vector consisting of desirable objective function values, i.e. reference point.

We show the effects of specifying different preferences (in the form of different reference points) on solutions obtained in four sample cases with three different scalarizations discussed in Subsection 2.3.1, namely achievement scalarizing function (ACH) Equation 3, function from satisficing trade-off method (STOM) Equation 4 and the one used in GUESS method Equation 5. Additionally, for comparison, we show results using the standard NIMBUS method subproblem formulation (STD) given in [45] and implemented in IND-NIMBUS software. In first two cases we were essentially optimizing only a single objective at a time, either bmep (Case 1) or maximum power (Case 2), to see how this affects the other two objectives, and also to show the weakness of the single objective approach.

In Case 3, we aimed at maximum coverage and wanted to keep power and bmep at a reasonable level at the same time. In Case 4, we slightly relaxed our demand for maximum coverage, and aimed at a somewhat higher power output while retaining the bmep requirement at the same level as in Case 3. The exact reference point values (Ref) and obtained Pareto optimal objective function values produced (Pwr, Cov and Bmp) with different scalarizations (after a modest amount of 200 simulator calls for each case) are shown in Table 11.

From the results given in Table 11 we can deduce that power and bmep are not such conflicting objectives as we first thought. Case 1 aimed at maximum bmep, and Case 2 for maximum power, and yet power and bmep values between those cases do not differ significantly. However, coverage is a different matter. In Cases 1 and 2, coverage is rather poor, and despite of high power output and

TABLE 11 Reference points and solutions with different scalarization methods.

	Case 1			Case 2			Case 3			Case 4		
	Pwr	Cov	Bmp	Pwr	Cov	Bmp	Pwr	Cov	Bmp	Pwr	Cov	Bmp
Ref	0	0	15	34	0	0	22	1	10	25	0.88	10
STD	25.8	0.54	12.1	26.9	0.49	10.8	22.1	0.93	10.4	25.7	0.88	10.3
ACH	25.6	0.75	12.0	27.0	0.49	11.3	20.1	0.95	9.45	25.3	0.89	10.5
STOM	25.8	0.55	12.1	26.6	0.59	10.7	17.8	0.97	8.54	25.6	0.89	10.3
GUESS	27.1	0.43	11.3	27.2	0.49	11.8	20.8	0.95	10.2	25.2	0.88	10.7

high bmep, a bike equipped with this engine would probably perform poorly on the racetrack. In Case 3, we wanted to maximize coverage and keep power and bmep at a reasonable level at the same time. As we can see, there is a moderate decrease in maximum power, a slight decrease in bmep, but a huge increase in coverage. This setup would probably perform quite well also on the racetrack. In Case 4, we wanted to have more maximum power, and thus we relaxed the coverage requirement and required more power. When using a more realistic reference point, all three desirable objective function levels were reached as a result. As can be seen in Table 11, our optimization system obtained very satisfying results with regard to the reference point used. It is also important to notice how easy and straightforward it is to compare preference information with the resulting objective function values. The data also indicates that different scalarization functions produce slightly different Pareto optimal solutions, yet they all can be used to guide solution to the desired direction.

In Figure 27, some exhaust pipe designs with the resulting power curves are presented as examples. For each case, the results of a scalarization function that has produced in some sense the best values have been selected. In Case 1, the pipe shape looks credible, power output is sufficient and power peak is at a reasonable rpm level (slightly under 12 000 rpm). However, the power curve looks very "peaky". In Case 2, where the aim was to attain maximum power, the resulting pipe shape is not very appealing according to the traditions of the trade, and the power curve is fluctuating, which would make the bike very unpleasant to drive. In addition, the power peak is located quite high as expected, slightly under 14 000 rpm, which would compromise mechanical reliability of the engine. From these two cases we can deduce that the single objective approach with a simple objective function definition (here maximum power or bmep) is not adequate, and it may be even grossly misleading if the designer lacks expertise of the domain.

In Case 3, we wanted to obtain such values for maximum power, coverage and bmep that would result with a good performance in a real life situation. Peak power is here somewhat smaller than in the previous cases, but the shape of the power curve is very pleasant with an even delivery of power, resulting in much more integrated power (the area under the power curve between some given interval). In Case 4, we wanted to gain more power at the expense of coverage. The power curve looks again very pleasant, the peak power has increased and the power peak has shifted a bit higher on rpm scale, as expected. If the requirement

