EVOLUTIONARY OPTIMIZATION METHODS FOR ACCELERATOR DESIGN

By

Alexey A. Poklonskiy

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Physics and Astronomy
and
Mathematics

2009

ABSTRACT

EVOLUTIONARY OPTIMIZATION METHODS FOR ACCELERATOR DESIGN

By

Alexey A. Poklonskiy

Many problems from the fields of accelerator physics and beam theory can be formulated as optimization problems and, as such, solved using optimization methods. Despite growing efficiency of the optimization methods, the adoption of modern optimization techniques in these fields is rather limited. Evolutionary Algorithms (EAs) form a relatively new and actively developed optimization methods family. They possess many attractive features such as: ease of the implementation, modest requirements on the objective function, a good tolerance to noise, robustness, and the ability to perform a global search efficiently which make them the tool of choice for many design and optimization problems. In this work we study the application of EAs to problems from accelerator physics.

We review the most commonly used methods of unconstrained optimization and describe the GATool, evolutionary algorithm and the software package, used in this work. Then we use a set of test problems to assess its performance in terms of computational resources and the quality of the obtained result. We justify the choice of GATool as a heuristic method to generate cutoff values for the COSY-GO rigorous global optimization package. We design the model of their interaction and demonstrate that the quality of the result obtained by GATool increases as the information about the search domain is refined, supporting the usefulness of this model. We discuss GATool's performance on the problems with static and dynamic noise and study useful strategies of GATool parameter tuning for these and other difficult problems.

We review the challenges of constrained optimization with EAs and then describe REPA, a new constrained optimization method based on repairing, in exquisite detail, including the properties of its two repairing techniques: REFIND and REPROPT. We assess REPROPT's performance on the standard constrained optimization test problems for EA with and suggest optimal default parameter values based on the results. Then we study the performance of the REPA method on the same set of test problems and compare the obtained results with those of several commonly used constrained optimization methods with EA. Based on the obtained results, particularly on the outstanding performance of REPA on test problem that presents significant difficulty for other reviewed EAs, we conclude that the proposed method is useful and competitive. We discuss REPA parameter tuning for difficult problems and critically review some of the problems from the de-facto standard test problem set for the constrained optimization with EA.

We study several different problems of accelerator design and demonstrate how they can be solved with GATool. These problems include a simple accelerator design problem (design a quadrupole triplet to be stigmatically imaging, find all possible solutions), a complex real-life accelerator design problem (an optimization of the front end section for the future neutrino factory), and a problem of the normal form defect function optimization used to rigorously estimate the stability of the beam dynamics in circular accelerators. The positive results we obtained suggest that the application of EAs to problems from accelerator theory has large potential. The developed optimization scenarios and tools can be used to approach similar problems.

Dedicated to

life in all forms and appearances

and the evolution that drives it to perfection

ACKNOWLEDGMENTS

They say: “Everything that has a beginning has an end”. Now, as the very moment of inserting the final piece into the large puzzle called thesis has come, it is utterly satisfactory, and at the same time somewhat sad, to see this long-lasting endeavour coming to an end. The work that was done during all these years of graduate school has taken its final shape, is summarized, and written up in a logical sequence. The only thing it is waiting for is to be put to an ultimate examination by a strict yet fair committee of the established and renowned professionals, and thus meet its concluding test.

Even if your goal is not as big as the Universe, chances are that you do not have the power to reach it immediately. The road to fulfillment, assuming you venture to try, often consists of many small steps. Some of them would be easy to make, especially if you were in the right place at the right time, some would require more attention and efforts. Some would be so hard to make, that it would knock you off your feet and test your ability not to surrender to the circumstances, test your persistence, willpower and optimism. It would be hard not to lose your way and keep going, it would be hard not to dilute your ultimate goal in short-term accomplishments and everyday chores, but it is still possible. Even though there were all kind of steps on my road through the graduate school, here I am, seeing the end, several steps away from completion.

It is only when you get to your target, you can stop satisfied, look back and observe the whole trip, thinking of and thanking all the wonderful people that helped you along the way. So, first and foremost, I would like to thank all the people who were helping me throughout the Ph.D. program. Those who taught me, supported me, inspired me, served as living examples of various great personal qualities, cheered me up, and enlightened me in numerous friendly conversations. I want to thank all the people who will not get their personal words of gratitude. I do remember you and I am very grateful for what you have done for me. Thank you!

Undoubtedly, the most important person in the professional (and often personal) life of a graduate student is his or her scientific adviser. I consider myself very lucky to have several. All of them heavily influenced my professional development, the contents and style of this work, and my view of the world of science. I thank my main adviser, Dr. Martin Berz, Michigan State University, for his deep involvement with all of his students, including me, for his willingness to help, his deep understanding of scientific endeavors, and a great talent to explain even the most complicated topics such that they are comprehensible by a mere student. In addendum, I thank him for showing me so many great examples of the fundamental principles of the scientific research and ethics. I also thank Dr. Dmitriy Ovsyannikov and Dr. Alexandre Ovsyannikov, Saint-Petersburg State University, for inspiring me to join the graduate program and, together with Dr. Berz, for giving me a great opportunity to come and study here at Michigan State University. Another person who has continuously been working to make the international exchange programs available to Russian students is Dr. Victor Yarba, Fermilab. I am grateful to him for these efforts.

