JYVÄSKYLÄ STUDIES IN COMPUTING
142

Ernesto Mininno

Advanced Optimization Algorithms for Applications in Control Engineering

JYVÄSKYLÄN YLIOPISTO

JYVÄSKYLÄ STUDIES IN COMPUTING 142

Ernesto Mininno

Advanced Optimization Algorithms for Applications in Control Engineering

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston Agora-rakennuksen auditoriossa 3
joulukuun 12. päivänä 2011 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in the building Agora, auditorium 3, on December 12, 2011 at 12 o'clock noon.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2011

Advanced Optimization Algorithms for Applications in Control Engineering

JYVÄSKYLÄ STUDIES IN COMPUTING 142

Ernesto Mininno

Advanced Optimization Algorithms for Applications in Control Engineering

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2011

Editors
Timo Männikkö
Department of Mathematical Information Technology, University of Jyväskylä
Pekka Olsbo, Ville Korkiakangas
Publishing Unit, University Library of Jyväskylä

URN:ISBN:9789513945404
ISBN 978-951-39-4540-4 (PDF)
ISSN 1456-5390

ISBN 978-951-39-4539-8 (nid.)
ISSN 1456-5390

Jyväskylä University Printing House, Jyväskylä 2011

ABSTRACT

Mininno, Ernesto
Advanced Optimization Algorithms for Applications in Control Engineering
Jyväskylä: University of Jyväskylä, 2011, 72 p.(+included articles)
(Jyväskylä Studies in Computing
ISSN 1456-5390; 142)
ISBN 978-951-39-4539-8 (nid.)
ISBN 978-951-39-4540-4 (PDF)
Finnish summary
Diss.

In the last ten years optimization in industrial applications has obtained an increasing attention. In particular it has been demonstrated useful and effective in the solution of control problems. The implementation of an optimization algorithm on a real-time control platform must cope with the lack of a full power computer, thus it must use a very low amount of memory and computational power. On the other hand the presence of nonlinearities, sensors and approximations injects in the signals of the control loop some noise, resulting in a noisy fitness function to be optimized. In this work both issues are addressed in order to show how a novel algorithmic design can arise from the solution of these implementation problems, often underestimated in the theoretical approach. This thesis proposes a set of novel algorithmic solutions for facing complex real-world problems in control engineering. Two algorithms addressing the optimization in the presence of noise are discussed. In addition, a novel adaptation system inspired by estimation of distribution paradigm is proposed to handle highly multimodal fitness landscapes. A crucially important contribution contained in this thesis is the definition of compact Differential Evolution for optimization problems in presence of limited hardware. Finally an evolution of the latter algorithm in the fashion of Memetic Computing is proposed with reference to an industrial application problem.

Keywords: Compact Algorithms, Noise analysis, Evolutionary Algorithms, Real Time Control Systems, Differential Evolution, Robotic control.

Author	Dr. Ernesto Mininno Department of Mathematical Information Technology University of Jyväskylä Finland
Supervisors	Professor Tommi Kärkkäinen Department of Mathematical Information Technology University of Jyväskylä Finland
	Dr. Ferrante Neri Department of Mathematical Information Technology University of Jyväskylä Finland
Reviewers	Prof. Stefano Cagnoni Department of Computer Engineering. University of Parma. Italy
	Dr. Chi-Keong Goh Advanced Techno Centre Rolls-Royce Singapore Singapore
Opponent	Prof. Dr.-Ing. Hendrik Richter Elektrotechnik und Informationstechnik Institut Leipzig Germany

PREFACE

A large section of computer science devotes its research interest toward optimization and design of optimization algorithms. The nature of many real-world applications and the intrinsic features of the problems, such as conflicting objectives, time and space limitations, the presence of uncertainties, makes the problems extremely challenging. In order to address this class of problems, especially when no hypotheses on the mathematical structure of the optimization problem could be done, e.g. due to the lack of an explicit mathematical description of the objective function, computational intelligence optimization has been developed. Computational intelligence optimization is a subject which studies optimization problems by means of computational intelligence structures. From an alternative perspective, computational intelligence optimization is a subject which attempts to tackle optimization problem which cannot be solved by means of traditional exact optimization methods. In this sense, computational intelligence optimization is an important for addressing industrial problems or, more generally, real-world problems.

This thesis work addresses real-world optimization problems by means of computational intelligence techniques. This work has been produced thinking that algorithmic development and applications are two aspects of the same subject and are connected by a conceptual inter-dependency. In other words, each algorithm contained in this thesis has been designed by thinking at one real-world problem or to address a specific problem feature characterizing an engineering process. More specifically, this thesis contains five published articles, three of them published in journals and two of them contained in conference proceedings. Robotics and control engineering are the main application domains considered in this work. Two relevant aspects plaguing these fields of technology are considered in great details; these aspects are the presence of noise in the fitness computation and the hardware limitations, especially in the memory structures.

ACKNOWLEDGEMENTS

Foremost I would like to thank my colleague and dear friend Ferrante Neri, for his warm and kind support in these years in Jyväskylä. Without him everything would have been much harder and less enjoyable.

Next I would thank my main supervisor Tommi Kärkkäinen for his constant support during this work.

I would thank also Giovanni Iacca for the inspiring scientific conversation and the constant scientific support in the prosecution of our work.

I would like to dedicate this work to my parents Antonio and Susi, whose support and believing in my ability as a researcher have been of great inspiration and confidence to proceed to this goal.

LIST OF FIGURES

FIGURE 1 Schematic of a general evolutionary algorithm 22
FIGURE 2 EA pseudo-code 23
FIGURE 3 GA pseudo-code 24
FIGURE 4 ES pseudo-code 25
FIGURE 5 DE pseudo-code 27
FIGURE 6 Direct optimization framework 32
FIGURE 7 Indirect optimization framework 32
FIGURE 8 Equivalent circuit of the armature electrical dynamics 36
FIGURE 9 DC motor schematic 37
FIGURE 10 Cascade of elementary components of the controller 38
FIGURE 11 Anti-windup algorithm 39
FIGURE 12 Summary of the GA configuration 40
FIGURE 13 Speed and load references during the training test 42
FIGURE 14 Simulation of substitution process on the virtual population of the cGA 48
FIGURE 15 cGA pseudo-code 48
FIGURE 16 pe-cGA pseudo-code 50
FIGURE 17 ne-cGA pseudo-code 50
FIGURE 18 Normalization of a truncated Gaussian curve (with standard deviations $\sigma = 1$) so as to obtain a probability density function with a non-zero support only in the normalized interval $[-1, 1]$. 53
FIGURE 19 ne-cGA pseudo-code 54

CONTENTS

ABSTRACT
PREFACE
ACKNOWLEDGEMENTS
LIST OF FIGURES
CONTENTS
LIST OF INCLUDED ARTICLES

1 INTRODUCTION 13

2 OPTIMIZATION PROBLEMS AND ALGORITHMS 15
2.1 Basic Definitions and Notation: Optimization Problems 15
2.1.1 Randomness in optimization 16
2.1.2 No Free Lunch Theorems 17
2.2 A Short Introduction on Gradient based Methods 19
2.3 Derivative Free Search: Hooke-Jeeves and Nelder Mead 20
2.4 Computational Intelligence Optimization 21
2.4.1 Introduction to Evolutionary Algorithms 21
2.4.2 Genetic Algorithms 24
2.4.3 Evolution Strategies 25
2.4.4 Differential Evolution 26
2.4.5 Swarm Intelligence and Particle Swarm Optimization 27
2.4.6 Memetic Computing 28

3 WHY ARE REAL-WORLD APPLICATIONS MORE CHALLENGING THAN "TOY PROBLEMS"? 30
3.1 Control Engineering: Generalities 30
3.1.1 Suitability 30
3.1.2 Unsuitability 31
3.1.3 Online optimization framework 31
3.1.4 EA for the online tuning of controls 33
3.1.5 Control of an Electric Drive 35
3.1.5.1 Dc Drive 36
3.1.5.2 Design on unstructured controllers with GAs 37
3.2 Noisy Fitness Functions 43
3.2.1 Definitions 43
3.2.2 A Brief Survey on Algorithms for Noisy Optimization 44
3.3 Limited Memory and Real-Time implementations 46
3.3.1 Binary Compact Genetic Algorithms 47
3.3.2 Elitism in cGAs 48
3.3.3 Real-Coded Compact Genetic Algorithms 52

4 BUILDING UPON THE STATE-OF-THE-ART IN OPTIMIZATION FOR CONTROL ENGINEERING 56

4.1 Compact Differential Evolution 56
4.1.1 Objectives 56
4.1.2 Results........ 57
4.1.3 Relation with the whole context 57
4.2 Memetic Compact Differential Evolution 57
4.2.1 Objectives 57
4.2.2 Results........ 57
4.2.3 Relation with the whole context 58
4.3 Estimation Distribution Differential Evolution 58
4.3.1 Objectives 58
4.3.2 Results........ 58
4.3.3 Relation with the whole context 58
4.4 Noise Analysis Memetic Differential Evolution 59
4.4.1 Objectives 59
4.4.2 Results........ 59
4.4.3 Relation with the whole context 59
4.5 Noise Analysis Compact Genetic Algorithm 60
4.5.1 Objectives 60
4.5.2 Results........ 60
4.5.3 Relation with the whole context 60

5 CONCLUSION 61

YHTEENVETO (FINNISH SUMMARY) 62

REFERENCES........ 63

INCLUDED ARTICLES

LIST OF INCLUDED ARTICLES

PI E. Mininno, F. Neri, F. Cupertino, D. Naso. Compact Differential Evolution. *IEEE Transactions on Evolutionary Computation, Vol. 15, No. 1*, 2011.

PII F. Neri and E. Mininno. Memetic Compact Differential Evolution. *IEEE Computational Intelligence Magazine*, 2010.

PIII E. Mininno and F. Neri. Estimation Distribution Differential Evolution. *EvoApplications, Part I*, 2010.

PIV E. Mininno and F. Neri. A memetic Differential Evolution approach in noisy optimization. *Memetic Computing*, 2010.

PV E. Mininno and F. Neri. Noise Analysis Compact Genetic Algorithm. *EvoApplications, Part I*, 2010.

1 INTRODUCTION

The introduction of sophisticated research and optimization methods in various fields of science and engineering has produced a strong effort in the design of new, high performance, algorithms. In the last ten years a lot of effort has been spent on the application of optimization techniques in industrial contests. The specific field of application for these algorithms can range from the optimization of the system design, to the tuning of the parameter of a control system. Among them, the application of advanced metaheuristics such as Evolutionary Algorithms (EA) Hart et al. (2004) and Swarm Intelligence Algorithms (SIA) e.g. Eberhart and Kennedy (1995) in online and offline control application has received a lot of attentions. These metaheuristics are often robust search and optimization methods that are able to cope with ill-behaved problem domains, exhibiting attributes such as multimodality, discontinuity, time-variance, randomness, and noise. These approaches have proved particularly successful in problems that are difficult to formalize mathematically, and which are therefore not conducive to analysis. This includes systems that are highly nonlinear, that are stochastic, and that are poorly understood. Problems involving these classes of process tend to be difficult to solve satisfactorily using conventional methods. The metaheustics, as their name says (from ancient Greek "beyond the capability to find"), are general purpose optimizers which do not require any piece of information on the problem such as the explicit structure of the objective function and its gradient. These features of metaheuristics make them attractive for industrial applications. On the other hand, the well known No Free Lunch theorems (NFL), discussed in Wolpert and Macready (1997), have dramatically changed the perspective in the design and application of optimization methods. The theorems state the non-existence of an universally better algorithm over the others. This new perspective led the research community to push more effort in the design and programming new specific algorithms, tailored to the engineering problem to be optimized.

On the other hand many difficulties can arise from the actual implementation of optimization algorithms into the hardware used in industrial applications, such as noise from the sensors and the lack of enough computational power to accomplish both the control and the optimization tasks. For example, robots for domestic purposes (e.g. a vacuum cleaner robot) need to undergo a learning process and solve which can result into a complex optimization problem. This problem must be solved quickly and without counting on a full power computer, due to volume and cost constraints. Obviously, algorithms employing an archive, computationally expensive learning processes or a large population size are not affordable for the hardware.

This thesis is intended as a step further in the process of both designing robust optimization algorithms with knowledge about the optimization problem and taking in consideration the peculiar characteristics of the final implementation that can directly affect the design and the structure of the algorithm itself.

In order to guide the reader through this path, the structure of the work is organized as described in the following. In the second chapter all the general definitions regarding optimization algorithms and problems, with an overview of the most common optimization framework are discussed.

In the third chapter a deep review of the current literature on the application of optimization algorithms to industrial application, with specific attention to control problems, is shown. Then a more accurate introduction to the problem of noise handling in optimization algorithms and a discussion with examples of the issues of memory consumption in real-time optimization is done.

In the fourth chapter a review of the contribution of each paper with respect the whole contest is presented.

2 OPTIMIZATION PROBLEMS AND ALGORITHMS

2.1 Basic Definitions and Notation: Optimization Problems

Mathematical techniques of search and optimization are aimed at providing a formal means for the best decision in real-life optimization problem, such as long-term investment decision, scheduling strategy for delivering systems and dynamic systems control. *Stochastic* search and optimization methods have gained rapidly an important role due to the intrinsic uncertainty in information that may be available for carrying out the task.

Let Θ be the domain of allowable value for a vector $\boldsymbol{\theta} \in \Theta$. The general optimization problem can be formulated as:

- **Problem** Find the values of a vector $\boldsymbol{\theta} \in \Theta$ that minimize a scalar-valued *loss function* $L : \boldsymbol{\theta} \in \Theta \to \Lambda$.

The vector $\boldsymbol{\theta}$ represents a set of parameters that must be picked in the best way. The loss function $L(\boldsymbol{\theta})$ is a scalar measure of the overall performance of the system for a give set of parameters. The loss function is also known as *performance measure, objective function* or *fitness function*, depending on the context and the optimization method.

The optimization problem above can be formally represented as finding the set:

$$\Theta^* \equiv \arg\min_{\boldsymbol{\theta}\in\Theta} L(\boldsymbol{\theta}) = \{\boldsymbol{\theta}^* \in \Theta | L(\boldsymbol{\theta}^*) \leqslant L(\boldsymbol{\theta}) \forall \boldsymbol{\theta} \in \Theta\}. \tag{1}$$

In the equation (1) $\boldsymbol{\theta}$ is a p-dimensional vector of parameters that are being adjusted and $\Theta \subseteq \mathbb{R}$ is the domain for $\boldsymbol{\theta}$. In practical applications, it is not easy to obtain a closed-form analytical solution to (1). One of the main distinctions in optimization is between global and local optimization. In general, it would be preferable to obtain a global optimum solution for the optimization problem (i.e., at least one $\boldsymbol{\theta}^* \in \Theta$, whit each $\boldsymbol{\theta}^*$ providing a lower value of L than any other). In practice, a global solution is not necessarily available and a *local* solution must be accepted. Moreover, considering the inherent limitations of the vast majority of optimization algorithms, it is usually only possible to ensure that an algorithm will approach a local minimum with a limited amount of resource and computational time. However in this work will be considered some stochastic algorithms (i.e. genetic algorithms, compact genetic algorithms, etc.) that are sometimes able to find global optima among multiple local solutions.

2.1.1 Randomness in optimization

A lot of optimization problems, such as control system design problems, contrast with classical deterministic search methods because of the intrinsic uncertainty in the loss function measurement and the deterministic manner used to obtain the search direction at every step of the algorithm. In many practical problems, such information is not available, indicating that deterministic algorithms are inappropriate. Noisy optimization problems can be a result of the presence of sensors, measurement devices, numerical simulators, and other objects which perform measurements and approximations within the objective function calculation. If the noise is unavoidable and cannot be eliminated from the objective function computation, the optimization algorithm should take into account this difficulty of the problem and perform its action notwithstanding the presence of noise. A strong connection between non-smooth optimization and robust statistics could be analyzed, as for example shown in Kärkkäinen and Heikkola (2004).

Here, simply consider a generic process state measurement result as reported in equation 2.

$$Y_i = X_i + \epsilon_i, i = 1 \ldots N. \tag{2}$$

In this situation the "real" process state X is overlapped by the noise ϵ. The loss function L can be then computed as

$$\min_{\theta \in \mathbb{R}^n} L(\theta) = \min_{\theta \in \mathbb{R}^n} \sum_i \|\theta - Y_i\| . \tag{3}$$

In this case, the values of $L(\theta)$ are affected by the noise signal ϵ and must be taken into consideration for the choice of the optimization algorithm.

For this reason, the *stochastic* definition in Spall (2003) for a problem and/or an algorithm apply when:

- **Property A** There is a random noise in the measurements of $L(\theta)$
- **Property B** There is a random choice made in the search direction at each step of the algorithm.

In order to consider Property A, let us study the function,

$$L(\theta) \equiv y(\theta) + \epsilon(\theta), \tag{4}$$

where ϵ represent the noise term. In an optimization problem, it is not necessarily possible to apply common statistical assumption of independent, identically distributed noise. In particular, noise often arise when physical system measurement are used to calculate the loss function. Some specific cases of relevance are:

- Real-time control problems where data as collected "on-the-fly" (online) during a system operation.
- Problems where physical data are processed sequentially on a dynamic system with each value of θ applied to the system sequentially. In some condition each loss evaluation could be affected by the previous one when the system do not reach a steady state condition.

The Property B states that the deliberate introduction of randomness in the search process can improve the convergence speed and make it less sensitive to errors in the computational chain Data/Model/Loss values. For this reason this injection of randomness have beneficial effects as it allows "spontaneous" movements to unexplored areas of the search space Θ that may contain an unexpected good θ value.