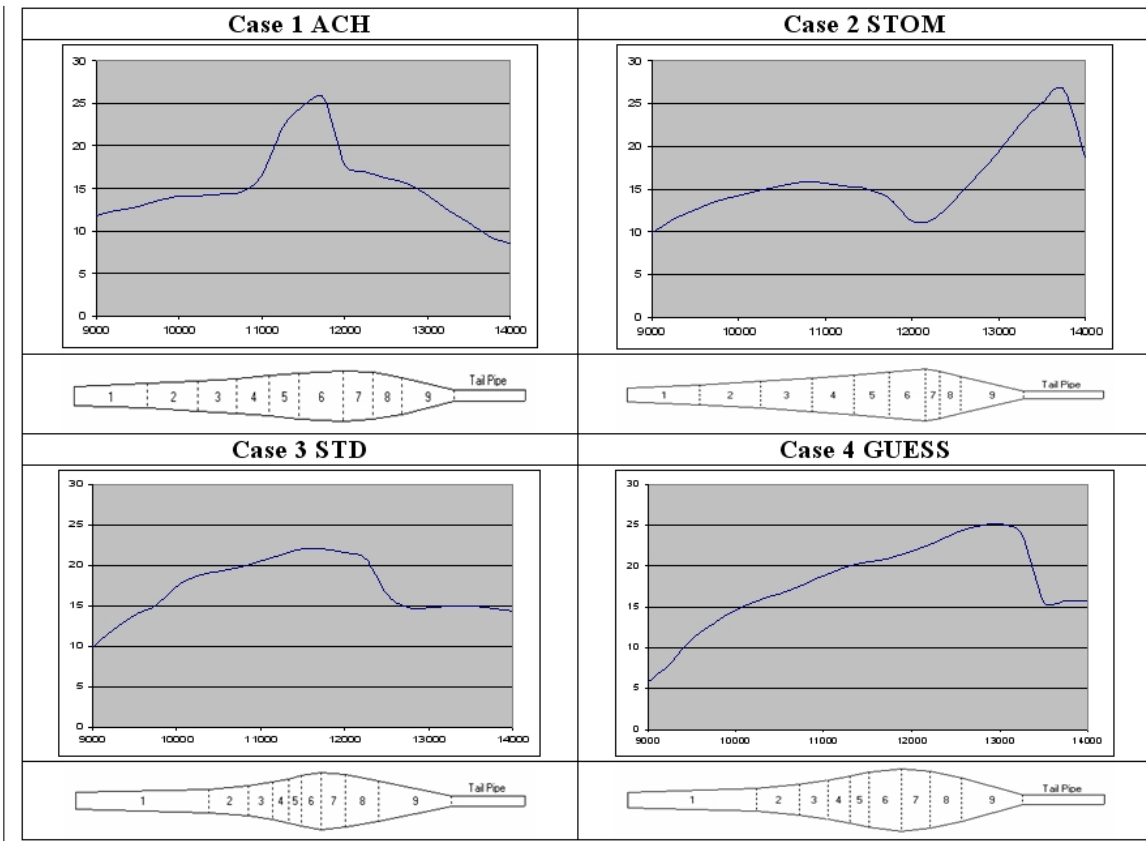


FIGURE 27 Examples of pipe shapes and corresponding power curves for all four cases.

for bmp had been higher, the power peak would have been lower in the rpm scale. In visual inspection, the exhaust pipe shapes in Cases 3 and 4 are very appealing and very similar to the ones used in real racing engines.

Overall, results with reference point based scalarization are very pleasing, and preference information can be given straightforwardly and without guesswork, in contrast to weight based scalarization. The results follow reference point information rather accurately with a modest number of only 200 evaluations. Defined this way, the reference point is reachable. Playing with preference information also allows the designer to learn about the problem itself by seeing what kinds of function values can or cannot be reached, and how they affect each other.

5.2.2 Number of function evaluations and solution quality

We wanted to study further effects of reduced number of evaluations to solution quality. This data is presented in Tables 12 and 13, where the best scalarization function values are presented within the intervals of 25 function evaluations along with separate objective function values. In Table 12 the reference point used (22.0, 1.0, 10.0) was the same as in Case 3 in Table 11, and in Table 13 the reference point used (25.0, 0.88, 10.0) was the same as in Case 4 in Table 11. Blank cells mean that there was no improvement to the previous evaluation count level.

TABLE 12 Effect of the number of function evaluations to solution. Reference point: power = 22, coverage = 1.0, bmep = 10.

Evals	ACH			STOM			GUESS		
	Pwr	Cov	Bmp	Pwr	Cov	Bmp	Pwr	Cov	Bmp
25	21.321	0.940	9.598	17.632	0.963	9.045	19.226	0.911	9.863
50	20.131	0.952	9.448				19.791	0.945	9.922
75							23.884	0.905	10.752
100							21.005	0.936	9.859
125									
150									
175				17.804	0.971	8.538	20.760	0.945	10.177

In Table 12 the reference point value for coverage was set to an unreachable theoretical maximum 1.0, and this is clearly visible in function values: coverage climbed quite high, whereas power and bmep lagged behind reference point values. In this sense, the reference point was not realistic nor reachable. A multitude of blank cells with ACH and STOM functions suggests that Pareto optimal results satisfying the preferences were hard to come by using these functions. GUESS could improve results more consistently, and its final solution would probably be slightly better than that of ACH and STOM functions.

TABLE 13 Effect of the number of function evaluations to solution. Reference point: power = 25, coverage = 0.88, bmep = 10.

Evals	ACH			STOM			GUESS		
	Pwr	Cov	Bmp	Pwr	Cov	Bmp	Pwr	Cov	Bmp
25	23.994	0.871	11.027	24.325	0.917	10.319	24.305	0.841	11.655
50	24.268	0.885	10.925				24.392	0.866	10.761
75	24.964	0.876	10.198						
100				24.381	0.883	10.976	25.104	0.878	10.649
125	24.927	0.878	10.782	25.121	0.878	10.656			
150				25.616	0.889	10.274			
175	25.114	0.884	10.653				25.183	0.884	10.683

In Table 13 the reference point was adjusted to a more realistic and reachable level, coverage was decreased to 0.88, power was increased to 25, and bmep was kept at the same level with a value of 10. With this preference information all methods produced rather steady improvement on function values (minor part of blank cells) as the number of function evaluations allowed was increased. Even at the very beginning, with under 25 function evaluations, all methods produced solutions quite close to a given reference point, and after 175 function evaluations all methods had exceeded given reference point values. With this preference information ACH and GUESS behaved almost identically, whereas STOM produced a slightly lower bmep value and a slightly higher maximum power.