I thank two of my advisers "in the field", Dr. Carol Johnstone and Dr. David Neuffer, Fermilab, for demonstrating to me how the practical problems of the frontiers of science are being attacked and solved, and for teaching me to trust scientific intuition and insights as much as rigorous step-by-step derivations. Additional words of gratitude have to be said to Dr. Carol Johnstone for her endless help in my everyday life in the United States and for making the accommodation process much smoother than it could have been. I am also very grateful to Dr. Patricia Lamm, Mathematics Department, Michigan State University, for being such a great professor and role model of a great teacher. I thoroughly enjoyed her lectures, her creative application of the modern technologies in the classroom, and even her carefully thought homework assignments. I am grateful to Dr. Sheldon Newhouse, Mathematics Department, Michigan State University, for his enthusiasm and help with exams, to Dr. Keith Promislow and Dr. Huyi Hu, Mathematics Department, Michigan State University for their support, and to all other members of my guidance committee: Dr. S.D. Mahanti, Dr. Joey Huston and Dr. Kyoko Makino, Physics & Astonomy Department, Michigan State University, for their help. Personal kudos to Dr. Kyoko Makino for her very extensive proofreading of this work and valuable technical corrections.

I separately thank my colleagues from the Beam Theory group: Dr. Kyoko Makino, Dr. Shashikant Manikonda, Johannes Grote, Youn-Kyung Kim, Alex Wittig and, of course, my comrade Dr. Pavel Snopok for sharing my professional and personal life in all their ups and downs for almost the entire period of my graduate studies. I will always remember our scientific discussions, small talks about science, conference travels, dinners, and parties. I am grateful for the help in any circumstances and the pleasure that I had, in working with all these wonderful people.

A great portion of my gratitude belongs to my mother, Larisa Poklonskaya, and my father, Alexander Poklonskiy, for raising me, for being my first teachers of everything, for giving me this infinite supply of the curiosity that drives me through the life, and for their unconditional love and support. Many thanks go to my friend, Alyona Myasnikova, for being there whenever I needed her and for her encouragement and inspiration. I would also like to thank Dmytro Berbasov and Oksana Odnovol for being understanding, forgiving, patient, and cheerful friends and roommates, especially during the thesis writing phase when I would rarely get off the work and did not bother about “unimportant” things like dirty dishes or general mess. Personal thanks to Oksana for proofreading parts of this work. Additional thanks and deep gratitude to my colleagues from Microsoft who helped me to finish up this work by fixing and polishing the language: Kristen Lovin, Bill Donkervoet, and Richard Zadorozny.

I thank Debbie Simmons and Brenda Wenzlick, secretaries of the Physics & Astronomy Department, Michigan State University, for their good work and helpfulness.

Thanks to Fermi National Accelerator Laboratory for financial support which is, of course, very important for scientific success.

Finally, I would like to thank you for reading this work and myself for actually writing it.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Beam and Accelerator Theory

1.1.1 Differential Algebra and Map Methods

The dynamics of the various objects in Physics are often described by a system of the nonlinear ordinary differential equations

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t), \tag{1.1.1}$$

where $\mathbf{x}$ is a vector of coordinates of the considered object, t is time, and $\mathbf{f}$ is a nonlinear vector function that describes various forces acting on the object and thus governing the dynamics. Initial conditions

$$\mathbf{x}(0) = \mathbf{x}_{\mathrm{i}} \tag{1.1.2}$$

specify the initial position of the object, i.e. its position at the moment of time that is considered initial. It is often advantageous to describe the action of the system (1.1.1) with a so-called flow operator, $\mathcal{M}_T$, which establishes a mapping between the initial position $\mathbf{x}_{\mathrm{i}}$ of the object at $t = 0$ and its final position $\mathbf{x}_{\mathrm{f}}$ that the object assumes at time T:

$$\mathbf{x}_\mathrm{f} = \mathcal{M}_T(\mathbf{x}_\mathrm{i}). \tag{1.1.3}$$

The flow operator approach is especially useful for studying various properties of the dynamics in systems that are periodic in t. Since it captures the essential properties of the dynamics in the system, it is possible to assess the properties of the flow $\mathcal{M}_T$ instead of the dynamics of individual objects with varying initial conditions. A good example of such system is a circular particle accelerator.

The problem here is that even in the cases of relatively simple functions $\mathbf{f}$ it is frequently not possible to determine the system map in a closed form, so for the practical purposes $\mathcal{M}_T$ is often calculated via numerical integration of the equations (1.1.1). However, if the function $\mathbf{f}$ is only weakly nonlinear, i.e. if its behaviour is mostly determined by the linear component, then its map is also only weakly nonlinear and thus can be represented as a Taylor expansion with practically acceptable precision. Developments in the field of Differential Algebra (DA) and its applications to Automatic Differentiation have have opened the possibility to compute the Taylor series for maps of such systems to an arbitrary high order. A detailed treatment of the Differential Algebra framework and its numerous applications including map methods for Accelerator Physics can be found in [18].