Monte Carlo methods are a class of computational algorithms that rely on repeated random sampling to compute their results.The term describes a large and widely used class of approaches that tend to follow a particular pattern:

- Define a domain of possible inputs.
- Generate inputs randomly from the domain using a certain specified probability distribution.
- Perform a deterministic computation using the inputs.
- Aggregate the results of the individual computations into the final results.

The use of Monte Carlo randomness is strictly related to an important class of algorithms that emulate evolutionary principles of optimization; randomness is a central part of both simulated and physical evolution through the introduction of mutations and random crossover between the parents. In a different class of algorithms the injected randomness can be used to compute quantities that act like their deterministic counterparts. For example, in *Simultaneous Perturbation Stochastic Approximation (SPSA)*, presented in Spall (2000) a random approximation of the gradient is used to drive the search direction of the algorithm.

2.1.2 No Free Lunch Theorems

All the algorithms belonging to the class of *Evolutionary Computation* applications were supposed to be valid in general-purpose "black-box" optimization problem. In 1997 Wolpert and Macready Wolpert and Macready (1997) demonstrated the lack of an universal best algorithm in the so called *no free lunch* (*NFL*) theorems. In essence, the thesis is that an algorithm that is effective on one class of problem is *guaranteed* to be ineffective on another one. In particular, if there are N_θ possible values for θ and N_L possible values for the loss, then by direct enumeration there are $(N_L)^{N_\theta}$ possible mappings of θ to possible loss values. The NFL theorems indicate that:

$$\boxed{\text{All the possible algorithms } a_i \text{ perform the same when averaging over all } (N_L)^{N_\theta} \text{ possible mappings from } \Theta \text{ to the output space.}}$$

The NFL theorems are restricted to the discrete domain, but all optimization problems that run on a digital computer meet this constraint because of the intrinsic discretization of Λ (the set of all the admissible values for $L(\theta)$) in 32 or 64 bit representation. In order to have a more formal definition of the NFL theorems, it is possible to define a "sample" of *visited points* as

$$d_m \equiv \{(d_m^\theta(1), d_m^y(1)), \ldots, (d_m^\theta(m), d_m^y(m))\}. \tag{5}$$

Thus $d_m^\theta(i)$ indicates the Θ value of ith successive element in a sample of size m and $d_m^y(i)$ is its associated loss value. The space of all samples of size m is $D_m = (\Theta \times \Lambda)^m$. An optimization algorithm a is represented as a mapping from previously visited sets of points to a single new point in $\Theta : a : d \in D \rightarrow \{x | x \notin d^x\}$. The performance of an algorithm a iterated m times on a loss function L can be measured with $P(d_m^y | L, m, a)$: this is the conditional probability of obtaining a particular sample d_m. With these definitions the following NFL theorem 1 can be enunciated:

Theorem 2.1.1 *NFL 1: For any pair of algorithms a_1 and a_2*

.

$$\sum_L P(d_m^y | f, m, a_1) = \sum_L P(d_m^y | f, m, a_2)$$

Even if the NFL theorems indicate that "all algorithms perform the same", a key qualifier is the "averaging over all possible mappings" statement. This means that when an underling structure of the problem is known, *and* an algorithm uses that structure, it is certainly possible to achieve a better performance with that algorithm than another on the given problem.

Given the above intuition, an informal mathematical representation of the NFL theorems is

$$\boxed{\text{Size of applicability domain} \times \text{Efficiency = Constant.}}$$

So, in essence an algorithm cannot have *both* wide applicability and uniformly high efficiency.

2.2 A Short Introduction on Gradient based Methods

If L has an analytical expression, and under the conditions that L is unimodal and twice differentiable, one can attempt to find its minimum using an analytical method, based on the decomposition of f into a Taylor series Spall (2003). It is possible to formulate the problem as described in equation 6.

$$L(\theta) = L(\theta_0) + g(\theta)(\theta - \theta_0) + (\theta - \theta_0)^T \frac{1}{2} G(\theta_0)(\theta - \theta_0) + \ldots, \tag{6}$$

where $g(\theta)$ is the gradient of L and $G(\theta)$ is the Hessian matrix of L . Since the optimum θ^* of f is a stationary point where $g(\theta^*) = 0$, we can derive that

$$\theta^* = -g(\theta_0)G^{-1}(\theta_0) + \theta_0 \tag{7}$$

A simplified version of this method can be obtained using the *steepest descent* method, which consists in replacing $G^{-1}(\theta_0)$ with the identity matrix and compute

$$\theta_1 = \theta_0 - \gamma g(\theta_0) \tag{8}$$

The iteration of this equation may conduct to values of θ_n closer to the sought extremum than that of θ_{n-1}. The γ value is the size of the algorithm step. The choice of the proper step size however becomes yet another problem, more so since it depends on the objective function. Moreover, this approximation of the above-described analytical solution causes new problems to appear, calling for more elaborate techniques which can fall into two categories: quasi-newton methods attempt to approximate the inverse Hessian matrix in various ways which often require extensive matrix computations, while conjugate gradient methods forgo the Hessian matrix entirely and repose on line optimization in conjugate directions. But even if these methods show fast convergence properties on quadratic, once or twice-differentiable, unimodal functions, their efficiency is not guaranteed on arbitrary functions.

In optimization, quasi-Newton methods (also known as variable metric methods) are algorithms for finding local maxima and minima of functions. Quasi-Newton methods are based on Newton's method to find the stationary point of a function, where the gradient is 0. Newton's method assumes that the function can be locally approximated as a quadratic in the region around the optimum, and use the first and second derivatives (gradient and Hessian) to find the stationary point. In Quasi-Newton methods the Hessian matrix of second derivatives of the function to be minimized does not need to be computed. The Hessian is updated by analyzing successive gradient vectors instead. Quasi-Newton methods are a generalization of the secant method to find the root of the first derivative for multidimensional problems. In multi-dimensions the secant equation is under-determined, and quasi-Newton methods differ in how they constrain the solution, typically by adding a simple low-rank update to the current estimate of the Hessian -see Davidon (1991)-.

2.3 Derivative Free Search: Hooke-Jeeves and Nelder Mead

In the case of non-differentiable functions -or functions without analytical expressions- a different approach to the solution of the problem 1 must be adopted. Derivative-free optimization methods have been developed to allow the optimization of arbitrary functions. Also known as direct search methods, they consist in essence in generating a solution θ and testing its fitness by computing $L(\theta)$.

The Hooke-Jeeves algorithm -see Hooke and Jeeves (1961)- is based on a single base point θ_0 and a step size h. The main idea of the algorithm is to explore the neighbourhood of θ_0 along each of the axes of the search space, and to find whether a step of size h towards the positive or the negative direction is leading to a better fitness. If no improvement has been found after exploring both directions, the original position of θ_0 on that axis is retained. Once every axis has been probed, the new point θ_1, obtained by offsetting θ_0 by h or $-h$ along the relevant axes, is evaluated. If the fitness of θ_1 is no better than the one of θ_0, a new exploration of the neighbourhood of θ_0 is undertaken, this time with a smaller step size. Otherwise, a new base point θ_2 is chosen by taking one step further from θ_1 in the direction defined by θ_0 and θ_1 (formally $\theta_2 = \theta_1 + (\theta_1 - \theta_0)$), optimistically assuming that the direction is leading towards a better fitness, and the algorithm is applied again on θ_2. While the Hooke-Jeeves algorithm relies on a single point and the systematic exploration of its neighbourhood, the Nelder-Mead algorithm, as described in Nelder and Mead (1965), makes use of a set of $n+1$ points in Θ, $\theta_0, \ldots, \theta_n$ forming an $n+1$-dimensional polyhedron, or *simplex*. At each iteration of the algorithm, the indices points are sorted by their increasing fitness so that θ_0 has the best fitness and θ_n presents the worst fitness. The procedure then consists in constructing a candidate replacement point θ_r for θ_n by reflection of θ_n in respect with the center θ_m of the other $\theta_0, \ldots, \theta_{n-1}$ points. Depending on the performance of θ_r compared to θ_0 and θ_{n-1}, an extension point may be created in an optimistic attempt to explore further in the same direction, or on the contrary a contraction point may be computed closer to θ_m. If none of the above attempts lead to a better solution, the simplex is contracted around its best point in order to reduce the exploration range in the next iteration of the algorithm.

It is worth noting that even though these algorithms do not require any knowledge about the fitness function and particularly its derivative, they still do make use of some crude form of gradient by sampling the search space and measuring the difference of the fitness between two points from this sample. If this "gradient" is leaning toward an improvement, the algorithms will make optimistic attempts to follow it in the hope to find yet a better point in that direction.

2.4 Computational Intelligence Optimization

In multi-point methods, the search space is sampled in multiple points, which are considered concurrently. One can consider two classes of metaheuristics, employing multiple starting points and imitating natural processes. Evolutionary algorithms thus considers the multiple starting points as a population of individuals which breed with each other and adapt themselves to their environment. Multiple solutions thus support each other, in the sense that new solutions are derived from several precursor solutions. In swarm intelligence algorithms, the starting points are considered as members of a flock or a swarm, each individual having only a limited intelligence and following simple behavioural rules, but contributing altogether to the solution of the problem. Multiple solutions are then rather following the lead of one of them.

2.4.1 Introduction to Evolutionary Algorithms

The role of evolution in biology and life sciences is well known. Essentially, evolution acts as a type of natural optimization process based on the conceptually simple operations of competition, reproduction, and mutation. The term *Evolutionary Computation* (EC) refers to a class of stochastic search and optimization methods based on the mathematical emulation of natural evolution. That is, EAs mimic the process of natural evolution, taking into account the results of the interplay between the creation of new genetic information and its evaluation and selection. A key point of this class of algorithms is its *population based* structure. In general, a set of *individuals* is continuously evolved and evaluated through the algorithm process. Over the course of evolution, the stochastic nature of reproduction leads to a permanent production of novel genetic information and therefore to the creation of new differing candidate solution (an offspring). In general, any iterative, population based approach that uses selection and random variation to generate new solutions can be regarded as an EA -see Neri et al. (2011)-.

In the context of evolutionary computation the formally equivalent *loss function* (4) is called *fitness function*. It must be noted that a significant group of researchers in the wider CI community considers the nomenclature “loss function” to be used in minimization context while “fitness function” must be used in the case of maximization. However it is straightforward to switch from a minimization problem to a maximization one just by considering the optimization of a dual $-L(\theta)$ or $\frac{1}{L(\theta}$ problem, as discussed in Spall (2003). The use of the term *fitness* is due to the biological metaphor used in evolutionary computation, where each solution is considered an *individual* that tries to survive through its capacity to adapt -see Holland (1975)-.

Each individual within the population is assigned a fitness value $L(\theta)$ as described in (4), which expresses how good the solution is at solving the problem. The fitness value probabilistically determines how successful the individual will be at propagating its genes (its code) to subsequent generations. Better solutions are assigned higher values of fitness than worse performing solutions.

Evolution is performed using a set of stochastic operators, which manipulate the candidate solutions. Most evolutionary algorithms include operators that select individuals for reproduction, produce new individuals based on those selected, and determine the composition of the population at the subsequent generation. Recombination and mutation are two well-known operators. All this is illustrated in Figure 1.

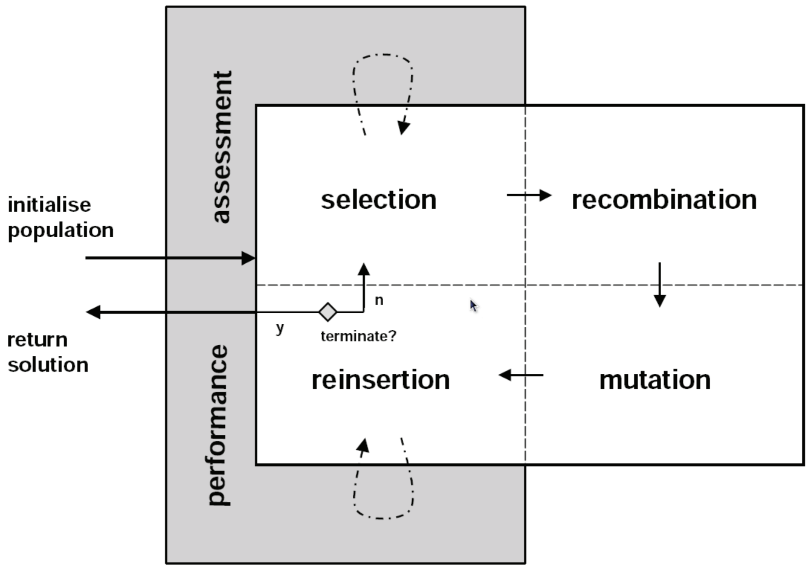


FIGURE 1 Schematic of a general evolutionary algorithm

The recombination operator involves the exchange of information between solutions, in order to create new solutions. The mutation operator makes small, random, changes to a chromosome. Historically considered as a background operator by the genetic algorithm community, its role is often regarded as providing a guarantee that the probability of searching any given string will never be zero and providing an opportunity to recover good genetic material that may be lost through the action of selection and recombination. Once the new generation has been constructed, the processes that result in the subsequent generation of the population are begun once more. The evolutionary algorithm explores and exploits the search space to find good solutions to the problem. It is possible for an EA to support several dissimilar, but equally good, solutions to a problem, due to its use of a population.

EAs are robust tools, able to cope with discontinuities and noise in the problem landscape. They have proved useful at tackling problems that cannot be solved using conventional means. Inclusion of domain-specific heuristics is not a prerequisite, although it may improve the performance of an EA.

An evolutionary algorithm seeks to maximize (or minimize) the mean fitness of its population through the iterative application of the genetic operators previously described. The fitness value of a solution in the EA domain corresponds to a cost value in the problem domain. An explicit mapping is made

```
t=0
initialize P(t)
evaluate P(t)
while budget condition do
    P'(t)=variation P(t)
    evaluate P'(t)
    P(t+1)=selection P'(t)
    t=t+1
end while
```

FIGURE 2 EA pseudo-code

between the two domains. 'Cost' is a term commonly associated with traditional optimization problems and is equally familiar to control engineers through use of such optimization-based design procedures as the linear-quadratic regulator. It represents a measure of performance: namely, the lower the cost, the better the performance. Optimizers seek to minimize cost. Hence, it is evident that by minimizing fitness, the EA is effectively minimizing the cost. Raw performance measures must be translated to a cost value. This process is usually straightforward for single-objective problems, but becomes more complicated in the multi objective case. Every possible decision vector has an associated cost value and fitness value. The enumeration of all such vectors leads to the cost landscape and fitness landscape for the problem.

A general framework for an EA (see Michalewicz (1992)) is shown in Figure 2, where P(t) denotes a population of n individuals at generation t.

At least three variants of evolutionary algorithms have to be distinguished: Genetic Algorithms (GA), Evolution Strategy (ES), and Evolutionary Programming (EP) -see Michalewicz (1992)-. The main differences lie in:

- the representation of individuals;
- the design of the variation operators (mutation and recombination);
- the selection/reproduction mechanism.

In general, in real-world applications the set Θ is the space of the physical parameters of the system (i.e. the parameters of the controller in a control loop) and constitute the so-called *phenotype space*. On the other hand the genetic operators often work in an abstract mathematical space known as the *genotype space*. Two different approaches can be followed: the first is to choose a standard algorithm and a decoding function according to the requirements of the algorithm. The second one is to design the representation as close as possible to the characteristics of the phenotype space. For example, as discussed in Cupertino et al. (2004), the parameters of a controller can be either represented as a binary string -using a decoding function to obtain the right phenotype representation- or coding directly the controller parameters in a real valued genetic algorithm. In the next sections each of these approaches will be reviewed and explained in detail.

```
t=0
initialize P(t)
evaluate P(t)
while budget condition do
    P'(t)=crossover P(t)
    P''(t)=mutation P'(t)
    evaluate P''(t)
    P(t+1)=selection P''(t)
    t=t+1
end while
```

FIGURE 3 GA pseudo-code

2.4.2 Genetic Algorithms

Genetic Algorithms (GA) -see Goldberg (1989) and Michalewicz (1992)- are the most used and best known Evolution Algorithms. The metaphor underlying genetic algorithms is that of natural evolution. In evolution, the problem that each species face is the one of searching for beneficial adaptations to a complicated and changing environment. The "knowledge" that each species has gained is embodied in the makeup of the chromosomes of its members. The general framework shown in the previous Section 2 is still valid, considering the introduction of the two main GA operators: *crossover* and *mutation*. With this modification the evolution process of a GA become as described in figure 3.

That is, a GA is a stochastic algorithm that processes at each step a population of n individuals $\mathbb{P}^k = \{\theta_1^k, \dots, \theta_n^k\}$. In the traditional form proposed by Holland (1975) the mathematical representation for each individual (the genotype) is in the form displayed in equation 2.4.2.

$$f : \{0,1\}^l \to Y \subseteq \mathbb{R}.$$

In case of continuous parameter optimization problem, GA typically represent a real valued vector $\theta \in \Theta$ as a binary string $\mathbf{x} \in \{0,1\}^l$ through the implementation of encoding and decoding functions $h : M \to \{0,1\}^l$ and $h' : \{0,1\}^l \to M$ that facilitate the mapping operation. The genetic operators used in a common GA are:

- **Selection** The Selection operator is based solely on the loss (fitness) values of the individuals. It is in general implemented as a probabilistic operator, using the relative loss value $\frac{L(\theta_k)}{\sum_j L(\theta_j)}$ $(1 - \frac{L(\theta_k)}{\sum_j L(\theta_j)}$ when minimizing) to determine the selection probability of an individual θ_k.
- **Crossover** The standard GA (sGA) performs a so-called one-point crossover, where two individuals are chosen randomly from the population, a position in the bit string (or in the real-valued vector θ_k) is randomly determined as the crossover point and an offspring is generated by concatenating the left substring of one parent with the right substring of the other one. In some applications where real code representation for Θ is chosen its is possible to find real versions of the crossover operator, such as arithmetic crossover

```
t=0
initialize P(t)
evaluate P(t)
while budget condition do
    P'(t)=mutation P(t)
    evaluate P'(t)
    if ($\mu + \lambda$)-ES then
        P(t+1)=selection (P(t) U P'(t))
    else
        P(t+1)=selection P'(t);
    end if
    P(t+1)=selection P''(t)
    t=t+1
end while
```

FIGURE 4 ES pseudo-code

(a linear combination of the vector couple (θ_i, θ_j)). Sometimes the crossover operator can be domain specific, in order to preserve the consistency of the generated solutions Michalewicz and Schoenauer (1996). As an example the partially mapped crossover can be taken in consideration Goldberg and Lingle (1985). It passes ordering and value information from the parent tours to the offspring tours. A portion of one parent's string is mapped onto a portion of the other parents string and the remaining information is exchanged.