From Tables 12 and 13 we can deduce that reference point based scalarization functions are very usable because of their straightforward use of prefer-

ence information, which directly relate to real function values. Also, using an extremely modest number of iterations, rather good solutions can be found. It is possible to assess their usability from the designers point of view and further adjust preference information to meet the designer's goal. It is also beneficial for the designer to be able to get a grasp of the reachable objective function value ranges by playing with preference information, and thus learn something of the system itself. This, in turn, offers a natural basis for the use of interactive methods. In addition, in an interactive solution process, the number of function evaluations used, i.e. computational accuracy, can be adjusted dynamically. At the final stages of the interactive procedure, when the designer decides that the optimal mutual emphasis of separate objectives is found, (s)he can execute the final optimization run with a higher number of function evaluations, ranging from hundreds to thousand, to make certain the resulting solution is optimal.

5.2.3 Interactive solution process using NIMBUS

In Table 14 we present the results of an interactive optimization run using NIMBUS software [45] with in-built ACH scalarization and the CRS2 algorithm as an underlying solver. With NIMBUS several different scalarization functions can be employed synchronously, but as the data in Tables 10, 11, 12 and 13 indicates, differences in results between scalarization functions are negligible, and thus we chose to use only ACH scalarization in order to decrease the computational burden.

Normally IND-NIMBUS initially calculates the ideal and nadir objective vectors, and updates these values during the optimization run as necessary. In our case, these values were given beforehand manually. The nadir objective vector values for maximum power, coverage and bmep were 0.0, 0.0, 0.0 respectively, and the ideal objective vector values were 34.0, 1.0, 15.0. The values for both vectors were selected so that they are not reachable in real life.

With IND-NIMBUS it is possible to generate a set of intermediate Pareto optimal solutions between any two Pareto optimal solutions to assess, for example, variations in function values. This feature was not exploited in our example case.

In Table 14 we present, for each classification step, the desirable objective function values (Cls), corresponding results (Res) and the number of function evaluations (Evals) used in that step. We denote solution as values for all three functions with triplet (f_1, f_2, f_3) . In Step 1 the neutral compromise solution was calculated using 200 function evaluations. After that, the classification steps were executed with only 50 function evaluations for each new classification. Only the final two steps were executed to 1000 evaluations.

In Step 1 a neutral compromise solution was calculated using 200 iterations, resulting in the solution (26.0, 0.78, 11.3). In Step 2 new classification was done based on the result of Step 1. The aim was to gain higher coverage, thus power was allowed to degrade to 25.0, coverage was required to improve to 0.85 and for bmep slight degradation to 11.0 was allowed. The resulting solution (24.6, 0.86,

TABLE 14 Interactive solution process.

		Pwr	Cov	Bmp
Step 1	Cls	-	-	-
Evals: 200	Res	26.0	0.78	11.3
Step 2	Cls	$I^{bound}(25.0)$	$I^{asp}(0.85)$	$I^{bound}(11.0)$
Evals: 50	Res	24.6	0.86	10.9
Step 3	Cls	$I^{bound}(24.0)$	$I^{asp}(0.9)$	$I^{asp}(11.5)$
Evals: 50	Res	22.7	0.87	11.1
Step 4	Cls	$I^{asp}(25.5)$	$I^{asp}(0.9)$	$I^{bound}(10.5)$
Evals: 1000	Res	25.5	0.9	10.6
Step 5	Cls	$I^{sat}(25.5)$	$I^{bound}(0.88)$	$I^{asp}(11.2)$
Evals: 1000	Res	25.3	0.87	11.2

10.9) was quite close to the given classification regardless of only 50 iterations.

In Step 3 classification was done based on the result of Step 2. The aim was to gain even more coverage and slightly more bmep, thus power was allowed to degrade to 24.0, and coverage and bmep were required to improve to 0.9 and 11.5, respectively. As a result (22.7, 0.87, 11.1) power dropped quite a lot and coverage and bmep were increased only slightly, probably due to the tiny number of 50 iterations. In Step 4 we continued classification from the solution of Step 3. We wanted to approach the final solution with quite a high demand for power and coverage (25.5 and 0.9), at the same time we allowed degradation of bmep to 10.6. With 1000 function evaluations a given classification was achieved and partly exceeded with the solution (25.5, 0.9, 10.6). This solution had the power peak at a quite high level (25.5 hp at 13250 rpm), thus, in classification at Step 5 we wanted to keep power at the same level (25.5), and relaxed the demand for coverage slightly to 0.88 and required bmep to improve to 11.2. The result was as expected, and all the objectives followed a given classification quite accurately with the solution (25.3, 0.87, 11.2), and power peak settled at a reasonable level with 25.3 hp at 12500 rpm. In Figure 28 the power curve and the corresponding exhaust pipe shape of the final solution of the interactive optimization procedure is displayed.