- **Mutation** In a genetic algorithm the mutation operator has a small importance Holland (1975). This operator works by inverting a bit in the chromosome bit string (or by randomly altering a real *gene* $\theta_k(i)$) with a very small probability such as $p_m \in [0.005, 0.05]$.

2.4.3 Evolution Strategies

ES was originally designed for constrained continuous variable optimization problems. Like GAs, ES move a population of candidate solutions from generation-to-generation with the aim of converging to a global minimum θ^*. The two general forms of ES in most widespread use, as described in Schwefel and Rudolph (1995) and Bäck et al. (Apr, 1997), are referred to by the notation $(\mu + \lambda)$-ES and (μ, λ)-ES; these are referred to the selection methods of the ES and will be discussed later. ES do not need an encoding function h as for the binary-GA because they work directly with $\theta \in \Theta$. The core steps for ES are reported below in figure 4.

The main differences between $(\mu + \lambda)$-ES and (μ, λ)-ES are in the selection operator: at each generation a new $P'(t)$ subpopulation of λ individuals is created. In the $(\mu + \lambda)$-ES the new population $P(t + 1)$ of N individuals is selected from the combination of the original μ individuals in $P(t)$ plus the new λ offspring; on the contrary, in (μ, λ)-ES, the N best values are selected from the population $P'(t)$ of $\lambda > N$ offspring only.

In this framework the main "variation" operator at each generation is the mutation. New populations are generated from parent θ values according to the formula 9,

$$\theta_{child} = \theta_{parent} + \mu(0, D_{parent}), \tag{9}$$

where $N(0, \sigma)$ is an n-dimensional normal random vector with a diagonal covariance matrix σ; this matrix is sometimes -see Schwefel and Rudolph (1995)- called *step size*. In ES various adaptation mechanism have been implemented to control the step size during the algorithm evolution. More formally, an individual $\mathbf{a} = (\theta, \boldsymbol{œ})$ consist of object variables $\theta \in \Theta$ -the candidate solution- and strategy parameters $\boldsymbol{œ} \in \mathbb{R}_+$. The mutation operator works, as shown in equation (9), by adding a normal random vector $\mathbf{z} \in \mathbb{R}$ with $z_i \sim N(0, \sigma_i)$. The effect of the mutation, thus, is shown in equation 11,

$$\begin{aligned} \sigma_i' &= \sigma_i e^{\tau' N(0,1) + \tau N_i(0,1)} \qquad (10) \\ x_i' &= x_i + \sigma_i' N_i(0,1) \qquad (11) \end{aligned}$$

where $\tau' \propto \frac{1}{\sqrt{2n}}$ and $\tau \propto \frac{1}{\sqrt{2\sqrt{n}}}$. The adaptation algorithm described in (11) is known as *self-adaptation* (Schwefel (1981)).

2.4.4 Differential Evolution

Differential evolution Storn and Price (1995) is a versatile optimizer which, although being originally described as an evolutionary algorithm, shares in certain circumstances some features with Swarm Intelligence algorithms. The general structure of DE is the same as the one of other evolutionary algorithms, with which it shares in particular the concepts of mutation and crossover, but the Differential Evolution cannot be anymore considered to be inspired by evolutionary processes found in Nature. What sets Differential Evolution apart from e.g., Genetic Algorithms or Evolution Strategies is that while in those the mutation is an operation which produces a random change in an individual, the mutation operator in Differential Evolution takes place before the crossover and produces on one hand a deterministic change, and on the other hand may not involve the individual itself at all. Another noticeable difference is that the crossover operation in the Differential Evolution involves one parent and its provisional offspring rather than two parents as is the case in the above-mentioned evolutionary algorithms. The general structure of the Differential evolution is illustrated in Figure 5.

The initial sampling of the population is performed pseudo-randomly with a uniform distribution function within the decision space. At each generation, for each individual x_i of the S_{pop}, three individuals x_r, x_s and x_t are pseudo-randomly extracted from the population. According to the DE logic, a provisional offspring x'_{off} is generated by mutation as:

$$x'_{off} = x_r + F \cdot (x_s - x_t) \tag{12}$$

where $F \in [0, 1+]$ is a scale factor which controls the length of the exploration vector $x_s - x_t$ and thus determines how far from point x_t the provisional

```
t=0
initialize P(t)
evaluate P(t)
**while** budget condition **do**
    P'(t)=Mutation P(t)
    P''(t)=Crossover P'(t) and P(t)
    evaluate P''(t)
    P(t+1)=selection P''(t) over P(t)
    t=t+1
**end while**
```

FIGURE 5 DE pseudo-code

offspring should be generated. When the provisional offspring has been generated by mutation, each gene of the individual x'_{off} is exchanged with the corresponding gene of x_i with a uniform probability and the final offspring x_off is generated:

$$x'_{off,j} = \begin{cases} x'_{off,j} & if \quad U(0,1) < CR \quad or \quad j = j_{rand} \\ x_{i,j} & otherwise \end{cases} \tag{13}$$

where $U(0,1)$ is a uniformly distributed random variable in $[0,1]$; j is the index of the gene under examination, j_{rand} is a randomly selected gene -which is always exchanged, in order to prevent cases where no gene from the provisional offspring is exchanged- and CR is the crossover rate used to define the amount of parent's information to pass to the offspring. The resulting offspring x_{off} is evaluated and, according to a one-to-one spawning strategy, it replaces x_i if and only if $f(x_{off}) < f(x_i)$; otherwise no replacement occurs. It must be remarked that although the replacement indexes are saved one by one during the generation, the actual replacements occur all at once at the end of the generation.

2.4.5 Swarm Intelligence and Particle Swarm Optimization

While evolutionary algorithms find their inspiration in the theory of evolution and natural selection, swarm intelligence has its roots in the observation of groups of animals, where some kind of collective intelligence emerges although the animals themselves do not present signs of intelligence but rather following relatively simple rules. The observation of ants has thus led to the development of the Ant Colony Optimization algorithm, while the metaphor of a flock of birds is the foundation for the Particle Swarm Optimization algorithm.

Although the Particle Swarm Optimization -see Eberhart and Kennedy (1995)- is not directly inspired by a natural phenomenon, it is based on the metaphor of a group of particles using their "personal" and "social" experience in order to explore the problem's space and locate optima. A particle represents a pseudo-randomly initialized point x_i in that space, and is associated to a pseudo-randomly initialized velocity v_i. The algorithm iteratively updates the particle's position until a terminating criterion is met. The particle's position x_i is thus updated by applying the formula 14.

$$x'_i = x_i + v_i \tag{14}$$

Each particle additionally keeps a record of which one, of all the positions it has taken, was the most successful. This point x_i^{best} is called *local best*. Moreover, the population of particles tracks which one of the particles' local bests has the best fitness, and this solution x^{best} is named *global best*. The particle's velocity is then updated based on these two points, using the formula

$$v_i' = v_i + \phi_1 U(0,1)(x_i^{best} - x_i) + \phi_2 U(0,1)(x_i^{best} - x_i) \tag{15}$$

where ϕ_1 and ϕ_2 are parameters and $U(0,1)$ represents a uniformly-distributed pseudo-random variable in the interval $[0,1]$. A particle's local best can be considered as the particle's memory, and thus the personal learning experience of the particle due to successful and unsuccessful moves. The global best, being shared by all the particles represents the above-mentioned social experience. The particles therefore decide of their movements along two dimensions, the first one being their own experience, and the second one being the imitation of the population's most successful individual. The utilization of random scale factors allow to maintain diversity in the population and to prevent premature convergence.

2.4.6 Memetic Computing

Memetic Computing is a subject of computational intelligence which studies algorithmic structures composed of multiple interacting and evolving modules (memes), see Neri and Cotta (2011a). The first time that the term memetic has been used in computer science was in late '80s, see Moscato and Norman (1989), when the definition of Memetic Algorithms (MAs) has been coined. In their early definition, MAs were a modification of GAs employing also a local search operator for addressing the Traveling Salesman Problem (TSP) Moscato (1989). While the employment of hybrid algorithms in optimization was already broadly used, a novel and visionary perspective to optimization algorithms in terms of memetic metaphor has been given in Moscato (1989). According to the metaphor, a meme is the unit of cultural transmission, see Dawkins (1976). Human knowledge can be seen as a structure of memes where each meme that can be duplicated in human brains, modified, and combined with other memes in order to generate a new meme.

An operative definition of MA has been given in Hart et al. (2004), where MAs are defined as population-based metaheuristics composed of an evolutionary framework and a set of local search algorithms which are activated within the generation cycle of the external framework.

A plenty of algorithms fitting within this definition have been lately proposed in literature to address a multitude of real-world optimization problems, see Neri and Cotta (2011b) and Neri et al. (2011). A crucially important issue in MA implementation is the selection and coordination of the memes within an algorithmic structure, see Neri et al. (2011). By updating the classification given in Ong et al. (2006), MA structures for automatic coordination of the algorithmic components can be subdivided as: 1) Adaptive Hyper-heuristic, see e.g. Cowling et al. (2000), Kendall et al. (2002), Burke et al. (2003), and Kononova et al.

(2008), where the coordination of the memes is performed by means of heuristic rules; 2) Self-Adaptive and Co-Evolutionary, see e.g. Smith (2007), Yu and Suganthan (2010), and Krasnogor and Smith (2005), where the memes, either directly encoded within the candidate solutions or evolving in parallel to them, take part in the evolution and undergo recombination and selection in order to select the most promising operators; 3) Meta-Lamarckian learning, see e.g. Ong and Keane (2004), Korošec et al. (2011), Nguyen et al. (2009), and Le et al. (2009), where the success of the memes biases their activation probability, thus performing an on-line algorithmic design which can flexibly adapt to various optimization problems; 4) Diversity-Adaptive, see e.g. Caponio et al. (2007), Neri et al. (2007), Neri et al. (2007b), Neri et al. (2007a), Tirronen et al. (2008), Caponio et al. (2009), and Tang et al. (2007), where a measure of the diversity is used to select and activate the most appropriate memes. In addition, it is worthwhile commenting Baldwinian systems, i.e. those MAs which do not modify the solutions after the employment of local search, see Yuan et al. (2010) and Gong et al. (2010).

Recently, two concurrent reasons led to an extension of MA definition towards the broader concept of Memetic Computing. The first reason is several hybrid implementations which would nicely fit the memetic metaphor were technically not MAs according to the definition above. The second reason is that part of the community started thinking about the automatic generation of algorithms and brought to a more abstract level the memetic metaphor. For example, an early attempt of mentioning a system capable to automatically generate MAs for given problems is given in Meuth et al. (2009). In this context the term Memetic Computing (MC) has been coined and defined as "...a paradigm that uses the notion of meme(s) as units of information encoded in computational representations for the purpose of problem solving", see Ong et al. (2010).

A reorganization of the subject have been reported in Neri and Cotta (2011b) and Neri et al. (2011) where the definition of MC is reported:

Memetic Computing is a broad subject which studies complex and dynamic computing structures composed of interacting modules (memes) whose evolution dynamics is inspired by the diffusion of ideas. Memes are simple strategies whose harmonic coordination allows the solution of various problems.

3 WHY ARE REAL-WORLD APPLICATIONS MORE CHALLENGING THAN "TOY PROBLEMS"?

3.1 Control Engineering: Generalities

3.1.1 Suitability

The evolutionary approach has proved particularly successful in problems that are difficult to formalize mathematically, and which are therefore not conducive to analysis. This includes systems that are highly nonlinear, that are stochastic, and that are poorly understood, e.g. as discussed in Fleming (2002). Problems involving these classes of process tend to be difficult to solve satisfactorily using conventional methods. The EA's lack of reliance on domain-specific heuristics makes it attractive for application in this area. Very little a priori information is required, but this can be incorporated if so desired. A single control engineering problem can contain a mixture of decision variable formats. This can prove significantly problematic for conventional optimizers that require variables of a single mathematical type or scientific unit. Since the EA operates on an encoding of the parameter set, diverse types of variable can be represented (and subsequently manipulated in an appropriate manner) within a single solution. For example, the decision vector {sensor type A, 18.3 degrees, blue, 2π} does not pose an intrinsic problem to the EA. The EA is a robust search and optimization method that is well able to cope with ill-behaved cost landscapes, exhibiting such properties as multi-modality, discontinuity, time-variance, randomness, and noise. Each of these properties can cause severe difficulties to traditional computational search methods, in addition to the lack of amenity to an analytical solution. Furthermore, an EA search is directed and, hence, represents potentially much greater efficiency than a totally random or enumerative search.

3.1.2 Unsuitability

Safety critical applications do not appear, initially, to be favourable towards EA usage due to the stochastic nature of the evolutionary algorithm. No guarantee is provided that the results will be of sufficient quality for use online. When EAs are evaluated on benchmark problems, they are commonly tested many (typically 20-30) times due to the algorithms' stochastic nature. There is also the matter of how individuals will be evaluated if no process model is available (as may well be the case). Some supporting theory exists for evolutionary algorithms, especially for ES, but this is unlikely to prove sufficient to win the approval of standards and certification agencies. Much care would clearly be needed for critical systems. Real-time performance is of particular interest to the engineer. However, EAs are very computationally intensive, often requiring massively parallel implementations in order to produce results within an acceptable time-frame. Hence, online application to real-time control is often infeasible. Processes with long time-constants represent the most feasible application.

3.1.3 Online optimization framework

Online applications present a particular challenge to the optimization. Successful applications in this field have been somewhat limited to date. The benefits of an EA for online control systems engineering applications are the same as those discussed for off-line applications. However, an online EA approach must be used with particular caution. There are several considerations to be made. It is important that an appropriate control signal is provided at each sample instant. If unconstrained, the actions of the 'best' current individual of the EA may inflict severe consequences on the process. This is unacceptable in most applications, especially in the case of a safety- or mission-critical system. Given that it may not be possible to apply the values represented by any individual in an EA population to the system, it is clear that evaluation of the complete, evolving, population cannot be performed on the actual process. The population may be evaluated using a process model, assuming that such a model exists, or performance may be inferred from a system response to actual input signals. Inference may also be used as a mechanism for reducing processing requirements by making a number of full evaluations and then computing estimates for the remainder of the population based on these results.

In a real-time application there is a limited amount of time for which an optimizer can be executed between decision points. Given current computing power, it is unlikely that an EA will execute to convergence within the sampling time limit of a typical control application. Hence, only a certain number of generations may be evolved. For systems with long sample times, an acceptable level of convergence may well be achieved.

In the case of a controller, an acceptable control signal must be provided at each control-point. If the EA has evolved for only a few generations then population performance may still be poor. A further complication is that the system, seen from the perspective of the optimizer, is changing over time. Thus, the evolved control signal at one instant can become totally inappropriate at the next. EAs can cope with time varying landscapes to a certain extent, but a fresh run of the algorithm may be required. In this instance, the initial population can be seeded with previous 'good' solutions. Note that this does not guarantee fast convergence and may even lead to premature convergence.

In literature, two main frameworks for the online controller optimization can be identified.

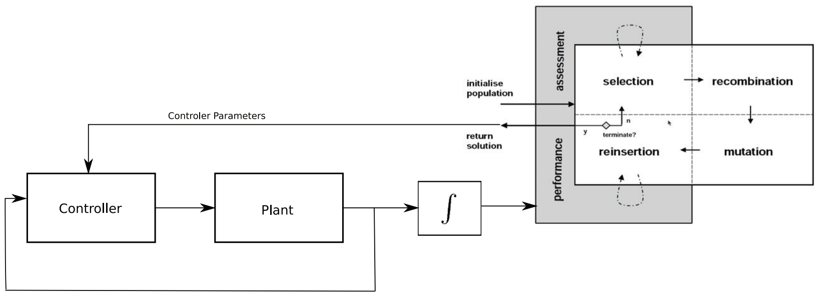


FIGURE 6 Direct optimization framework

When the stochastic nature of the search for a good solution does not compromise the overall quality of the process functioning, then the framework shown in Figure 6 can be used. In this configuration, the optimization algorithm works online with the control system. A cost function is computed over a proper time window with a specific time integral criterion (e.g. IAE, integral absolute error). The width of the time window must be chosen according to the system main time constant. The convergence to a good global solution, or vice versa, to a good performance of the control system can take some time.

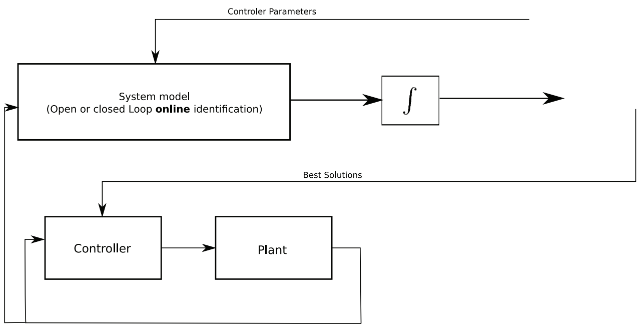


FIGURE 7 Indirect optimization framework

For the so called "mission critical" systems, such as chemical plants, a second framework is considered more suitable. The indirect optimization framework, shown in 7, uses an online system identification procedure in order to properly simulate the system responses. This model is identified offline in a first stage, then it is used to optimize the control parameters. Only the best solution (i.e. at each generation or at the end of the optimization) is actually used on the real system. Sometimes a proper feedback from the real system is used to train again (online training) the system model. This framework is not very common in literature, also due to the intrinsic difficulties in obtaining a good model of the system.

3.1.4 EA for the online tuning of controls

A typical task of a EA in this context is to find the best values of a predefined set of free parameters associated to either a process model or a control law. The general problem of evolutionary control system design has been tackled in various ways, which can be broadly classified from three different viewpoints, as summarized in Table 1. Namely, the main characteristics of the various approaches presented in literature reside in the structure of the control law, in the formulation of the control objective, and in the mechanism of computing the fitness. The first aspect involves the definition of a parametrized control law, i.e., the selection of controller parameters that have to be optimized by the GA. From this viewpoint, the largest part of the literature focuses on the optimization of linear PID controllers. Various case studies finally show that GAs lead to more effective PID regulators than those obtainable with more conventional design approaches. With similar motivations, some researchers also focused on problems with larger search spaces, e.g., combination of PIDs with lead compensators, see Grum et al. (2001), or other linear transfer function -as in Boussak and Capolino (1992)-, obtaining analogous results, as discussed in Chipperfield and Fleming (1996).

GAs have also been used to optimize nonlinear control strategies. In particular, a large amount of research focused on the design of fuzzy controllers (FCs) using evolutionary approaches -see Hoffmann (2001) for a survey-. In fact, although in principle FCs can be programmed using intuitive or expert knowledge about the controlled process in the form of linguistic rules, a fine tuning procedure is often necessary to reach satisfactory results. In the case of FCs, the manual tuning is even harder due to the non-linearity of most operations performed by such controllers, and the consequent lack of knowledge about the effects of each configuration parameter on the input-output law. Furthermore, fuzzy inference algorithms are generally difficult to implement on commercial low-cost microcontroller, and must often be converted in look-up tables with additional memory requirements. It must also be remarked that, even if fuzzy controllers are universal approximators that can reproduce any input-output law, they may not immediately offer significant improvements in terms of robustness and performance. In fact, the use of classical PID controllers is still preferred in industrial contexts unless nonlinear approaches are strictly required.

A second classification criterion of the existing literature concerns the formulation of the fitness, i.e., the objective function describing the control goals. As shown in Table 1, most technical literature defines the fitness with single indices, as integral time errors in system's step response. Control design problems are inherently multi-objective, and often a single time-response-based index is not

TABLE 1 Classification of literature on EAs for control

Controller Type	PI/PID	Duivenbode et al. (1998), Krohling and Rey (2001), Chen et al. (2001), Acarnley et al. (2000), Porter and Jones (1992)
	Fixed structure - Linear Transfer Function (or Matrix) w/wo anti-windup	Fonseca and Fleming (1998), Chen and Cheng (1998), Lee et al. (1998), Grum et al. (2001)
	Optimizable Structure - Combination of linear (lead, lag, integrator) with anti windup devices	Hoffmann (2001)
Fitness	Integral of the error indices (e.g. IAE, ITAE)	Duivenbode et al. (1998), Krohling and Rey (2001), Acarnley et al. (2000), Porter and Jones (1992)
	$\mathbf{H}_\infty$ norm (stability, disturbance attenuation)	Krohling and Rey (2001), Chen and Cheng (1998), Grum et al. (2001)
	Multi-objective (aggregated): combination of time and frequency indices, control action, disturbance rejection	Bobbin and Yao (1997), Salvatore et al. (2002), Lee et al. (1998), Yu et al. (1999)
	Multi-objective (Pareto-optimal): combination of time and frequency indices, control action, disturbance rejection	Chipperfield and Fleming (1996), Fonseca and Fleming (1998), Grum et al. (2001), Chen et al. (2001)
	Other	Zinober et al. (1995), Marrison and Stengel (1997), Leng et al. (2000)
Computing the fitness	Analytical formulae	Krohling and Rey (2001), Chen and Cheng (1998)
	Simulation	Bobbin and Yao (1997), Chipperfield and Fleming (1996), Salvatore et al. (2002), Porter and Jones (1992), Duivenbode et al. (1998)
	Direct experiments on process	Lee et al. (1998), Grum et al. (2001), Acarnley et al. (2000)

sufficient to describe the desired specification of the controller. Furthermore, the fitness can incorporate any measurable, observable, or calculable behaviour, or characteristic of the considered problem -see Yu et al. (1999)-, so the choice of integral time errors seems not to fully exploit the potentialities of the GAs. In fact, a fitness function can simultaneously take into account several aspects (e.g., in time and frequency domain) of the design problem. The objectives can be either combined by aggregating different indices in a unique fitness -as discussed in Salvatore et al. (2002)- or considered separately in a multi-objective optimization problem -see Chipperfield and Fleming (1996)-. The latter approach, however, requires a more complex implementation of the GA, a larger number of iterations and human supervision to guide the search toward the desired trade-off of objectives. The third main classification criterion is related to the method to compute the fitness of each controller. Namely, this can be achieved through closed formulae, through computer simulations, or by direct experimentation on the process. Closed formulae are generally available only using linear models for the process and the controller, and neglecting all the nonlinear effects, such as hysteresis, voltage drops across brushes in electric drives -see Marrison and Stengel (1997)-. In such cases, using simple linear controllers, the integral-time-based performance indices (e.g., IAE or ITAE) can be computed through closed formulae of controller parameters. More frequently, nonlinear effects are not negligible but an accurate simulation model of the controlled process is available. In such cases, a straightforward way to compute the fitness of the controller is to perform closed loop simulation. Generally, GAs require order of thousand fitness evaluations to converge properly. Therefore, each simulation involved in fitness computation should take a short enough time to make this approach feasible. When the fitness is measured by direct experiments on the physical hardware loop, the considerable advantage of these approaches is the reliability of final results. While all the model-based techniques expressly rely on the accuracy of the model used in the simulations, in this case the effect of the actual and unknown a priori high-order phenomena and nonlinearities are fully accounted in the fitness, and the final controller (the one generating the best fitness) is ready-for-use with known performance. On-line GA design techniques allow us to obtain automatic design tools that do not require skilled expertise for system modelling, or trial-and-error controller optimization. On the other hand, in spite of the simplicity of the basic idea, online genetic optimization requires to deal with several challenging problems, which have strongly limited to date the number of successful applications in this field.

3.1.5 Control of an Electric Drive

The application of the EA approach to the online optimization of a real system, the case of the control of a DC-Drive, as studied by the author in Cupertino et al. (2004) is presented in this section as an example of control application for EAs.

Designing and developing a system capable to properly run an evolutionary search procedure directly commanding the physical hardware can be extremely complex. First of all, safety mechanisms to avoid that poor-performing or even unstable solutions could permanently damage the hardware must be developed. The algorithm must also be designed to avoid interference among subsequent solutions (i.e., it must be ensured that the fitness of an individual does not depend on previously tested solutions, and nor will it affect the performance of the subsequent ones). Furthermore, to obtain significant and practically useful results, thousands of experiments must be conducted to obtain controllers with high fitness, and mechanisms improving the speed of convergence become indispensable to obtain results of practical interest in reasonable search times.

An objective function has been designed that aggregates in a single scalar fitness a considerable number of conflicting performance measures, taking into account the actual operating requirements and conditions of an electric drive, ranging from good reference signal tracking to effective disturbance rejection, from the limitation of actual power supply to the minimization of the effects of uncertainties due to frictions, hysteresis, etc. A further contribution resides in the mechanism used to properly limit the experimentation of badly performing or unstable controllers. In this case, each index contributing to the total fitness is computed on-line (i.e., updated at each time sample of the experiment) and constantly monitored. During the evaluation of an individual, if some of the monitored indices exceed predefined thresholds, the current experiment is immediately stopped without interrupting the evolutionary search, which proceeds to the evaluation of the next individual in the population. According to experimental investigation, this highly practical scheme is sufficient to guarantee that unnecessary stress of the hardware equipment is avoided.

3.1.5.1 Dc Drive

The complete set of equations for a DC motor can be easily found starting from the physical principles behind it, as shown in Figure 8. For the armature circuit it

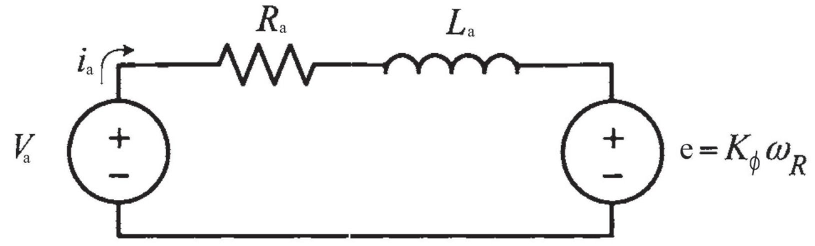


FIGURE 8 Equivalent circuit of the armature electrical dynamics

have,

$$L_a \frac{di_a}{dt} = -R_a i_a - K_\phi \omega_R + V_a \tag{16}$$

where,

- R_a: Armature resistance

- L_a: Armature inductance
- ω_R: Rotor speed
- K_ϕ: EMF constant.
- V_a: Armature voltage
- i_a: Armature current

The torque due to the external magnetic field acting on the current in loop is of the form

$$\tau_e = K_\phi i_a, \tag{17}$$

where K_ϕ is the torque constant. Finally, the mechanical equations are,

$$J\frac{d\omega_R}{dt} = K_\phi i_a - f\omega_R - \tau_L, \tag{18}$$

$$\frac{d\theta_R}{dt} = \omega_R. \tag{19}$$

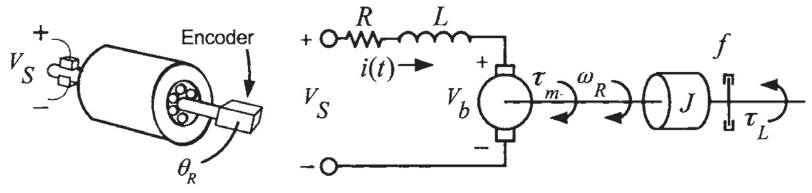


FIGURE 9 DC motor schematic

A picture of a DC motor servo system and its associated schematic is shown in Figure 9. In the schematic, J is the rotor moment of inertia and f is the coefficient of viscous friction. It is possible to substitute in equation (16) $\tau_a = \frac{L_a}{R_a}$ and $\tau_m = \frac{R_a J}{K_\phi^2}$. With these, the s-domain transfer functions of the DC motor become:

$$\omega(s) = \frac{1}{K_\phi}\frac{1}{\tau_a\tau_m s^2 + \tau_m s + 1}V_a(s) \tag{20}$$

3.1.5.2 Design on unstructured controllers with GAs

The design of a discrete control system for electric drives can be formulated as a search problem. As in optimal control design, we aim to find the controller (and its order) achieving the maximum satisfaction of a closed-loop cost-to-performance merit function. The merit function takes into account the transient and steady-state behaviour, the disturbance rejection, and the form and entity of the control action. Ideally, the search algorithm must also be able to recognize and handle unstable or badly performing individuals, possibly prior to their actual experimentation on the real hardware. This section describes the main structure and operators of the search algorithm, the mechanism developed to deal with unstable solutions, and the fitness function.

A preliminary step in GA's configuration is to define the encoding/decoding strategy that associates a generic solution (i.e., the unstructured linear controller) to an individual suitable for genetic operations as crossover and mutation. The definition of the coding strategy can significantly affect the performance of the GA in terms of accuracy and speed of convergence, and is therefore critical in on-line application. Since our aim is to work on populations of controllers of heterogeneous structure, the chromosomes have to include discrete (the order) and continuous information (the position of poles and zeroes, or equivalently the polynomial coefficients). As described in Figure 10 the controller is viewed as

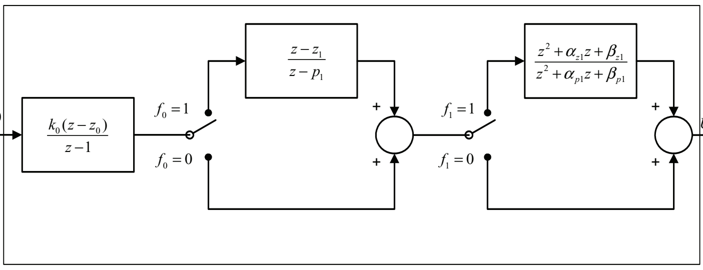


FIGURE 10 Cascade of elementary components of the controller

a cascade of elementary controllers. The schema used contains a PI controller, which guarantees a steady-state constant-reference tracking, a first-order lag or lead compensator, and a second-order compensator with real or complex roots. In this way, the maximum order of the controller is four. A sequence of binary flags (indicated as f_0 and f_1 in Figure 10) associated to each elementary controller constitutes the header of the encoded string, and specifies which components are actually active. The rest of the string contains the sequence of the parameters of each elementary controller (k_0, z_0, z_1, p_1, etc.) which evolve and converge independently from the (de)activation of the corresponding flag. Then, the strings corresponding to the independent controllers are concatenated to form a unique linear transfer function, described in Figure 10 as $G(z) = U_L(z)/E(z)$, where U_L is the nonsaturated control action, E the error, and z is the Z-transform complex variable.

The actual control action $u(k)$ at discrete time k is computed applying an anti-windup algorithm to the linear law $G(z)$. In particular, indicating with n the order of $G(z)$, and defining $N(z)$ and $D(z)$ its numerator and denominator, respectively, the actual control $u(k)$ is computed as

$$u(k) = \begin{cases} u_L(k), & \text{if } |u_L(k)| \leq u_{max} \\ sign\left(u_L(k)\right) u_{max}, & \text{otherwise} \end{cases} \tag{21}$$

with,

$$u_L(k) = \frac{1}{b_0} \left(\sum_{i=0}^{n} a_i - e(k-i) - \sum_{j=1}^{n} b_j u(k-j) \right) \tag{22}$$

where a_i and b_j, $(i, j = 1, \ldots, n)$ are the coefficients of $N(z)$ and $D(z)$, respectively and u_max is the upper bound of the control action. In Figure 11 it is summarized

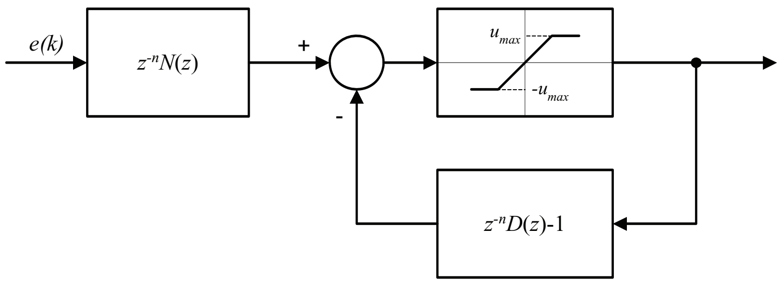
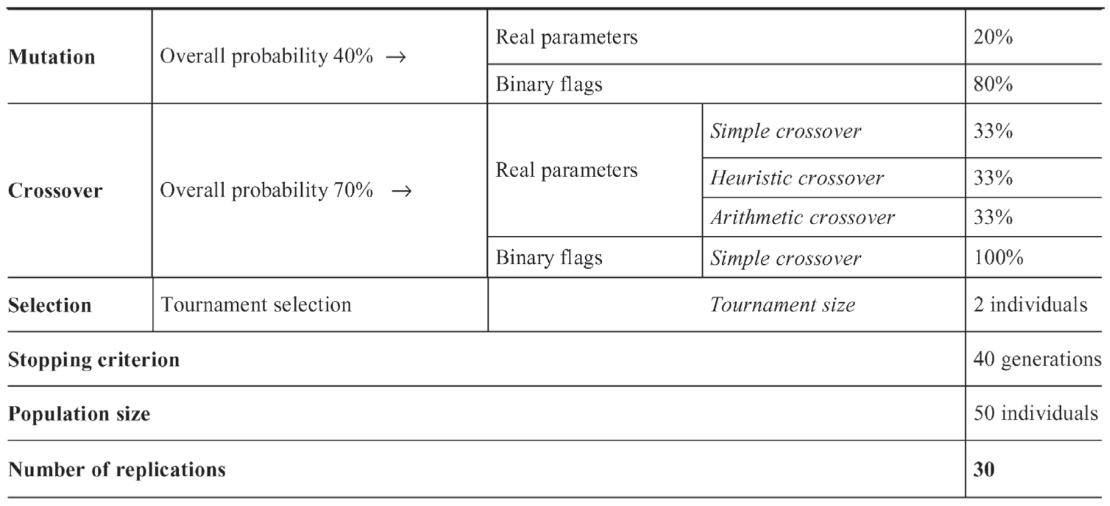


FIGURE 11 Anti-windup algorithm

the anti-windup algorithm with a block diagram.

Specific variants of the conventional mutation and crossover operators have been developed with this hybrid encoding schema. The mutation consists in two different operators that are applied randomly with different probability. The first type of mutation (*binary mutation*) is relative to the binary flags describing the structure, and works as a conventional binary mutation, i.e., inverting the value of the selected flag (adding or removing one component in the cascaded controller in Figure 10). The second type of mutation (*real mutation*) is only applied to real-valued parameters and consists of random additive perturbation of them within the prescribed bounds. Since the effects of the two mutation operators are significantly different, their rate of occurrence must be carefully selected to obtain the desired evolution of the search. The binary mutation acting on structure flags may seem to perform a strong alteration of the mutated solution, since it adds or remove poles or zeros from the mutated individual. Obviously, the entity of the alteration also depends on the position of the removed or added roots with respect to the remaining ones (e.g., adding a dominated pole will not significantly alter the response of the individual). In fact, this makes the binary mutation very similar to most mutation operators (e.g., the single-bit mutation in conventional binary coded GAs), which can occasionally produce strong alteration of the mutated individual (e.g., when the single-bit mutation is applied to the most significant gene). During the preliminary simulation investigation made to properly configure the GA, it has been observed that to obtain a satisfactory speed of convergence, the probability of binary mutation should be significantly lower than the one for real mutation.