The interactive optimization procedure allowed us to guide the solution to the desired direction step by step, and also learn about what kind of solutions are attainable. In intermediate steps, only a minimal number of the 50 function evaluations were used, yet the solutions obtained followed the given classifications accurately enough to allow guidance of solutions to a desired direction. In the final two steps, 1000 function evaluations were used, and this led to a good final solution, as the numbers and figures in Figure 28 indicate: the power curve is smooth and wide, and the exhaust pipe shape resembles the shapes used in real racing engines.

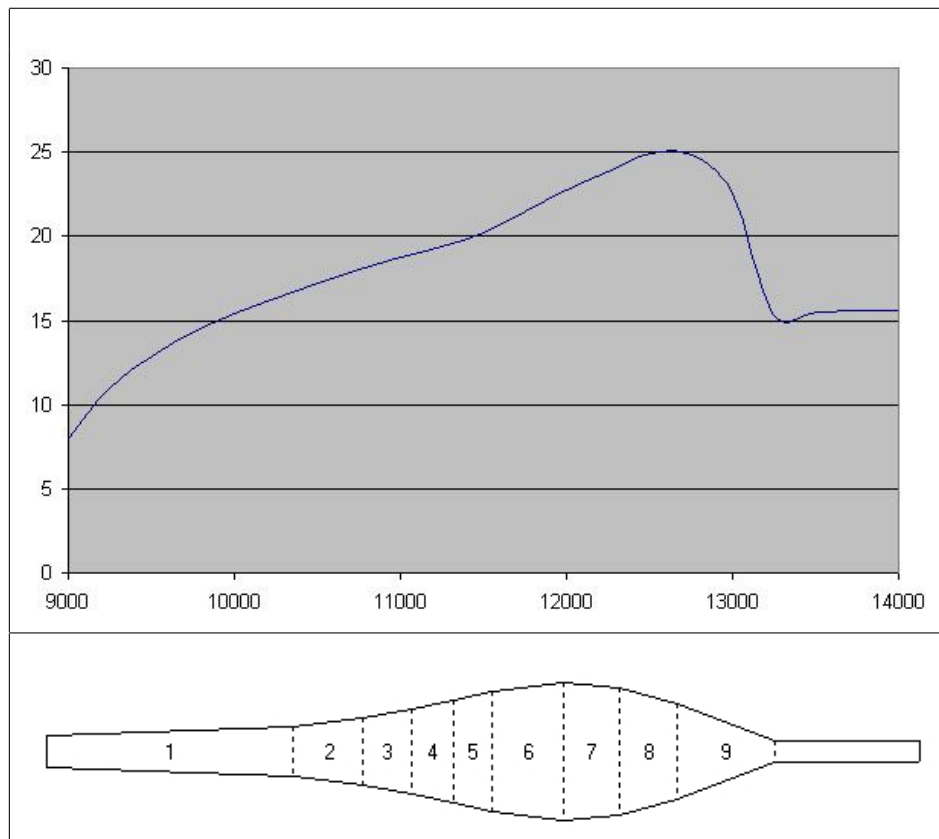


FIGURE 28 Power curve and corresponding pipe shape of the final interactive solution (25.3, 0.87, 11.2).

6 DISCUSSION AND CONCLUSIONS

In many industrial processes it is imperative to be able to control features (quality, production cost, strength, etc.) of the end product. Processes can be simulated using specific mathematical models, upon which the effects of different input variable values of the process can be tested to produce a desired end result. Often, input variable values are adjusted by trial-and-error, based on the designer's expertise, producing results of varying quality.

Instead, with simulation based optimization, the designer no longer adjusts design variable values manually, but rather he/she describes the desired end result with a higher abstraction level, and an appropriate optimization algorithm finds such values for the design variables, that give an optimal result in the sense the designer wished.

In this study, we considered three important characteristics of simulation based optimization systems: efficiency (generally, running the simulator is computationally expensive) in terms of objective function evaluations, global search instead of local one (to avoid local minima often present because of high nonlinearity), and the treatment of the problem as a multiobjective optimization one. Additionally, the gradient information is often unavailable, as was the case also with our problem.

In our study, the aim was to relieve the burden of the designer in the design process by replacing the manual trial and error process with a computer implemented optimization system, which is capable of operating without continuous human intervention, as is required in the time consuming trial and error processes. With an optimization system, we can achieve optimal results, in contrast to the trial and error process, where the process ends when some satisfactory result, which may still be arbitrarily far from the real optimum, is found.

To relieve the burden of design process we created a system for simulation based optimization, which consisted of several heterogenous software modules. The most prominent of those were the engine simulator and different optimization algorithms. With a specifically written adapter and interfacing code, our system enabled virtually the use of any optimization algorithm and simulator combination.

Our example case concerned internal combustion engine design. More specifically, we optimized the performance of a 2-stroke engine by altering the exhaust pipe shape. We created some novel ways to model the exhaust pipe shape using only a modest number of design variables, while retaining the generality and flexibility of the model. We defined several ways for measuring the goodness of a particular engine design (objective functions), and we also addressed the important topic of viewing the optimization problem from the perspective of the vehicle performance as a whole. From these objectives we formulated both single and multiobjective optimization problems, which we solved using efficient global optimization algorithms and different scalarization methods. As a whole, we solved the engine design problem with a somewhat more practical value than in some of the previous studies where objective functions were defined in a trivial way and objective function evaluations were splurged, which is not allowable if objective function values are produced via a time-consuming simulation process.

To create an efficient, global and multiobjective optimization system, we had to do the following: identify or create an efficient and flexible way to model the exhaust pipe shape, select an efficient global optimization algorithm, and select appropriate scalarization function(s) to be used. To justify our selections, we conducted a series of comparisons as depicted below. Furthermore, we employed the interactive NIMBUS method, and made attempts to reduce the number of function evaluations required and yet were able to guide the optimization process to the desired direction.

With the single objective optimization task we compared four different ways to model the exhaust pipe shape using the Niederreiter random sampling and the CRS2 optimization algorithm. The Bezier model, which produced the best objective function values, with respect to function evaluations used, was deemed the best. This was probably due to the low number of design variables ($n = 6$) used, and also to the flexibility of the model (i.e., it was able to capture large variety of possible pipe shapes). In general, we can assume that while the flexibility of the model increases, and the number of design variables decreases, the usability and efficiency of the model improves from the perspective of the optimization algorithm.

Once the Bezier model had been selected, different global optimization algorithms were compared using the Bezier model and a single objective optimization task. Here, the CRS2 algorithm was chosen, because it proved to be superior, and it was also pleasant to use, because there were no parameters except the population size to be set for the algorithm itself. The DE algorithm got very close in performance to CRS2, and it is possible that with some additional parameter tweaking it might have outperformed CRS2. However, the drawback of the DE algorithm lies in the difficulty of choosing proper values for all the parameters and the DE strategy. Somewhat surprisingly, hybrid optimization algorithms performed poorly, probably due to their complex structure. The method based on statistical modeling of objective function performed otherwise well, but the overhead of the algorithm itself proved to be prohibitive. One possible way to improve its performance could be to decrease the rate of fitting of the statisti-

cal model. Currently the model is fitted after each function evaluation, and as the number of already performed evaluations grows, fitting of the model takes a very long time.

For the multiobjective optimization case, and for the comparison of different scalarization functions, we chose to use the Bezier model with the CRS2 algorithm. Multiple objectives were scalarized into a single objective using several different types of scalarization functions, and their properties were compared. In our case, the method with no preference information produced a rather good solution, although the quality of the solution obtained obviously depends on nadir and ideal point values, which were only estimated in this study.

From the class of scalarization methods that use preference information we used both weight and reference point based functions. Weight based function proved to be impractical, because there was no direct relationship between weights and objective function values. Also results of current weighting were seen after a completion of the whole time consuming optimization run. This made it very difficult to control objective function values.

Instead, reference point based functions provided practical means to represent the preference information and to control the solution process. A reference point consists directly of desirable function values, and whenever reference point values were reachable, a solution was found with quite a low number of function evaluations. According to our comparison, there were no significant differences in solutions that different reference point based scalarization functions used produced.

The interactive NIMBUS method offers some benefits over plain use of scalarization function. With NIMBUS the user classifies function values of an already found solution by allowing them either improve, degrade, keep their current value, or have whatever value. From the current solution, a new solution with user preferences is created, and the iterative process is continued until the decision maker finds the final solution. With an interactive process, the decision maker can also learn about the nature of the problem. By using different classifications it is possible to learn how objectives affect each other, and also what kinds of solutions in general can or cannot be reached. This gives the decision maker a good opportunity to solve the problem as a whole, instead of tweaking each of the objectives at a time. A deeper understanding of the problem is likely to make the decision maker very confident of the quality of the final solution.

With the interactive method, we also could decrease the number of function evaluations to a rather low level, and yet guide the solution process to a desired direction. This means time savings, and retains the interactive nature (which is endangered by time consuming simulation runs) of the process to some extent. To be assured of the goodness of the final solution, we used a larger number of function evaluations to get to the final solution.

In this study we compared several global optimization algorithms, different ways to model exhaust pipe shape, and different scalarization functions. Also, several ways to define goodness of the particular engine were formulated. By

employing multiobjective methods in general, and interactive NIMBUS method in particular, we could guide the solution process and also learn about interrelationships between different objective functions.

In practice, we discovered the usefulness and practicality of multiobjective optimization in general and of reference point scalarization methods in particular. By using them we could guide the solution process to the direction we wished by simply giving desirable values for each of the objective functions. Also, use of the interactive NIMBUS method allowed us to decrease the number of function evaluations and yet be able to guide the solution process efficiently.

Results gained using the optimization system seem very credible by the traditions of the trade. The optimized pipe shapes resemble closely to the ones seen in the aftermarket performance exhaust pipes, and respective power curves suggest high usability for the optimized pipes. Unfortunately, resources budgeted for this study did not allow us to empirically verify the real performance of the exhaust pipes neither using the dynamometer nor on the racetrack, and thus we can say that the results are optimal only if the simulator itself is trustworthy.

Anyhow, we believe that the methods depicted in this study can be exploited in several different fields of engineering. Especially, we have addressed solving of nonlinear, multiobjective and multimodal problems (frequently faced in engineering problems), without forcing them to a restrictive single objective format in the modelling phase, or solving them only locally. With our approach, problems retain their original characteristics, and the designer / decision maker can freely pursue a solution which best corresponds to his/her desires, while having certainty that the solution found is globally optimal.