The probability of occurrence of the various mutation operators are summarized in Table 12. Similar considerations can be extended to the crossover between solutions, which also consists of various operators applied with different frequencies. Namely, depending on a random condition, the crossover can be applied either to the flags or to the parameters of a chromosome. If the binary part is selected, the algorithm applies a multi-point crossover (i.e., flags are randomly swapped between individuals), otherwise, it applies one of three different real-valued crossovers (the simple, arithmetical, and heuristic crossover operators described in Michalewicz (1992)). Simple crossover works similarly to



Mutation	Overall probability 40% $\rightarrow$	Real parameters		20%
		Binary flags		80%
Crossover	Overall probability 70% $\rightarrow$	Real parameters	*Simple crossover*	33%
			Heuristic crossover	33%
			Arithmetic crossover	33%
		Binary flags	*Simple crossover*	100%
Selection	Tournament selection		*Tournament size*	2 individuals
Stopping criterion				40 generations
Population size				50 individuals
Number of replications				**30**

FIGURE 12 Summary of the GA configuration

the binary one, by randomly swapping the genes of the two parents. The arithmetic crossover generates two new individuals along the direction defined in the search hyper-rectangle by the two parents (i.e., it performs a linear convex combination of the real-valued chromosomes). These operators always produce an offspring that is included in the hyper-rectangle having two extreme vertices in the two parents. If used alone, both these crossover operators lead the populations to quickly concentrate in gradually smaller regions of the solution space, often causing the premature convergence of the GA. To avoid this risk, we introduce the third crossover operator, which works implicitly as a local hill-climbing heuristic. Namely, the heuristic crossover only generates a new individual that lies outside the hyper-rectangle delimited by parents, in the direction of the parent with the better fitness. Finally, this operator returns the best parent as second child. Applying the three operators together, it is possible to achieve the desired compromise between exploration and exploitation for our problem.

As already mentioned, the rate of occurrence of each mutation and crossover operator must be defined with care to obtain an efficient search leading to satisfactory controllers. In particular, to fully exploit the advantages of an evolutionary optimization in a search space containing controllers of different orders, intuitively, the populations should evolve so that, in the first iterations, controllers of different structure coexist and compete for reproduction, while the associated real-valued parameters converge to their optimized values for each different structure. Later on, during the run, the most successful structures (in terms of the specific fitness used) should progressively gain larger portions of the population until the latest iterations, when the population should converge to regulators having the same order, and the GA should perform a fine tuning of the real valued parameters.

A second aspect of fundamental importance in on-line GA applications is the definition of special mechanisms to avoid the experimentation of unstable or badly performing individuals. Namely, due to the inherently random exploration of the GAs, the crossover and mutation may generate unstable closed-loop systems, especially during the first iterations. Testing such controllers is not necessary and often dangerous, since it may cause hardware damages. To overcome this risk, as proposed by some authors (see, Chen and Cheng (1998)), it would be desirable to constrain the search into subregions where the control system is stable or it has predictable behaviour. Unfortunately in the case discussed here such regions are not known *a priori* with sufficient certainty. On the one hand, imposing conservative bounds may exclude regions containing good controllers from the search, this way compromising the optimality of the final results. In Cupertino et al. (2004) a heuristic strategy has been adopted that effectively overcomes the limitations of model-based analysis. Instead of computing the fitness at the end of each experiment, as usually done in on-line tuning approaches, we compute each index contributing to the total fitness on-line, i.e., updating its value at each time sample of the experiment. The on-line value of each fitness term is constantly monitored, and whenever it exceeds a predefined threshold the current experiment is immediately stopped. In fact, in our application, badly performing controllers exhibit an irregular behaviour even in the earliest part of the experiment. It is so possible to detect unstable (or highly unsatisfactory) solutions well before the involved signals reach potentially dangerous values. Furthermore, the effects of frictions combined with a limited supply of control action contribute to limit the effects of badly performing individuals. In case a monitored index exceeds the prescribed threshold, the value of the fitness is multiplied by a penalty factor and assigned to the individual, and the algorithm proceeds with another experiment. In this way, the evolutionary search is never interrupted until the terminating condition occurs.

The reference input for the control loop also plays a fundamental role in the success of the evolutionary design. It is not so difficult to tune a good PI controller looking at motor speed during startup from zero to full speed, trying to minimize the IAE of the speed response. This task can be performed by skilled operator or using a simple search algorithm. Unfortunately, the obtained controller does not give satisfactory responses in the low speed range or when the load condition changes. Controllers for electric drives must provide satisfactory responses when operating within the linearity ranges of the actuators, and at the same time they must limit the effects of saturations in presence of abrupt set-point changes. These aspects are often contrasting, i.e., good controllers in full-load conditions exhibit poor performance when no load is applied to the motor and *vice versa*. In order to take all these aspects into account in our problems, the system is excited with a speed reference of variable amplitude: first small and then high reference changes let the controllers operate both in linear and saturated conditions. At the same time, a suitable sequence of load steps is applied so to explore the motor drive performance in the whole speed range at different load levels, as shown in Figure 13.

To optimize the overall response of the motor drive, the employed objective function is a weighted sum of several performance indices that are directly mea-

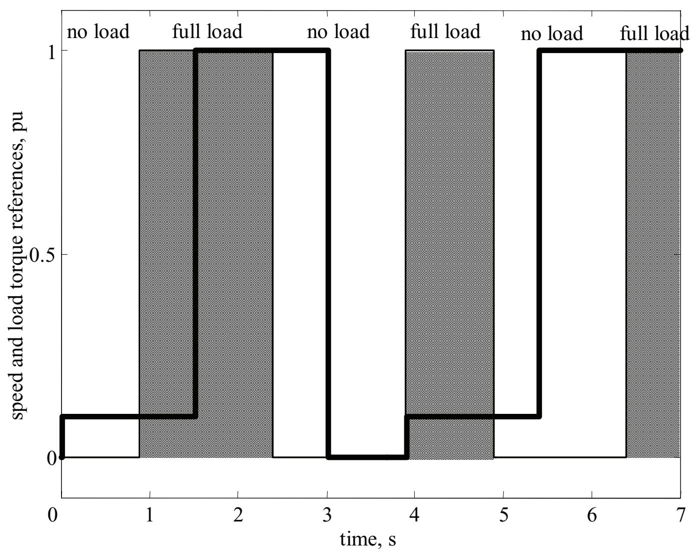


FIGURE 13 Speed and load references during the training test

sured during the system response to the input signal described above. The fitness to be minimized is:

$$f = \sum_{j=1}^{5} \alpha_j f_j, \tag{23}$$

where α_j represent positive weights and f_j are five performance indices defined in the following. The formulation of a single aggregated index was initially preferred to "truly" multiobjective GAs (MOGAs) searching for the Pareto-front of nondominated solution (see, Fonseca and Fleming (1998)), because the latter approaches in general require a larger number of fitness evaluation to converge. The indices used in the optimization problem are the following:

- *Steady-State Speed Response:* This index measures the speed error in the settling phases

$$f_1 = \sum_{j=1}^{n} |\omega_R(j) - \omega_R^*(j)| g_\omega(j), \tag{24}$$

  where ω_R and ω_R^* indicate motor speed and speed reference (i.e., setpoint), respectively, and the integer n indicates the number of stored signal samples. When the speed reference changes, g_ω is set equal to zero, and after 0.25 s is set equal to one. In this way, the speed error is not considered for the calculation of f_1 when the speed controller saturates. In fact, in this condition, the speed behaviour cannot be further improved because the motor acceleration is limited by the actual motor current.

- *Speed Overshoot:* We define the single overshoot index as

$$f_{os} = \frac{\omega_R^{max} - \omega_R^{steady}}{\omega_R^{steady}} \tag{25}$$

where $\omega_R^{max} = \max_{h=1,\ldots,n} |\omega_R(j)|$, and ω_R^{steady} is the steady-state speed value. The global overshoot index f_2 is the sum of the indices f_{os} measured for every speed step change.

- *Transient Response Duration:* The index f_3 is the sum of the settling times (within 2% of the reference) measured for every speed step change.
- *Rise Time:* The rise time from 2% to 98% of the reference speed step is measured every time the speed set point is modified. The fourth index f_4 measures the sum of all the rising times.
- *Current Reference Oscillations for Constant Speed Reference:* For each step change the following index is measured:

$$f_{ref} = \sum_{j=n_i}^{n_{i+1}-1} |i_a^*(j) - i_a^{mean}| g_\omega(j) \tag{26}$$

  where n_i is the number of the sample corresponding to a speed step change, i_a^{mean} is the mean value of i_a^* calculated between n_i and $n_{i+1} - 1$ when $g_\omega(j) = 1$. The index f_5 is the sum of the f_{ref} indices and accounts for undesired ripples and oscillations of the current that increase losses and vibrations.

3.2 Noisy Fitness Functions

3.2.1 Definitions

As already described in the previous sections, the presence in the control loop of sensors,measurement devices, numerical simulators, and other objects which perform measurements or approximations within the calculation of the objective function. If the noise is unavoidable and cannot be eliminated from the objective function computation, the optimization algorithm should take into account this difficulty of the problem and perform its action notwithstanding the presence of noise.

For example, simply consider a generic process state measurement result

$$Y_i = X_i + \epsilon_i, i = 1 \ldots N. \tag{27}$$

In this situation the "real" process state X is overlapped by the noise ϵ. The loss function L can be then computed as

$$\min_{\theta \in \mathbb{R}^n} L(\theta) = \min_{\theta \in \mathbb{R}^n} \sum_i \|\theta - Y_i\| . \tag{28}$$

In this case, the values of $L(\theta)$ are affected by the noise signal ϵ and must be taken into consideration for the choice of the optimization algorithm. The optimization algorithm should be able to handle the noise and perform a compensation and/or filtering during the optimization process. As summarized in Di Pietro et al. (2004), the noise in the objective function causes two types of undesirable behavior: 1) a candidate solution may be underestimated and thus eliminated, 2) a candidate solution may be overestimated, thus saved and allowed to lead towards incorrect search directions. Equivalently, a noise fitness landscape can be seen as characterized by false optima which consequently mislead the algorithm search, see Neri and Mäkinen (2007). For these reasons, the application of a classical (deterministic) optimization algorithm is often inadequate, since its search strategy would be heavily jeopardized by the noise. Some examples of deterministic algorithms whose structure has been modified in order to handle noise are given in Anderson et al. (2000) and Neri et al. (2008). On the other hand, due to their inner structure and population based setup, Evolutionary Algorithms (EAs) are considered to be very promising with noisy problems and manage to improve upon the initial fitness values, see Arnold and Beyer (2003), Beyer and Sendhoff (2006), and Arnold and Beyer (2006). However, the noise still represents a challenge to be handled, and standard EAs often do not manage to detect satisfactory solutions. In particular, the most critical operation is, as highlighted in Branke and Schmidt (2003), selection of the most promising solutions for subsequent phases of the optimization algorithms. Obviously, the selection requires a prior fitness-based comparison operation whose reliability and significance can be heavily jeopardized by the noise.

3.2.2 A Brief Survey on Algorithms for Noisy Optimization

In a noisy fitness scenario the application of a classical (deterministic) optimization algorithm is often inadequate, since its search strategy would be heavily jeopardized by the noise. Some examples of deterministic algorithms whose structure has been modified in order to handle noise are given in Anderson et al. (2000) and Neri et al. (2008).

On the other hand, due to their inner structure and population based setup, Evolutionary Algorithms (EAs) are considered to be very promising with noisy problems and manage to improve upon the initial fitness values, see Arnold and Beyer (2003), Beyer and Sendhoff (2006), and Arnold and Beyer (2006). However, the noise still represents a challenge to be handled, and standard EAs often do not manage to detect satisfactory solutions. In particular, the most critical operation is, as highlighted in Branke and Schmidt (2003), selection of the most promising solutions for subsequent phases of the optimization algorithms. Obviously, the selection requires a prior fitness-based comparison operation whose reliability and significance can be heavily jeopardized by the noise.

Following the analysis reported in Jin and Branke (2005), noise handling components for EAs can be classified into two categories, each category being divided into two sub-categories:

- Methods which require an increase in the computational cost

  Explicit Averaging Methods

Implicit Averaging Methods

- Methods which perform hypotheses about the noise

  Averaging by means of approximated models

  Modification of the Selection Schemes

Explicit averaging methods consider that, in the presence of noise, re-sampling of the fitness values followed by the averaging (for zero-mean noise) of these values is beneficial in order to perform a correct fitness estimation. As a matter of fact, increasing the sample size is equivalent to reducing the variance of the estimated fitness. Thus, ideally, an infinite sample size would reduce to zero uncertainties in the fitness estimations, transforming the problem into a non-noisy one.

Implicit averaging consists of enlarging the population size in order to give the solutions a chance to be re-evaluated. In addition, a large population size allows the evaluations of neighbour solutions and thus an estimation of the fitness landscape in a certain portion of decision space. In Miller and Goldberg (1996), the fact that a Genetic Algorithm (GA) with infinite population size would be noise-insensitive has been proven.

The topic, whether a re-sampling or an enlargement of the population size is better when it comes to noise handling, has been discussed in literature and various results supporting both philosophies have been presented, e.g., Beyer (1993), and Hammel and Bäck (1994).

In both cases, these methods lead to an increase in the amount of fitness evaluations with respect to a standard EA. This fact obviously implies an increase in computational overhead, which can be unacceptable in real-world applications where the computational cost of each fitness evaluation may be high. Thus, in order to obtain efficient noise filtering without excessive computational cost, various solutions have been proposed in literature. In Aizawa and Wah (1993) and Aizawa and Wah (1994), two variants of adaptive re-sampling systems based on the progress of evolution have been proposed. In Stagge (1998), a variable sample size performed by means of a probabilistic criterion based on the solution quality is presented. In Neri et al. (2006) and Neri et al. (2008) both sample and population size are adaptively adjusted on the basis of a diversity measure. In Branke and Schmidt (2003), Branke and Schmidt (2004), and Cantú-Paz (2004) sequential approaches have been proposed which aim at reducing the sample size during the tournament selection and performing massive re-sampling only when strictly necessary.

Regarding methods which employ approximated models in order to perform the averaging, the main idea is that computation of a fitness value by means of the values of neighbour points can give, without an extra fitness evaluation, a reliable estimation of the fitness. More specifically, according to this algorithmic philosophy, the fitness value estimated by an approximated (and computationally cheap) technique is not less imprecise than the original noisy value. For example, Branke et al. (2001) and Sano et al. (2000) propose taking fitness estimates of neighbouring individuals in order to predict the fitness value of some candidate solutions. Paper Neri et al. (2008), by employing a similar philosophy, proposes construction of a local linear surrogate model (an approximate model of the true fitness function) which locally performs the noise filtering.

Other works make some assumptions regarding the noise in order to propose integration of a filtering component within the selection schemes so as to perform sorting of the solutions without the use of a large amount of samples. A theoretical study about the threshold choice is presented in Beielstein and Markon (2002). In Rudolph (2001), under the hypothesis that the noise is bounded, application of a partial order on the set of noisy fitness values is proposed.

In recent years, noise filtering components have been designed and integrated into metaheuristics as opposed to the classical ES and Genetic Algorithms (GAs). For example, in Ball and Bowler (2003) a modified Simulated Annealing (SA) has been proposed for noisy problems. In Gutjahr (2003), an implementation of Ant Colony Optimization (ACO) for noisy environments has been proposed.In Bartz-Beielstein et al. (2007), a sequential sampling procedure has been proposed for a Particle Swarm Optimization (PSO). In Pan and Wanga (2006) and Klamargias et al. (2008) very accurate statistics-based procedures for noise handling are integrated within PSO structures.

Some noise handling components have also been proposed for Differential Evolution (DE). In several papers, e.g. in Krink and Fogel (2004), it is empirically shown that the DE is not recommended for noisy problems. However, recent works propose some modifications of the DE schemes which make it very competitive with advanced metaheuristics tailored to optimization in a noisy environment. According to the explanation given in Das and Konar (2005a) and Das and Konar (2005b), DE is based on an overly deterministic structure for handling difficulties imposed by the noise. Thus, introduction of stochastic elements in the framework (the scale factor, in this case) can greatly improve the DE performance in the presence of uncertainties. By following a similar logic, in Rahnamayan et al. (2006), an opposition based DE (i.e., a DE which performs extra sampling of symmetrical points) is proposed for noisy environment and shows that generation of the opposition based points beneficially perturbs determinism of a DE structure in the presence of a noisy fitness. The fact that stochastic elements integrated within a DE can be beneficial for reaching a high performance level can be further confirmed by several studies in literature, e.g. Brest and Greiner (2006), which show the improvements for static functions in high dimensions. In addition, in Bo Liu and Ma (2008), a DE hybridized with a SA for noisy problems has been proposed.

3.3 Limited Memory and Real-Time implementations

In many real-world applications, an optimization problem must be solved despite the fact that a full power computing device may be unavailable due to cost and/or space limitations. This situation is typical of robotics and control problems. For example, a commercial vacuum cleaner robot is supposed to, over the time, undergo a learning process in order to locate where obstacles are placed in a room (e.g. a sofa, a table etc) and then perform an efficient cleaning of the accessible areas. Regardless of the specific learning process, e.g. a neural network training, the robot must contain a computational core but clearly cannot contain all the full power components of a modern computer, since they would increase the volume, complexity, and cost of the entire device. Thus, a traditional optimization meta-heuristic can be inadequate under these circumstances. In order to overcome this class of problems compact Evolutionary Algorithms (cEAs) have been designed. A cEA is an Evolutionary Algorithm (EA) belonging to the class of Estimation of Distribution Algorithms (EDAs), see Larrañaga and Lozano (2001). The algorithms belonging to this class do not store and process an entire population and all its individuals therein but on the contrary make use of a statistic representation of the population in order to perform the optimization process. In this way, a much smaller number of parameters must be stored in the memory. Thus, a run of these algorithms requires much less capacious memory devices compared to their correspondent standard EAs.

3.3.1 Binary Compact Genetic Algorithms

Compact Genetic Algorithms (*cGA*) Harik et al. (Nov, 1999) are evolutionary algorithms that mimic the behavior of conventional GAs by evolving a probability vector (PV) that describes the hypothetic distribution of a population of solutions in the search space. A cGA iteratively processes the PV with updating mechanisms that mimic the typical selection and recombination operations performed in a standard GA (sGA) until a stopping criterion is met. In Harik et al. (Nov, 1999) it is shown that the cGA is almost equivalent to a sGA with binary tournament selection and uniform crossover on a number of test problems, and also suggested some mechanisms to alter the selection pressure in the cGA. The main strength of the cGA is the significant reduction of memory requirements, as it needs to store only the PV instead of an entire population of solutions.

A building block, as discussed in the *Schemata Theorem* (see Holland (1975)), is a set of genes that as a whole give a high contribution to the fitness of an individual. In the initial population there will be some instances of the building block; then during the action of a GA the number of instances of the building block can increase or decrease. In (Harik et al. (1997)) the authors conducted an analysis about the behavior of the building blocks within a random walk model. This analysis suggests that it is possible to directly simulate this process for order-one problems[1]. The main idea is to simulate l independent random walks through the introduction of a probability vector PV, $\mathbf{p} \in [0,1]^l$, as described in Figure 14.

The general framework of the PBIL (and the EDA) is valid also for the cGA. The main difference is the connection that is made between the cGA and the sGA. Specifically for order-one problems the two algorithms are approximately equiv-

[1] By order-one problem we mean a problem that can be solved to optimality by combining only order-one schemata.

$$\begin{array}{c} \vdots \\ i-th \\ \vdots \\ j-th \\ \vdots \end{array} \begin{bmatrix} \vdots \\ \boxed{0\,1\,0\,0\,0} \\ \downdownarrows \\ \boxed{1\,1\,0\,1\,1} \\ \vdots \end{bmatrix} \Longrightarrow \begin{array}{c} \vdots \\ i-th \\ \vdots \\ j-th \\ \vdots \end{array} \begin{bmatrix} \vdots \\ \boxed{0\,1\,0\,1\,0} \\ \vdots \\ \boxed{0\,1\,0\,1\,0} \\ \vdots \end{bmatrix}$$

FIGURE 14 Simulation of substitution process on the virtual population of the cGA

```
counter $t = 0$
{** **PV** initialization **}
**for** $i = 1 : n$ **do**
    initialize $PV[i] = 0.5$
**end for**
**while** budget condition **do**
    generate 2 individuals **a b** by means of **PV**
    $[\mathbf{winner}, \mathbf{loser}] = compete(\mathbf{a}, \mathbf{b})$
    {** **PV** Update **}
    **for** $i = 1 : n$ **do**
        **if** $winner[i]! = looser[i]$ **then**
            **if** $winner[i] = 1$ **then**
                $PV[i] = PV[i] + 1/N_p$
            **else**
                $PV[i] = PV[i] - 1/N_p$
            **end if**
        **end if**
    **end for**
    counter update $t = t + 1$
**end while**
```

FIGURE 15 cGA pseudo-code

alent. The basic idea in the cGA implementation is about the selection process: imagine a selection scheme in which two individual (i.e. the *i*-th and *j*-th individuals) are randomly chosen in a "virtual" population of *n* individuals and suppose to delete the worst one and to make a copy of better one, as shown in Figure 14.

This scheme is equivalent to a steady state binary tournament selection where the proportion of the winning alleles will increase by $1/n$. For instance in the Figure 14 the proportion of 0's in the first, fourth and fifth position will increase by $1/n$. At the other position the proportion will remain the same. This suggest that an update rule increasing a gene's proportion by $1/n$ simulates a small step in the action of a GA with a "virtual" population of size *n*.

3.3.2 Elitism in cGAs

The great advantage of low computational cost and memory usage of the cGA has a drawback when facing it with complex problems such as fully deceptive and multimodal problems. In Ahn and Ramakrishna (Aug 2003) the authors propose two new elitism-based compact GAs that belong to a class of EDA -the *persistent compact GA* (*pe-cGA*) and the *nonpersistent compact GA* (*ne-cGA*). The main objective is to efficiently solve difficult problems using the cGA without unduly compromising on memory and computational cost.

The pe-cGA deals with the problem of lack of memory by simply retaining the best solution found so far, thereby mitigating the disruptive effects of uniform cross-over. Another interesting aspect is that in this configuration the pe-cGA shows a close connection to (1+1)-ES with self-adaptive mutation as discussed above. This can be interpreted as a revelation of the relationship between EDA and ES. In addition, the pe-cGA offers some advantages over (1+1)-ES to GA practitioners. On the other hand, it can be further improved by controlling the strength of elitism. This scheme is used to prevent the rapid degeneration of genetic diversity, thereby improving the quality of the solution. The only difference of ne-cGA from pe-cGA is in the operational mechanism, whereby a chromosome that is randomly generated replaces the elite one when a certain criterion indicating the allowable length of inheritance is not satisfied.

In sGA elitism provides a means for reducing the genetic drift by ensuring that the best chromosome(s) is allowed to pass/copy their traits to the next generation. Elitism can increase the selection pressure by preventing the loss of "low" salience genes of chromosomes due to deficient selection pressure and it improves the performance with regard to optimality and convergence of GAs in many cases. However the degree of elitism should be adjusted in order to prevent premature convergence. In general a more difficult problem requires a higher selection pressure for finding a better solution. This is because higher selection pressure offsets the disruptive effects of uniform crossover, thereby encouraging convergence to a better solution. The introduction of this mechanism in cGA without any extra memory consumption can be obtained replacing only the loser individual by the new individual in the standard cGA pseudocode. In other words, the winner is never eliminated in so far as a better chromosome has not yet been produced from the PV. This is the framework of the pe-cGA as reported in Figure 16.

While gaining in selection pressure, on the other hand, the pe-cGA may lose genetic diversity owing to implied elitism leading to premature convergence and to a suboptimal solution. In order to prevent this behavior in ne-CGA a parameter η -the allowable length of the elite chromosome's inheritance- is introduced to control the strength of elitism. With this modification the pseudocode of the ne-cGA becomes:

Two remarkable results are discussed in Ahn and Ramakrishna (Aug 2003) and are reported here: equivalence of pe-cGA and (1+1)-ES and remarks on the maximum length of inheritance η.

Theorem 3.3.1 *The pe-cGA is equivalent to the (1+1)-ES with self adaptive mutation.*

Proof Let $L : \Theta \rightarrow \mathbb{R}$ be the loss function to be minimized. Consider the Markovian process $\mathbf{X}_{k \geqslant 0}$ generated by a stochastic algorithm,

$$\mathbf{X}_{k+1} = \begin{cases} \mathbf{X_k} + l_k \mathbf{Z}_k, & \text{if } L(\mathbf{X}_k + l_k \mathbf{Z}_k) < L(\mathbf{X}_k) \\ \mathbf{X}_k, & \text{otherwise} \end{cases} \tag{29}$$

```
counter $t = 0$
{** **PV** initialization **}
**for** $i = 1 : n$ **do**
    initialize $PV[i] = 0.5$
**end for**
generate **elite** by means of **PV**
**while** budget condition **do**
    generate 1 individual **a** by means of **PV**
    {** Elite Selection **}
    $[\mathbf{winner}, \mathbf{loser}] = compete(\mathbf{a}, \mathbf{elite})$
    **if** $\mathbf{a} == \mathbf{winner}$ **then**
        $\mathbf{elite} = \mathbf{a}$
    **end if**
    {** **PV** Update **}
    **for** $i = 1 : n$ **do**
        **if** $winner[i]! = looser[i]$ **then**
            **if** $winner[i] = 1$ **then**
                $PV[i] = PV[i] + 1/N_p$
            **else**
                $PV[i] = PV[i] - 1/N_p$
            **end if**
        **end if**
    **end for**
    counter update $t = t + 1$
**end while**
```

FIGURE 16   pe-cGA pseudo-code

```
counter $t = 0$ and $\theta = 0$
{** **PV** initialization **}
**for** $i = 1 : n$ **do**
    initialize $PV[i] = 0.5$
**end for**
generate **elite** by means of **PV**
**while** budget condition **do**
    generate 1 individual **a** by means of **PV**
    {** Elite Selection **}
    $[\mathbf{winner}, \mathbf{loser}] = compete(\mathbf{a}, \mathbf{elite})$
    $\theta = \theta + 1$
    **if** $\mathbf{a} == \mathbf{winner}$ OR $\theta \geq \eta$ **then**
        $\mathbf{elite} = \mathbf{a}$
        $\theta = 0$
    **end if**
    {** **PV** Update **}
    **for** $i = 1 : n$ **do**
        **if** $winner[i]! = looser[i]$ **then**
            **if** $winner[i] = 1$ **then**
                $PV[i] = PV[i] + 1/N_p$
            **else**
                $PV[i] = PV[i] - 1/N_p$
            **end if**
        **end if**
    **end for**
    counter update $t = t + 1$
**end while**
```

FIGURE 17   ne-cGA pseudo-code

where l_k is the step length control parameter that is increased as far as mutation improves solutions. Each random vector $\mathbf{Z}_k$ has a joint probability density function (pdf) with independent marginal densities. The (29) can model an (1+1)-ES with self-adaptive mutation if the step length control parameter is changed when the relative frequency of improving mutations is below or above some threshold within τ trials. (see Rudolph (1999)).
Now let $\mathbf{Y}_k$ be a random vector generated from the PV. The probability distribution of $\mathbf{Y}_k$ is given by

$$F_{\mathbf{Y}_k}(y_1, \ldots, y_l) = \prod_{i=1}^{l} \mathbf{P}_k(i) \tag{30}$$

where $\mathbf{P}_k(i)$ represents the i-th element of PV at the k-th generation.
By employing $\mathbf{Y}_k$ the pe-cGA can described by

$$\mathbf{X}_{k+1} = \begin{cases} \mathbf{Y}_k, & \text{if } L(\mathbf{Y}_k)) < L(\mathbf{X}_k) \\ \mathbf{X}_k, & \text{otherwise} \end{cases} \tag{31}$$

With some straightforward manipulation and the relation

$$l_k \mathbf{Z}_k = \mathbf{Y}_k - \mathbf{X}_k \tag{32}$$

with the pdf of $\mathbf{Z}_k$ computed as

$$F_{\mathbf{Z}_k}(z_1, \ldots, z_l) = F_{\mathbf{Z}_k}(\mathbf{z}) = |l_k| F_{\mathbf{Y}_k}(l_k \mathbf{z} + \mathbf{X}_k). \tag{33}$$

With equations (31) and (33) it is possible to conclude that the two algorithms (i.e., the pe-cGA and the (1+1)-ES with self adaptive mutation) follow the identical model ■

The second main result in Ahn and Ramakrishna (Aug 2003) relates to the length of inheritance η.

Theorem 3.3.2 *The allowable length of inheritance η should not exceed the simulated population size n. That is, $\eta < n$.*

Proof Define $\mathbf{W}_k$ as a random vector generated from a random PV set to 0.5. Let $\mathbf{V}_k$ be another vector defined as $(\mathbf{P}_{k+1} - \mathbf{P}_k)$. This is the inter generational changes of the PV. Let us assume that the winner takes the PV to converge regardless of optimality of the solution. It is assumed that the winner always defeat its competitor when the PV converges, irrespective of the optimality of the solution. The PV evolution can be described by

$$\mathbf{P}_{k+1}(i) = \begin{cases} \mathbf{P}_k(i) + E\left(V_k(i) \,|X_k(i) = 1\right), & \text{if } \mathbf{X}_k(i) = 1 \\ \mathbf{P}_k(i) + E\left(V_k(i) \,|X_k(i) = 0\right), & \text{if } \mathbf{X}_k(i) = 0 \end{cases} \tag{34}$$

Each conditional expectation on $\mathbf{V}_k(i)$ can be computed as follows:

$$E[\mathbf{V}_k | \mathbf{X}_k(i) = 1] = \frac{1}{n} p[\mathbf{V}_k = \frac{1}{n} | \mathbf{X}_k = 1] \tag{35}$$

$$E[\mathbf{V}_k | \mathbf{X}_k(i) = 0] = -\frac{1}{n} p[\mathbf{V}_k = -\frac{1}{n} | \mathbf{X}_k = 0] \tag{36}$$

On the other hand each conditional probability is seen to be

$$p[\mathbf{Y}_k(i) = 0] = 1 - \mathbf{P}_k(i) \tag{37}$$
$$p[\mathbf{Y}_k(i) = 1] = \mathbf{P}_k(i) \tag{38}$$

Let us consider (37) first. To increase the i-th element of PV by $\frac{1}{n}$ given that i-th gene of the winner has "1", the i-th gene of its competitor should generate "0" because the winner in $(k-1)$-th generation becomes a winner in the present k-th generation. Using (35), (36), (37) and (38) it is possible to rewrite (34) as

$$\mathbf{P}_{k+1}(i) = \begin{cases} \mathbf{P}_k(i) + \frac{1}{n}\{1 - \mathbf{P}_k(i)\}, & \text{if } \mathbf{X}_k(i) = 1 \\ \mathbf{P}_k(i) - \frac{1}{n}\mathbf{P}_k(i), & \text{if } \mathbf{X}_k(i) = 0 \end{cases} \tag{39}$$

Define $\mathbf{P}_k^M$ and $\mathbf{P}_k^m$ as $\max_{\forall j} \mathbf{P}_k(j)$ and $\min_{\forall j} \mathbf{P}_k(j)$, respectively. None of the element in the PV should be brought to convergence by the same winner because there is no guarantee that the chromosome leads to an optimal solution. It means that:

$$\mathbf{P}_{k+\eta}^M \;<\; 1 \tag{40}$$
$$\mathbf{P}_{k+\eta}^m \;>\; 0. \tag{41}$$

This means that the allowable length of inheritance for binary pe-cGA must satisfy:

$$\begin{aligned}
\mathbf{P}_{k+\eta}^M &= (1-\frac{1}{n})\mathbf{P}_{k+\eta-1}^M + \frac{1}{n} \\
&= (1-\frac{1}{n})^2\mathbf{P}_{k+\eta-2}^M + \{(1-\frac{1}{n}) + 1\}\frac{1}{n} \\
&= \dots \\
&= (1-\frac{1}{n})^{\eta}\mathbf{P}_k^M + \{1-(1-\frac{1}{n})^{\eta}\} \\
&\approx (1-\frac{\eta}{n})\mathbf{P}_k^M\frac{\eta}{n} < 1.
\end{aligned}$$

Thus the allowable length of inheritance η must satisfy

$$\eta < n \tag{42}$$

By a similar method it can be easily shown that also for $\mathbf{P}_k^m > 0$ it must be valid the (42) ■

3.3.3 Real-Coded Compact Genetic Algorithms

In (Mininno et al. (2008)), it is presented a novel cGA that combines the non-persistent elitist compact GA proposed in (Ahn and Ramakrishna (Aug 2003)) with a real-valued chromosome representation. The real-valued coding has been widely used in evolutionary algorithms (including GAs Michalewicz (1992)) to avoid the additional computational cost related to the binary-to-float conversions. To use a real-valued coding in a cGA, a number of modifications are necessary. Let us consider a modified minimization problem in an m-dimensional hyper-rectangle, that is

$$L : \hat{\Theta} \rightarrow \mathbb{R}. \tag{43}$$

Here the set $\hat{\Theta} \subseteq \mathbb{R}^m$ represents the hyper-rectangle of $\mathbb{R}^m$ with each component of $\theta \in \hat{\Theta}$ normalized to the interval $[-1, +1]$. The estimate of the single (binary) gene distribution in a cGA can be specified with a scalar in [0,1] expressing the probability of finding a "0" or a "1" in the single gene. As the rcGA uses real-valued genes, the distribution of the single gene in the hypothetical population must be described by a Probability Density Function (PDF) defined on the normalized interval $[-1, +1]$. A variety of different PDFs could be used for this purpose. Here it is assumed that the distribution of the j-th gene can be described with a Gaussian PDF with mean x_j and standard deviation σ_j. More precisely, since the Gaussian PDF is defined in $] - \inf, + \inf[$, it is proposed a "Gaussian-Shaped" PDF defined on $[-1, +1]$ whose height is normalized so that its area is equal to one. In Figure 18 it is illustrated the effect of the normalization procedure on a Gaussian PDF with zero mean and standard deviations equal to 1 and 10, respectively.

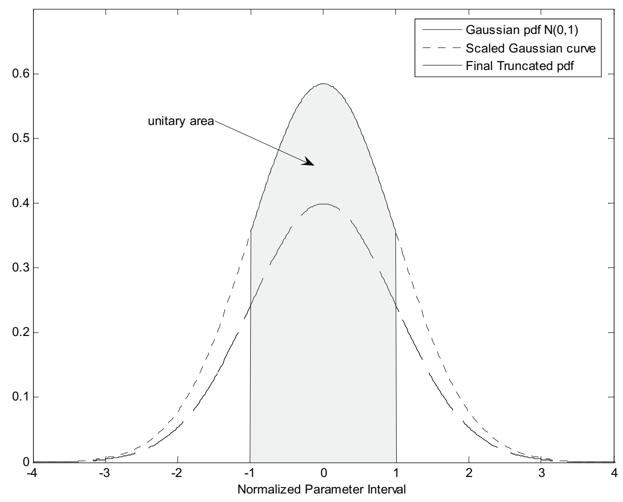


FIGURE 18 Normalization of a truncated Gaussian curve (with standard deviations $\sigma = 1$) so as to obtain a probability density function with a non-zero support only in the normalized interval $[-1, 1]$.