In the future, the two key areas for further research will be the development of even more efficient global optimization algorithms and the usability of current tools. Although computational power of hardware is constantly increasing, so is the complexity and reliability of the simulation models. Thus, the need for more efficient global optimization algorithms is evident. Although the CRS2 algorithm performed well, there are also interesting prospects in the direction of surrogate modelling and advanced clustering techniques as parts of global optimization algorithms.

In addition, usability and ease of use play an important role when the designer is familiarizing with, or evaluating an optimization system. In the current system, objective function values are both numerically and visually displayed, but the designer would benefit if (s)he could use some additional information. In our example case of engine design process, an additional display with power curves and exhaust pipe shapes related to each solution would have been useful.

Furthermore, although not with the highest order of importance from the perspective of development of optimization tools and methods, it would be interesting to study how well the optimized exhaust pipes of this study perform compared to the best known aftermarket performance exhaust pipes, either in simulated environments, or in real life applications.

REFERENCES

- [1] M.M. Ali and C. Khompatporn and Z.B. Zabinsky (2005): A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems, *Journal of Global Optimization* 31:635-672.
- [2] M.M. Ali and C. Storey (1994): Modified Controlled Random Search Algorithms, *International Journal of Computer Mathematics* 54: 229-235.
- [3] M.M. Ali, C. Storey and A. Törn (1997): Application of some Stochastic Global Optimization Algorithms to Practical Problems. *Journal of Optimization Theory and Applications* 95: 545-563.
- [4] I. Arbuckle, S. Naylor and M. Worthington (2002): Optimization Strategies Applied to Exhaust System Design. Published in *Computer Simulation for Automotive Applications*. R. Ballinger and G. Steen (editors). Society of Automotive Engineers, Inc., 111 - 117.
- [5] J. Banks (1998): *Handbook of Simulation : Principles, Methodology, Advances, Applications, and Practice*. John Wiley & Sons.
- [6] M. Bartholomew-Biggs, S. Brown, B. Christianson, L. Dixon (2000): Automatic differentiation of algorithms. *Journal of Computational and Applied Mathematics* 124, 171-190.
- [7] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty (1993): *Nonlinear Programming: Theory and Algorithms*. John Wiley and Sons, New York, 2nd edition.
- [8] P.K. Bergey, C. Ragsdale (2005): Modified differential evolution: a greedy random strategy for genetic recombination. *Omega* 33, 255-265.
- [9] H.-G. Beyer and H.-P. Schwefel (2002): Evolution Strategies: A Comprehensive Introduction. *Natural Computing*, Vol. 1, No. 1, 3-52.
- [10] P. Bezier (1970): *Emploi des Machines a Commande Numerique*". Masson et Cie. Paris.
- [11] L.T. Biegler (1989): Chemical Process Simulation. *Chemical Engineering Progress*, 85(10), 50-61.
- [12] G. P. Blair (1996): *Design and Simulation of two-stroke Engines*. Society of Automotive Engineers, Inc.
- [13] J. Bradley (1996): *The Racing Motorcycle - A technical guide for constructors*. Broadland Leisure Publications.
- [14] N. Cressie (1990): The origins of kriging. *Mathematical Geology*, 22, 197-202.
- [15] A. Dave and G.J. Hampson (2003): Robust Engine Design Using Engine Simulations. Published in *Modeling of SI Engines*. T. Morel (editor). Society of Automotive Engineers, Inc., 65 - 77.

- [16] K. Deb (2001): *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons.
- [17] J. F. Elder IV (1992): Global Rd Optimization when Probes are Expensive: the GROPE Algorithm. *Proceedings IEEE International Conference on Systems, Man, and Cybernetics, Chicago, Illinois, October 18-21, 1992*
- [18] T. D. Gillespie (1992): *Fundamentals of Vehicle Dynamics*. Society of Automotive Engineers, Inc.
- [19] P. E. Gill, W. Murray, M. H. Wright (1981): *Practical Optimization*. Academic Press.
- [20] T.J. Gogg and J.R.A. Mott (1996): *Improve Quality & Productivity with Simulation, 3rd Edition*. JMI Consulting Group.
- [21] A. Griewank (2000): *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial & Applied Mathematics.
- [22] J. Hakanen (2004): *Challenges of Process Integration to Interactive Multiobjective Optimization*. Licentiate Thesis, Department of Mathematical Information Technology, University of Jyväskylä.
- [23] J. Haslinger and R.A.E. Mäkinen (2003): *Introduction to Shape Optimization - Theory, Approximation and Computation*. Siam. Philadelphia.
- [24] A-R. Hedar (referenced at 11.11.2005): *Global Optimization* <http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/go.htm>.
- [25] A-R. Hedar and M. Fukushima (2002): Hybrid simulated annealing and direct search method for nonlinear unconstrained global Optimization. *Optimization Methods and Software*, 17, 891-912.
- [26] A-R. Hedar and M. Fukushima (2003): Minimizing multimodal functions by simplex coding genetic algorithm. *Optimization Methods and Software*, 18, 265-282.
- [27] A-R. Hedar and M. Fukushima (2003): Heuristic Pattern Search and Its Hybridization with Simulated Annealing for Nonlinear Global Optimization. *Optimization Methods and Software*, 19, 291-308.
- [28] H. Heisler (2002): *Advanced Vehicle Technology*. Butterworth-Heinemann.
- [29] M. Hofer (referenced at 28.11.2005): *Bezier Curve Demo*. <<http://www.math.ucla.edu/baker/java/hofer/Bezier.htm>>.
- [30] R. Hooke and T. Jeeves (1961): Direct search solutions of numerical and statistical problems. *Journal of the Association for Computing Machinery*, 8, 212-229.

- [31] W. Huyer and A. Neumaier (1999), Global optimization by multilevel coordinate search. *Journal of Global Optimization* 14, 331-355
- [32] D.R. Jones, M.Schonlau and W.J.Welch (1998): *Efficient Global Optimization of Expensive Black-Box Functions*. Kluwer Academic Publishers.
- [33] C.T. Kelley (1999): Detection and remediation of stagnation in the Nelder-Mead algorithm using a sufficient decrease condition. *SIAM Journal of Optimization*, 10, 43-55.
- [34] M. Keveney (referenced at 12.1.2006): *Animated Engines*. <<http://www.keveney.com/Engines.html>>.
- [35] S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi (1983): Optimisation by simulated annealing. *Science*, 220, 671-680.
- [36] J. Knowles (referenced at 11.11.2005): *ParEGO: an algorithm for multiobjective optimization of expensive functions* <<http://dbk.ch.umist.ac.uk/knowles/parego/>>.
- [37] J. Knowles (2006): *ParEGO: A Hybrid Algorithm with On-line Landscape Approximation for Expensive Multiobjective Optimization Problems*. *IEEE Transactions on Evolutionary Computation*. 10 (1), 50-66.
- [38] T. G. Kolda, R. M. Lewis, V. Torczon (2003): *Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods*. *SIAM Review*, Vol. 45, No. 3, 385-482.
- [39] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright (1998): Convergence properties of the Nelder-Mead Simplex method in low dimensions. *SIAM Journal of Optimization* Vol. 9, No. 1, 112-147.
- [40] J. Lampinen (2003): *Cam shape optimization by genetic algorithm*. *Computer-Aided Design* 35, 727-737.
- [41] J. van Leersum (1998): *A numerical model of a high performance two-stroke engine*. *Applied Numerical Mathematics* 27, 83-108.
- [42] Lidl and H. Niederreiter(1983): *Finite Fields*, Vol. 20 in the *Encyclopedia of Mathematics and its Applications*. Addison-Wesley.
- [43] K. Miettinen (1999): *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, Boston.
- [44] K. Miettinen and M.M. Mäkelä (2002): *On scalarizing functions in multiobjective optimization*. *OR Spektrum* 24, 193-213.
- [45] K. Miettinen and M.M. Mäkelä (2006): *Synchronous approach in interactive multiobjective optimization*. *European Journal of Operational Research* 170, 909-922.

- [46] K. Miettinen and M.M. Mäkelä (referenced at 13.6.2006): WWW-NIMBUS. <<http://nimbus.mit.jyu.fi/>>.
- [47] W. F. Milliken and D. L. Milliken (1995): Race Car Vehicle Dynamics. SAE International.
- [48] M. Mitchell (1998): An Introduction to Genetic Algorithms. The MIT Press.
- [49] J.A. Nelder and R. Mead (1965): A simplex method for function minimization. *Computer Journal*, 7, 308-313.
- [50] H. Niederreiter (1992): Random number generation and quasi-Monte Carlo methods. Society for Industrial and Applied Mathematics.
- [51] C.D. Perttunen, D.R. Jones and B.E. Stuckman (1993): Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Application*, 79, 157-181.
- [52] M.J.D. Powell (1964): An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives. *Computer Journal* 7, No. 2, 155-162.
- [53] W.L. Price (1977): Global Optimization by Controlled Random Search. *Computer Journal* 20: 367-370.
- [54] W.L. Price (1983): Global optimization by controlled random search. *Journal of Optimization Theory and Applications*, 40, 333-348.
- [55] W. W. Pulkrabek (2003): Engineering Fundamentals of the Internal Combustion Engine, 2nd Edition. Prentice Hall.
- [56] J. Sacks, W. Welch, T. Mitchell and H. Wynn (1989): Design and analysis of computer experiments (with discussion), *Statistical Science*, 4, 409-435.
- [57] M.J. Sasena (2002): Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations. Dissertation. University of Michigan.
- [58] P.K. Senecal, E. Pomraning and K.J. Richards (2002): Multi-Mode Genetic Algorithm Optimization of Combustion Chamber Geometry for Low Emissions. Published in *Compression Ignition Combustion and In-Cylinder Diesel Particulates and NOx Control*. R. Radovanovic, S.J. Charlton, C.Y. Choi, Y. Daisho, A. Osborn (editors). Society of Automotive Engineers, Inc., 91 - 101.
- [59] D.P. Solomatine (1998): Genetic and other global optimization algorithms - comparison and use in calibration problems. Balkema Publishers.
- [60] R. Storn (referenced at 16.11.2005): Differential Evolution Homepage <<http://www.icsi.berkeley.edu/storn/code.html>>.