The height normalization has a generally negligible computational cost, as it is obtained by means of a simple numerical procedure (Spall (2003) and Cody (1969)). Let us consider the normal distribution $N(\hat{x}, \sigma)$ and standard error func-

```
counter $t = 0$ and $\theta = 0$
{** **PV** initialization **}
**for** $i = 1 : n$ **do**
    initialize $PV[i] = 0.5$
**end for**
generate **elite** by means of **PV**
**while** budget condition **do**
    generate 1 individual **a** by means of **PV**
    {** Elite Selection **}
    $[\mathbf{winner}, \mathbf{loser}] = compete(\mathbf{a}, \mathbf{elite})$
    $\theta = \theta + 1$
    **if** $\mathbf{a} == \mathbf{winner}$ OR $\theta \geq \eta$ **then**
        $\mathbf{elite} = \mathbf{a}$
        $\theta = 0$
    **end if**
    {** **PV** Update **}
    **for** $i = 1 : n$ **do**
        **if** $winner[i]! = looser[i]$ **then**
            update mean and variance of $PV$
        **end if**
    **end for**
    counter update $t = t + 1$
**end while**
```

FIGURE 19 ne-cGA pseudo-code

tion $erf(x)$:

$$pdf(N(\hat{x}, \sigma), x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\hat{x})^2}{2\sigma^2}} \tag{44}$$

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{t^2} dt. \tag{45}$$

So, it is possible to scale the normal distribution $N(0, \sigma)$ multiplying its pdf by the factor:

$$K_{scale} = \left(erf\left(\frac{1}{\sqrt{2}\sigma} \right) \right)^{-1}. \tag{46}$$

Therefore, the PV became in the rCGA an $m \times 2$ matrix specifying the two parameters of the PDF of each single gene. Thus, it is possible to define:

$$PV^k = [\mathbf{x}^{(k)} \boldsymbol{œ}^{(k)}], \tag{47}$$

where $\mathbf{x}^{(k)} = \left[x_1^{(k)}, \ldots, x_m^{(k)} \right]$ is the vector of the mean values, $\boldsymbol{œ}^{(k)} = \left[\sigma_1^{(k)}, \ldots, \sigma_m^{(k)} \right]$ is the vector of the standard deviations, and k is the iteration index. Here it is reported the pseudocode description of the real-neCGA; the main difference with the bcGA regards the generation of the initial population and the update rule. The initial population is generated by setting $x_j^{(1)} = 0$ and $\sigma_j^{(1)} = \lambda$ where λ is a large positive constant (e.g., $\lambda = 10$, see Figure 18). In this way after height normalization it is possible to obtain a PDF that approximates sufficiently the uniform distribution $U(-1, 1)$ that is almost always used to generate initial individuals in real-valued GAs.

The update rule of the real cGA are:

$$x_j^{(k+1)} = x_j^{(k)} + \frac{1}{n}(w_i - l_i) \quad (48)$$

$$\left[\sigma_j^{(k+1)}\right]^2 = \left[\sigma_j^{(k)}\right]^2 + \left[x_j^{(k)}\right]^2 - \left[x_j^{(k+1)}\right]^2 + \frac{1}{n}\left(\left[w_j\right]^2 - \left[l_j\right]^2\right) \quad (49)$$

The update rule is designed so as to replicate the typical operations of a bcGA. As in Harik et al. (Nov, 1999), the idea is to mimic a steady state binary tournament selection. This operator simply replaces the looser with a copy of the winner, and consequently alters also the compact, probabilistic description of the population. In particular, while the mean is simply moved in the direction of the winner with a step whose size depends on the population size n, the standard deviation is subject to a less transparent effect, due to its quadratic structure.

4 BUILDING UPON THE STATE-OF-THE-ART IN OPTIMIZATION FOR CONTROL ENGINEERING

As mentioned in the previous chapter many issues must be taken in account when an evolutionary algorithm is applied to the solution of an optimal control problem. In particular the actual implementation of the algorithm on the real-time control platform must cope with the lack of a full power computer, thus it must use a very low amount of memory and computational power. On the other hand the presence of nonlinearities, sensors and approximations inject in the signals of the control loop some noise, resulting in a noisy fitness function to be optimized. The articles presented in this work can be subdivided into two categories. PI to PIII present better variation of the compact framework presented in Mininno et al. (2008). PIV presents a variation of the noise analysis mechanism within a memetic scheme. Then PV discuss the implementation of both a cEA framework and a component to reduce the influence of the noise on the optimization process.

4.1 Compact Differential Evolution

4.1.1 Objectives

Although the general motivation is similar to that of cGA and its variants, there are important issues especially related to differential evolution (DE) algorithm. First, the survivor selection scheme that performs a pair-wise comparison between the performance of a parent solution and its corresponding offspring. This logic can be naturally encoded into a compact algorithm unlike the case of a selection mechanism typical of genetic algorithms (GAs), e.g., tournament selection. In other words a DE can be straightforwardly encoded into a compact algorithm without losing the basic working principles (in terms of survivor selection).

The second issue is related to the DE search logic. A DE algorithm contains a limited amount of search moves which might contribute to jeopardizing the generation of high quality solutions which improve upon the current best performance. A cDE algorithm, due to its nature, does not hold a full population of individuals but contains its information in distribution functions and samples the individuals from it when necessary. Thus, unavoidably some extra randomness, with respect to original DE, is introduced.

4.1.2 Results

The suitability of cDE in the resolution of a sophisticated control problem is demonstrated in the paper by showing the results of an optimized neural network controller for a linear electric drive. These motors are often directly coupled with their load, and the absence of reduction/transmission gears makes the positioning performance strongly influenced by the various uncertainties related to electro-mechanical phenomena (striction, cogging forces), which therefore must be compensated with appropriate strategies. Beside that, the performance and robustness of this new compact algorithm is widely discussed, comparing its performance with those of other compact and population based algorithms.

4.1.3 Relation with the whole context

The main motivation behind this work is the enhancement of the performance of a standard cEA in terms of robustness and overall performance on various optimization benchmarks. In the context perspective, this work has been able to demonstrate the suitability of the DE scheme within a compact framework in order to obtain better performance in both numerical and experimental tests.

4.2 Memetic Compact Differential Evolution

4.2.1 Objectives

The extension of the cDE approach presented in the previous paper is here considered. The adoption of a memetic computing approach to the cDE framework is discussed in order to achieve an overall better performance in terms of robustness with respect to a set of benchmark problems. The low memory needs of the algorithm remains in this approach a main goal, and it is obtained also in the local searcher, which implements a hypercube around a certain point and samples points within the hypercube volume is integrated in order to assist, in a memetic fashion.

4.2.2 Results

The combination of these search structures makes the algorithm a compact structure which requires a very limited amount of memory resources and can therefore be implemented into computational devices characterized by modest computational power. This feature makes the proposed algorithm appealing for implementation in cheap and portable hardware. In order to test the suitability of the proposed approach, McDE has been tested for a challenging real-world application, i.e. the control of a Cartesian robot.

4.2.3 Relation with the whole context

It must be noticed that there is a lack, in literature, of modern compact MC approaches, especially for continuous optimization problems.The proposed McDE aims thus at combining a high performance with the capability to run on devices characterized by a limited memory.This goal is achieved by defining a compact structure for both evolutionary framework and local search, and hybridizing them under the supervision of a memetic logic.

4.3 Estimation Distribution Differential Evolution

4.3.1 Objectives

Differential evolution uses the generation of new solutions by combining randomly chosen parents algebraically. If for some reason, the algorithm does not succeed in generating offspring solutions which outperform the corresponding parent, the search is repeated again with similar step size values and will likely fail by falling into an undesired stagnation condition. In this fashion, this paper proposes a novel, relatively simple, and efficient adaptive scheme for performing control parameter setting in DE frameworks. The proposed control scheme samples the scale factor and crossover rate control parameters from a truncated adaptive Gaussian function, which adapt during the optimization process towards the most promising values and attempt to follow the needs of the evolution.

4.3.2 Results

The resulting algorithm has been proved robust and efficient over a set of various test problems. The proposed algorithm, despite its simplicity, offers a good performance compared to other modern sophisticated algorithms based on a Differential Evolution structure. Numerical results prove that the proposed algorithm is competitive for many test problems and outperforms the other algorithms considered in this study in most of considered cases.

4.3.3 Relation with the whole context

In a standard population based framework a novel mechanism for the control parameters adaptation has been studied. This algorithm is directly inspired by the Gaussian adaptation system behind all the real valued cEA presented in this work. The results of this hybridization with a population based algorithms suggest that this component would be beneficial for further improvements of cDE and McDE in future works.

4.4 Noise Analysis Memetic Differential Evolution

4.4.1 Objectives

The handling of noise and thus optimization despite the presence of noise, as discussed in chapter 2, are very important in this day and age for industries and, more generally, for the solution of real-world problems. If the noise is unavoidable and cannot be eliminated from the objective function computation, the optimization algorithm should take into account this difficulty of the problem and perform its action notwithstanding the presence of noise. The main motivation of this paper has been the explicitly addressing of the problem of noisy optimization by means of a Memetic approach. An algorithm which combines, within a DE framework, a controlled randomization of the parameters proposed in, a noise analysis component and a scale factor local search logic proposed.

4.4.2 Results

Memetic approaches have rarely been applied in noisy optimization problems and in literature there are no systematic studies on the performance of MAs in noisy environments. In this sense, this paper offers a valuable first contribution to the scientific community. The algorithm proposed has been tested on a broad and various set of benchmark test problems. The comparison carried out against six different algorithms confirms that a standard DE framework is inadequate in a noisy environment, the opposition based logic is beneficial in improving upon the standard DE performance but its benefits are marginal with respect to other modifications which can be carried out. The main drawback of the proposed approach is the high amount of parameters that need to be set due to the algorithmic complexity.

4.4.3 Relation with the whole context

Since the noise handling is a key point in on-line optimization for control applications, this work has been conducted to explore and better understand the application of a noise analysis component in a memetic framework. This work has produced a better understanding of this component in order to enhance the noise mitigation and therefore to successfully implement it in a compact framework, as described in the last paper here included

4.5 Noise Analysis Compact Genetic Algorithm

4.5.1 Objectives

As a final sum of the research efforts through the realization of a novel approach to on-line optimization for control systems, a first combination of both compact framework and noise analysis is presented in this paper. That is, if the application requires the solution of an optimization problem despite both limited hardware conditions and the presence of noise, a population based approach is not applicable and the employment of classical compact algorithms (as well as classical population based algorithms) might lead to unsatisfactory results due to the pernicious effect of noise. In this paper, we propose the first (to our knowledge) implementation of compact algorithm integrating a noise handling component.

4.5.2 Results

The proposed algorithm can be useful for those engineering system characterized by a limited memory and the presence of measurement systems which affect the fitness landscape and make it noisy. The resulting algorithm integrates an adaptive system for performing the minimum amount of fitness evaluations and still reliably selecting the elite individual for the subsequent algorithmic comparison. Numerical results show that the noise analysis system efficiently enhances the performance of a standard compact genetic algorithm. The experiments have been repeated for both persistent and nonpersistent elitist schemes. Results show that the nonpersistent scheme al- lows a more robust behavior with respect to the persistent elitism.

4.5.3 Relation with the whole context

This final paper can be considered the conceptual *summa* of all the efforts in finding a good compromise in real-time engineering optimization. Both low memory requirements and noise handling have been achieved, thus demonstrating as a prove of concept the possibility of obtaining a performing optimization and control framework for micro controllers.

5 CONCLUSION

The evolutionary approach has proved particularly successful in problems that are difficult to formalize mathematically, and which are therefore not conducive to analysis. This includes systems that are highly nonlinear, that are stochastic, and that are poorly understood. This characteristic makes them suitable for the resolution of control problems where a set of decisional variables can concur in the composition of the overall performance of the system. The suitability of this approach however must cope with some limitation that can occur in the actual implementation of this techniques on real control hardware. The actual implementation of the algorithm on the real-time control platform must cope with the lack of a full power computer, thus it must use a very low amount of memory and computational power. On the other hand the presence of nonlinearities, sensors and approximations inject in the signals of the control loop some noise, resulting in a noisy fitness function to be optimized. In this work both these aspects have been deeply analyzed and discussed. A survey of the main optimization frameworks and a deep review of the litterature regarding the application of such a paradigma within control frameworks has been shown.

During this work, a novel class of compact algorithm has been created as an ibridization of the standard real valued compact framework with the Differential Evolution framework. This led to a new set of more roboust algorithm, tested on real robotic testbench. At the same time the inclusion of a noise adaptive component to handle noisy fitnesses, that often compromise the results of optimization processes, has been defined and tested.

YHTEENVETO (FINNISH SUMMARY)

Viimeisen vuosikymmenen aikana optimointi käytännön sovelluksissa on saanut kasvavaa huomiota varsinkin teollisuudessa. Erityisesti optimointi on osoittautunut käytännölliseksi ja tehokkaaksi työkaluksi säätötekniikkaan liittyvissä ongelmissa. Optimointialgoritmin toteutuksen reaaliaikaiselle kontrollialustalle täytyy selviytyä ilman tehokasta tietokonetta, joten sen täytyy toimia rajoitetulla muistilla ja laskentateholla. Toisaalta sensorien ja likiarvojen aiheuttamat epälineaarisuudet näkyvät häiriönä optimoitavassa funktiossa. Tässä työssä esitellään ratkaisuja molempiin edellämainittuihin haasteisiin ja näytetään kuinka uusi algoritmimalli voi syntyä tällaisista toteutusongelmista, jotka jäävät usein huomiotta teoreettisissa lähestymistavoissa.

Tässä väitöksessä esitellään joukko uusia ratkaisualgoritmeja monimutkaisiin käytännön ongelmiin säätötekniikassa. Kaksi uutta algoritmia esitellään funktioille joissa esiintyy häiriötä. Lisäksi ehdotetaan uutta satunnaisdistribuutioon perustuvaa mukautuvaa systeemiä funktioille joilla on useita lokaaleja minimejä ja maksimeja. Erityisen tärkeänä osana tätä väitöskirjaa on kompaktin differentiaalievoluution käsitteen määrittely optimointiongelmille rajallisella laitteistolla. Lopuksi esitellään edellä mainitun algoritmin kehitys osana käytännön sovellusta.

REFERENCES

Acarnley, P. P., da Silva, W. G. & Finch, J. W. 2000. Application of genetic algorithms to the online tuning of electric drive speed controllers. IEEE Transactions On Industrial Electronics 27, 217-219.

Ahn, C. W. & Ramakrishna, R. S. Aug 2003. Elitism-based compact genetic algorithms. IEEE Transactions On Evolutionary Computation 7, 367-385.

Aizawa, A. N. & Wah, B. W. 1993. Dynamic control of genetic algorithms in a noisy environment. In Proceedings of the Conference on Genetic Algorithms, 48-55.

Aizawa, A. N. & Wah, B. W. 1994. Scheduling of genetic algorithms in a noisy environment. Evolutionary Computation 2 (2), 97–122.

Anderson, B., Moore, A. & Cohn, D. 2000. A nonparametric approach to noisy and costly optimization. In International Conference on Machine Learning.

Arnold, D. V. & Beyer, H.-G. 2003. A comparison of evolution strategies with other direct search methods in the presence of noise. Computational Optimization and Applications 24 (1), 135–159.

Arnold, D. V. & Beyer, H.-G. 2006. A general noise model and its effects on evolution strategy performance. IEEE Transactions on Evolutionary Computation 10 (4), 380–391.

Ball, R. C. & Bowler, N. E. 2003. Stochastic annealing. Physical Review Letters 91 (3), 03020-1–03020-4.

Bartz-Beielstein, T., Blum, D. & Branke, J. 2007. Particle swarm optimization and sequential sampling in noisy environments. In K. F. Doerner et al. (Ed.) Metaheuristics, Vol. 39. Springer. Operations Research/Computer Science Interfaces Series, 261–273.

Beielstein, T. & Markon, S. 2002. Threshold selection, hypothesis test, and DOE methods. In Proceedings of the IEEE Congress on Evolutionary Computation, 777-782.

Beyer, H.-G. 1993. Toward a theory of evolution strategies: Some asymptotical results from the (1,+ λ)-theory. Evolutionary Computation 1 (2), 165–188.

Beyer, H. G. & Sendhoff, B. 2006. Functions with noise-induced multimodality: a test for evolutionary robust optimization-properties and performance analysis. IEEE Transactions on Evolutionary Computation 10 (5), 507–526.

Bobbin, J. & Yao, X. 1997. Solving optimal control problems with a cost changing control by evolutionary algorithm. Proceedings IEEE International Conference on Evolutionary Computation, 331-336.

Boussak, M. & Capolino, G. 1992. Recursive least squares rotor time constant identification for vector controlled induction machines. Electrical machines and Power Systems 20, 137-147.

Bo Liu, X. Z. & Ma, H. 2008. Hybrid differential evolution for noisy optimization. In Proceedings of the IEEE Congress on Evolutionary Computation, 587–592.

Branke, J., Schmidt, C. & Schmeck, H. 2001. Efficient fitness estimation in noisy environments. In L. Spector et. al (Ed.) Genetic and Evolutionary Computation Conference. Morgan Kaufmann, 243–250.

Branke, J. & Schmidt, C. 2003. Selection in the presence of noise. In E. Cantu-Paz (Ed.) Proceedings of Genetic and Evolutionary Computation Conference, Vol. 2723. Springer. Lecture Notes in Computer Science, 766–777.

Branke, J. & Schmidt, C. 2004. Sequential sampling in noisy environments. In Parallel Problem Solving from Nature, Vol. 3242. Springer. Lecture Notes in Computer Science, 202–211.

Brest, J. & Greiner, S. 2006. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Transactions on Evolutionary Computation 10 (6), 646–657.

Burke, E. K., Kendall, G. & Soubeiga, E. 2003. A tabu search hyperheuristic for timetabling and rostering. Journal of Heuristics 9 (6), 451–470.

Bäck, T., Hammel, U. & Schwefel, H.-P. Apr, 1997. Evolutionary computation: Comments on the history and current state. IEEE Transactions On Evolutionary Computation 1, 3-17.

Cantú-Paz, E. 2004. Adaptive sampling for noisy problems. In Proceedings of the Genetic and Evolultionary Computation Conference. Springer. Lecture Notes in Computer Science, 947-958.

Caponio, A., Neri, F. & Tirronen, V. 2009. Super-fit control adaptation in memetic differential evolution frameworks. Soft Computing-A Fusion of Foundations, Methodologies and Applications 13 (8), 811–831.

Caponio, A., Cascella, G. L., Neri, F. & Sumner, M. 2007. A fast adaptive memetic algorithm for on-line and off-line control design of pmsm drives. IEEE Transactions on System Man and Cybernetics-part B 37 (1), 28–41.

Chen, B. & Cheng, Y. 1998. A structure specified h-infinity optimal control design or prac- tical applications: A genetic approach. IEEE Transactionon Control System Technolo- gies 6, 707-718.

Chen, G., Tang, K. S., Man, K. F. & Kwong, S. 2001. An optimal fuzzy pid controller. IEEE Transactions On Industrial Electronics 28, 757-765.

Chipperfield, A. & Fleming, P. 1996. Multiobjective gas turbine engine controller using genetic algorithms. IEEE Transactions On Industrial Electronics 43, 583-587.

Cody, W. J. 1969. Rational chebyshev approximations for the error function. Mathematical Computation 8, 631-638.

Cowling, P., Kendall, G. & Soubeiga, E. 2000. A hyperheuristic approach to scheduling a sales summit. In Proceedings of the Third International Conference on Practice and Theory of Automated Timetabling, Vol. 2079. Springer. Lecture Notes in Computer Science, 176–190.

Cupertino, F., Mininno, E. & Naso, D. 2004. Online genetic design of anti-windup unstructured controllers for electric drives with variable load. IEEE Transactions On Evolutionary Computation 8, 347-365.

Das, S. & Konar, A. 2005a. An improved differential evolution scheme for noisy optimization problems. In Pattern recognition and machine intelligence, Vol. 3776. Springer. Lecture Notes in Computer Science, 417–421.

Das, S. & Konar, A. 2005b. Improved differential evolution algorithms for handling noisy optimization problems. In Proceedings of the IEEE Congress on Evolutionary Computation, Vol. 2, 1691–1698.

Davidon, W. C. 1991. Variable metric method for minimization. SIOPT 1 (1), 1-17.

Dawkins, R. 1976. The Selfish Gene. Oxford: Clarendon Press.

Di Pietro, A., While, L. & Barone, L. 2004. Applying evolutionary algorithms to problems with noisy, time-consuming fitness functions. In Proceedings of the IEEE Congress on Evolutionary Computation, Vol. 2, 1254–1261.

Duivenbode, R., Rey, J. P., Brunisma, J. A. & Krohling, R. A. 1998. Genetic algorithms applied to controller design of an electric drive system. Proceedings of IEEE ISIE.

Eberhart, R. & Kennedy, J. 1995. A new optimizer using particle swarm theory. In Proceedings of the Sixth International Symposium on Micromachine and Human Science, Vol. 1, 39-43.

Fleming, P. 2002. Evolutionary algorithms in control system engineering: a survey. Control Engineering Practice 10, 1223–1241.

Fonseca, C. M. & Fleming, P. J. 1998. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. part i: A unified formulation. part ii: Applica- tion example. IEEE Transactions On System Man and Cybernetics, Part A 28, 26-47.

Goldberg, D. & Lingle, R. 1985. Alleles locia and the traveling salesman problem. ICGA 1 (1), 154-159.

Goldberg, D. E. 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley.

Gong, M., Jiao, L. & Zhang, L. 2010. Baldwinian learning in clonal selection algorithm for optimization. Information Sciences 180 (8), 1218–1236.

Grum, N., Schroder, P., Green, B. & Fleming, P. J. 2001. On-line evolution of robust control systems: An industrial active bearing application. Control Engineering Practice 9, 37-49.

Gutjahr, W. 2003. A converging ACO algorithm for stochastic combinatorial optimization. In A. Albrecht & K. Steinhoefl (Eds.) Stochastic Algorithms: Foundations and Applications, Vol. 2827. Springer-Verlag. Lecture Nptes in Computer Science, 10-25.

Hammel, U. & Bäck, T. 1994. Evolution strategies on noisy functions, how to improve convergence properties. In Y. Davidor, H. P. Schwefel & R. Männer (Eds.) Proceedings of Parallel Problem Solving from Nature, Vol. 866. Springer-Verlag. Lecture Notes in Computer Science, 159-168.

Harik, D., Cantu-Paz, E., Goldberg, D. & Miller, A. 1997. The gambler's ruin problem, genetic algorithms and the sizing of populations. In IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence. ⟨URL:citeseer.ist.psu.edu/article/harik97gamblers.html⟩.

Harik, G. R., Lobo, F. G. & Goldberg, D. E. Nov, 1999. The compact genetic algorithm. IEEE Transactions On Evolutionary Computation 3, 287-297.

Hart, W. E., Krasnogor, N. & Smith, J. E. 2004. Memetic evolutionary algorithms. In W. E. Hart, N. Krasnogor & J. E. Smith (Eds.) Recent Advances in Memetic Algorithms. Berlin, Germany: Springer, 3-27.

Hoffmann, F. 2001. Evolutionary algorithms for fuzzy control system design. Proceedings IEEE 89, 1318-1333.

Holland, J. H. 1975. Adaptation in Natural and Artificial Systems. Univ. of Michigan Press.

Hooke, R. & Jeeves, T. 1961. Direct search solution of numerical and statistical problems. Journal of the ACM 8, 212-229.

Jin, Y. & Branke, J. 2005. Evolutionary optimization in uncertain environments-a survey. IEEE Transactions on Evolutionary Computation 9 (3), 303–317.

Kendall, G., Cowling, P. & Soubeiga, E. 2002. Choice function and random hyperheuristics. In Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning, 667–71.

Klamargias, A. D., Parsopoulos, K. E. & Vrahatis, M. N. 2008. Particle filtering with particle swarm optimization in systems with multiplicative noise. In Proceedings of the 10th annual conference on Genetic and evolutionary computation. ACM, 57–62.

Kononova, A. V., Ingham, D. B. & Pourkashanian, M. 2008. Simple scheduled memetic algorithm for inverse problems in higher dimensions: Application to chemical kinetics. In CEC 2008. Hong Kong: IEEE Press, 3906-3913.

Korošec, P., Šilc, J. & Filipič, B. 2011. The differential ant-stigmergy algorithm. Information Sciences. to appear.

Krasnogor, N. & Smith, J. 2005. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. IEEE Transactions on Evolutionary Computation 9, 474–488.

Krink, T. & Fogel, G. 2004. Noisy optimization problems - a particular challenge for differential evolution ? In Proceedings of the IEEE Congress on Evolutionary Computation, 332–339.

Krohling, R. A. & Rey, J. P. 2001. Design of optimal disturbance rejection pid controllers using genetic algorithms. IEEE Transactions On Evolutionary Computation 5, 78-82.

Kärkkäinen, T. & Heikkola, E. 2004. Robust formulations for training multilayer perceptrons. Neural Computation, MIT 16, 837–862.

Larrañaga, P. & Lozano, J. A. 2001. Estimation of Distribution Algorithms:A New Tool for Evolutionary Computation. Kluwer.

Le, M. N., Ong, Y. S., Jin, Y. & Sendhoff, B. 2009. Lamarckian memetic algorithms: local optimum and connectivity structure analysis. Memetic Computing Journal 1 (3), 175–190.

Lee, C. O., Jeon, Y. & Hong, Y.-S. 1998. Optimization of the control parameters of a pneu- matic servo cylinder drive using genetic algorithms. Control Engineering Practice 6, 847-853.

Leng, S. B., Sim, Y. C. & Subramaniam, V. 2000. A combined genetic algorithms-shooting method approach to solving optimal control problems. International Journal on Systems and Science 31, 83-89.

Marrison, C. H. & Stengel, R. F. 1997. Robust control system design using random search and genetic algorithms. IEEE Transactions On Automatic Controls 42, 835-839.

Meuth, R., Lim, M. H., Ong, Y. S. & Wunsch-II, D. C. 2009. A proposition on memes and meta-memes in computing for higher-order learning. Memetic Computing Journal 1 (2), 85–100.

Michalewicz, Z. & Schoenauer, M. 1996. Evolutionary algorithms for constrained parameter optimization problems. Evolutionary Computation 4 (1), 1-32.

Michalewicz, Z. 1992. Genetic Algorithms + Data Structures = Evolution Programs. Springer.

Miller, B. L. & Goldberg, D. E. 1996. Genetic algorithms, selection schemes, and the varying effects of noise. Evolutionary Computation 4 (2), 113-131.

Mininno, E., Cupertino, F. & Naso, D. 2008. Real-valued compact genetic algorithms for embedded microcontroller optimization. IEEE Transactions On Evolutionary Computation 12, 347-365.

Moscato, P. & Norman, M. 1989. A Competitive and Cooperative Approach to Complex Combinatorial Search.

Moscato, P. 1989. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms.

Nelder, A. & Mead, R. 1965. A simplex method for function optimization. Computation Journal 7, 308-313.

Neri, F. & Cotta, C. 2011a. A primer on memetic algorithms. In F. Neri, C. Cotta & P. Moscato (Eds.) Handbook of Memetic Algorithms. Springer. Studies in Computational Intelligence. to appear.

Neri, F. & Cotta, C. 2011b. Memetic algorithms and memetic computing optimization: A literature review. Swarm and Evolutionary Computation. to appear.

Neri, F., Iacca, G. & Mininno, E. 2011. Disturbed exploitation compact differential evolution for limited memory optimization problems. Information Sciences 181 (12), 2469-2487.

Neri, F., Toivanen, J., Cascella, G. L. & Ong, Y. S. 2007. An adaptive multimeme algorithm for designing HIV multidrug therapies. IEEE/ACM Transactions on Computational Biology and Bioinformatics 4 (2), 264–278.

Neri, F., Cotta, C. & Moscato, P. 2011. Handbook of Memetic Algorithms, Vol. 379. Springer. Studies in Computational Intelligence.

Neri, F., Kononova, A. V. & Acciani, G. 2006. Prudent-daring vs tolerant survivor selection schemes in control design of electric drives. In Rothlauf, F. et al. (Ed.) Applications of Evolutionary Computing, Vol. 3907. Springer. Lecture Notes in Computer Science, 805–809.

Neri, F., Kotilainen, N. & Vapa, M. 2008. A memetic-neural approach to discover resources in P2P networks. In J. van Hemert & C. Cotta (Eds.) Recent Advances in Evolutionary Computation for Combinatorial Optimization. Springer. Studies in Computational Intelligence, 119–136.

Neri, F. & Mäkinen, R. 2007. Hierarchical evolutionary algorithms and noise compensation via adaptation. In S. Yang, Y. S. Ong & Y. Jin (Eds.) Evolutionary Computation in Dynamic and Uncertain Environments. Springer. Studies in Computational Intelligence, 345–369.

Neri, F., Tirronen, V., Kärkkäinen, T. & Rossi, T. 2007a. Fitness diversity based adaptation in multimeme algorithms: A comparative study. In Proceedings of the IEEE Congress on Evolutionary Computation, 2374–2381.

Neri, F., Toivanen, J. & Mäkinen, R. 2007b. An adaptive evolutionary algorithm with intelligent mutation local searchers for designing multidrug therapies for HIV. Applied Intelligence 27 (3), 219–235.

Neri, F., del Toro Garcia, X., Cascella, G. L. & Salvatore, N. 2008. Surrogate assisted local search on PMSM drive design. COMPEL: International Journal for Computation and Mathematics in Electrical and Electronic Engineering 27 (3), 573–592.

Neri, F., Cotta, C. & Moscato, P. 2011. Handbook of Memetic Algorithms. Springer. Studies in Computational Intelligence. to appear.

Nguyen, Q. C., Ong, Y. S. & Lim, M. H. 2009. A probabilistic memetic framework. IEEE Transactions on Evolutionary Computation 13 (3), 604–623.

Ong, Y. S. & Keane, A. J. 2004. Meta-lamarkian learning in memetic algorithms. IEEE Transactions on Evolutionary Computation 8 (2), 99–110.

Ong, Y. S., Lim, M. H., Zhu, N. & Wong, K. W. 2006. Classification of adaptive memetic algorithms: A comparative study. IEEE Transactions On Systems, Man and Cybernetics - Part B 36 (1), 141–152.

Ong, Y.-S., Lim, M.-H. & Chen, X. 2010. Memetic computation-past, present and future. IEEE Computational Intelligence Magazine 5 (2), 24–31.

Pan, H. & Wanga, L. 2006. Particle swarm optimization for function optimization in noisy environment. Applied Mathematics and Computation 181, 908–919.

Porter, B. & Jones, I. H. 1992. Genetic tuning of digital pid controllers. Electronic Letters 28, 843-844.

Rahnamayan, S., Tizhoosh, H. R. & Salama, M. M. 2006. Opposition-based differential evolution for optimization of noisy problems. In Proceedings of the IEEE Congress on Evolutionary Computation, 1865–1872.

Rudolph, G. 2001. A partial order approach to noisy fitness functions. In Proceedings of the IEEE Congress on Evolutionary Computation, 318–325.

Rudolph, G. 1999. Self-Adaptive Mutations May Lead to Premature Convergence. ⟨URL:citeseer.ist.psu.edu/article/rudolph99selfadaptive.html⟩.

Salvatore, L., Cupertino, F., Naso, D. & Turchiano, B. 2002. Design of cascaded controllers for dc drives using evolutionary algorithms. Proceedings of IEEE World Conference on Computational Intelligence.

Sano, Y., Kita, H., Kamihira, I. & Yamaguchi, M. 2000. Online optimization of an engine controller by means of a genetic algorithm using history of search. In Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning. Springer, 2929–2934.

Schwefel, H.-P. & Rudolph, G. 1995. Contemporary evolution strategies. Lecture Notes in Artificial Intelligence 929.

Schwefel, H.-P. 1981. Numerical Optimization of Computer Models. Wiley.

Smith, J. E. 2007. Coevolving memetic algorithms: A review and progress report. IEEE Transactions on Systems, Man, and Cybernetics, Part B 37 (1), 6-17.

Spall, J. C. 2000. Adaptive stochastic approximation by the simultaneous perturbation method. IEEE Transaction on Automatic Control 45, 1839-1853.

Spall, J. C. 2003. Introduction to stochastic search and optimization. Wiley.

Stagge, P. 1998. Averaging efficiently in the presence of noise. In Proceedings of the 5th International Conference on Parallel Problem Solving from Nature. Springer-Verlag, 188–200.

Storn, R. & Price, K. 1995. Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. ICSI.

Tang, J., Lim, M. H. & Ong, Y. S. 2007. Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems. Soft Computing-A Fusion of Foundations, Methodologies and Applications 11 (9), 873–888.

Tirronen, V., Neri, F., Kärkkäinen, T. & Rossi, T. 2008. An enhanced memetic differential evolution in filter design for defect detection in paper production. Evolutionary Computation 16, 529–555.

Wolpert, D. H. & Macready, W. G. 1997. No free lunch theorems for optimization. IEEE Transactions On Evolutionary Computation 1, 67-82.

Yu, E. L. & Suganthan, P. N. 2010. Ensemble of niching algorithms. Information Sciences 180 (15), 2815–2833.

Yu, J., Koza, J. R. & Keane, M. A. 1999. Automatic synthesis of both the topology and parameters for a robust controller for a nonminimal phase plant and a three-lag plant by means of genetic programming. Procediings 38th IEEE Int. Conference on Decision and Control, 5292-5300.

Yuan, Q., Qian, F. & Du, W. 2010. A hybrid genetic algorithm with the baldwin effect. Information Sciences 180 (5), 640–652.

Zinober, A. S. I., Moin, N. H. & Harley, P. J. 1995. Sliding mode control design using genetic algorithms. Proceedings on Genetic Algorithms Engineering Systems: Innovations, Applications, 238-244.

ORIGINAL PAPERS

PI

COMPACT DIFFERENTIAL EVOLUTION

by

E. Mininno, F. Neri, F. Cupertino, D. Naso 2011

IEEE Transactions on Evolutionary Computation, Vol. 15, No. 1

https://doi.org/10.1109/TEVC.2010.2058120

PII

MEMETIC COMPACT DIFFERENTIAL EVOLUTION

by

F. Neri and E. Mininno 2010

IEEE Computational Intelligence Magazine

https://doi.org/10.1109/MCI.2010.936305

PIII

ESTIMATION DISTRIBUTION DIFFERENTIAL EVOLUTION

by

E. Mininno and F. Neri 2010

EvoApplications, Part I

https://doi.org/10.1007/978-3-642-12239-2_54

PIV

A MEMETIC DIFFERENTIAL EVOLUTION APPROACH IN NOISY OPTIMIZATION

by

E. Mininno and F. Neri 2010

Memetic Computing

https://doi.org/10.1007/s12293-009-0029-4

Reading link: https://rdcu.be/b3X7F

PV

NOISE ANALYSIS COMPACT GENETIC ALGORITHM

by

E. Mininno and F. Neri 2010

EvoApplications, Part I

https://doi.org/10.1007/978-3-642-12239-2_62