- [61] R. Storn, K. Price (1997): Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11, 341-359.
- [62] S. Tiittanen (1983): *Motocross-mekaanikon käsikirja*. Published by the author.
- [63] A. Törn and A. Zilinskas (1989): *Global Optimization*. Springer-Verlag.
- [64] A. Törn (referenced at 10.01.2006): *Global Optimization*. <<http://www.abo.fi/~atorn/Globopt.html>>.
- [65] A. Törn, S. Viitanen (1994): Topographical Global Optimization Using Pre-Sampled Points. *Journal of Global Optimization* 5, 267-276.
- [66] J.B. Vosa, A. Rizzib, D. Darracqç, E.H. Hirschel (2002): Navier-Stokes solvers in European aircraft design. *Progress in Aerospace Sciences* 38, 601-697.
- [67] W. J. Welch, R. J. Buck, J. Sacks, H. P. Wynn, T. J. Mitchell, and M. D. Morris (1992). Screening, predicting, and computer experiments, *Technometrics*, 34, pp. 15-25.
- [68] A. P. Wierzbicki (1999): Reference point approaches. Published in *Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory, and Applications*. T. Gal, T. J. Stewart and T. Hanne (editors). Kluwer Academic Publishers. 9-1 - 9-39.
- [69] I. Williams (referenced at 12.1.2006): Ian Williams Tuning Homepage. <<http://www.iwt.com.au/index2.html>>.
- [70] J. Y. Wong (2001): *Theory of Ground Vehicles*, 3rd Edition. John Wiley & Sons, Inc.
- [71] T. Ye and S. Kalyanaraman (2003): A Recursive Random Search Algorithm For Large-Scale Network Parameter Configuration. Published in *Proceedings of the 2003 ACM SIGMETRICS*. 196 - 205.
- [72] W. Zucker, R.R. Maly and S. Wagner (2001): Evolution-Strategy Based, Fully Automatic, Numerical Optimization of Gas-Exchange Systems for IC Engines. Published in *SI Engine Modeling and Simulation*. T. Morel (editor). Society of Automotive Engineers, Inc., 29 - 41.

YHTEENVETO (FINNISH SUMMARY)

Useissa teollisissa prosesseissa on välttämätöntä pystyä hallitsemaan lopputuotteen ominaisuuksia (laatua, tuotantokustannuksia, lujuutta, yms.). Prosesseja voidaan simuloida erityisiä matemaattisia tietokonemalleja käyttäen. Näillä malleilla voidaan testata erilaisten suunnittelumuuttujien vaikutusta lopputulokseen, ja pyrkiä näin saavuttamaan haluttu lopputulos. Usein suunnittelumuuttujia säädetään suunnittelijan ammattitaitoon perustuen yrityksen ja erehdyksen periaatteella, mutta näin menetellen lopputuloksen laatu saattaa vaihdella suuresti.

Perinteisen yrityksen ja erehdyksen periaatteella tapahtuvan suunnittelun vaihtoehdoksi tässä työssä esitellään simulaatiopohjaisen optimoinnin tarjoamia mahdollisuuksia suunnitteluprosessiin. Simulaatiopohjaisessa optimoinnissa suunnittelija ei enää säädi yksittäisiä suunnittelumuuttujia, vaan sen sijaan kuvaa halutun lopputuloksen korkeammalla abstraktiotasolla, ja sopiva optimointialgoritmi hakee tämän lopputuloksen tuottavat arvot suunnittelumuuttujille.

Tämä työ keskittyy polttomoottorin ominaisuuksien (kuten hyötysuhde, suurin teho ja moottorin ominaisuuksien sopivuus vaihteistovälityksiin) optimointiin. Käsiteltävä ongelma, kuten monet muutkin tosielämän tekniset suunnitteluongelmat, asettaa muutamia erityisvaatimuksia käytettävälle optimointijärjestelmälle. Tässä tutkimuksessa keskitytään tarkastelemaan kolmea tärkeää erityispiirrettä: tehokkuutta (yleisesti ottaen simulaattorin ajaminen on laskennallisesti kallista) suhteessa objektifunktion evaluointeihin, globaalia hakua lokaalin sijaan (ongelmat ovat usein epälineaarisia ja sisältävät useita lokaaleja minimejä), sekä ongelman käsittelemistä monitavoitteisena optimointiongelmana. Käytännössä ei aina gradientti-informaatiota ole saatavilla, ja näin on myös tämän tutkimuksen esimerkkitapauksessa. Tässä työssä tarkastellaan muutamia näkökulmia ongelmaan, sekä esitetään tehokas tapa ratkaista kyseinen ongelma.

Työn tavoitteena on ratkaista tehokkaasti (käyttäen vain mahdollisimman vähäinen määrän objektifunktion evaluointeja), globaalisti, sekä monitavoitteisella tavalla monimutkaisen käytännönläheinen ongelma. Muuttamalla 2-tahtimoottorin pakoputken mittasuhteita ja muotoa vaikutetaan moottorin ominaisuuksiin, kuten tehoon ja vääntömomenttiin. Käytetyt objektifunktiot ovat monimutkaisempia kuin joissakin aikaisemmissa tutkimuksissa, ja lisäksi tarkasteltavana on koko laitteen suorituskyvyn kannalta tärkeä ongelma, moottorin ominaisuuksien sovittaminen ajoneuvon vaihteistovälityksiin. Lisäksi tämä työ käsittelee tosielämässä usein kohdattavaa ongelmaa, nimittäin useita ristiriitaisia objektifunktioita sisältävän ongelman ratkaisemista monitavoiteoptimoinnin skalarsointifunktioita hyödyntäen.