Particle accelerators typically consist of numerous subsystems influencing different aspects of particle dynamics. The original method of map calculation, which involves propagation of functional dependencies through a numeric solver of differential equations using automatic differentiation technique, is slow and imprecise. Computation of the flow for individual devices and then application of the composition property to obtain the flow of the whole accelerator can be performed within the Differential Algebra framework quite efficiently and with unlimited precision.

The law of map composition tells us that if we have two maps: $\mathcal{M}_{t_0,t_1}$ that relates the initial position at the t_0 to the final position at time t_1 and $\mathcal{M}_{t_1,t_2}$ that relates the initial position at time t_1 to the final position at time t_2 then the map that relates the initial position at time t_0 to the final position at time t_2 can be constructed via a map composition:

$$\mathcal{M}_{t_0,t_2} = \mathcal{M}_{t_1,t_2} \circ \mathcal{M}_{t_0,t_1}. \tag{1.1.4}$$

Using this property we can assemble the transfer map for the entire accelerator if we have transfer maps of all its elements which we can compute using DA.

1.1.2 Beam Dynamics

Particles in accelerators are rarely studied as standalone objects. Usually ensembles of particles that have similar coordinates are used. These ensembles are called particle beams. Since the particles in a beam are separated from each other by a relatively small distance, it is often convenient to select one imaginary particle that represents the motion of the whole beam inside the accelerator and then describe the motion of other particles in the beam in the coordinates that are relative to those of the reference particle [18, 89, 170, 175].

In the laboratory coordinate system the particle state is usually represented by a vector that consists of its space coordinates and the components of its momentum vector corresponding to the coordinate axes. Time usually serves as an independent variable:

$$\mathbf{z}(t) = (x, p_x, y, p_y, z, p_z)^{\mathrm{T}}. \tag{1.1.5}$$

In the curvilinear coordinate system that is attached to a reference particle the arclength along the reference trajectory is usually serves as an independent variable. In this coordinate system (often called the curvilinear coordinate system) the particle state is represented by the following coordinates:

$$\mathbf{z}(s) = \begin{pmatrix} x \\ a = p_x/p_0 \\ y \\ b = p_y/p_0 \\ l = k(t - t_0) \\ \delta = (E - E_0)/E_0 \end{pmatrix}, \tag{1.1.6}$$

where the x, y denote the position of the particle in this relative coordinate system, p_0 is an arbitrary fixed momentum (usually the one of the reference particle), E_0 and t_0 are the energy and the time of flight of the reference particle, E is a total energy of the particle, and k is a scaling coefficient that transforms time coordinate to space-like coordinate. In those coordinates the reference particle corresponds to a $\mathbf{z} = \mathbf{0}$.

The motion of a particle in the electromagnetic field is governed by the Lorentz force [93]:

$$\frac{d\mathbf{p}}{dt} = q\left(\mathbf{E} + \mathbf{v} \times \mathbf{B}\right). \tag{1.1.7}$$

In order to study the motion of the particles that form the beam in the curvilinear coordinates, those equations of the form (1.1.5) in the laboratory coordinate system, are transformed into the curvilinear coordinate system, for the special case when the reference trajectory is restricted to a plane (which is the case for most particle accelerators). Applying these transformations and using the reference trajectory simplifying assumption they can be brought to the following form:

$$x' = a(1+hx)\frac{p_0}{p_s} \tag{1.1.8}$$

$$y' = b(1+hx)\frac{p_0}{p_s} \tag{1.1.9}$$

$$a' = \left(\frac{1+\eta}{1+\eta_0}\frac{p_0}{p_s}\frac{E_x}{\chi_{e0}} + b\frac{B_z}{\chi_{m0}}\frac{p_0}{p_s} - \frac{B_y}{\chi_{m0}}\right)(1+hx) + h\frac{p_0}{p_s} \tag{1.1.10}$$

$$b' = \left(\frac{1+\eta}{1+\eta_0}\frac{p_0}{p_s}\frac{E_y}{\chi_{e0}} + \frac{B_x}{\chi_{m0}} - a\frac{B_z}{\chi_{m0}}\frac{p_0}{p_s}\right)(1+hx) \tag{1.1.11}$$

$$l' = \left((1+hx)\frac{1+\eta}{1+\eta_0}\frac{p_0}{p_s} - 1\right)\frac{k}{v_0} \tag{1.1.12}$$

$$\delta' = 0 \tag{1.1.13}$$

where $'$ is a derivative with respect to the arclength s, h is a radius of the curvature of the reference trajectory,

$$\frac{p_0}{p_s} = \left(\frac{\eta(2+\eta)}{\eta_0(2+\eta_0)}\frac{m^2}{m_0^2} - a^2 - b^2\right)^{-1/2},$$

$$\eta = \frac{E - eV(x,y,s)}{mc^2},$$

$$\chi_{m0} = \frac{p_0}{ze}$$

is the magnetic rigidity,

$$\chi_{e0} = \frac{p_0 v_0}{ze}$$

is the electric rigidity, B_x, B_y, B_z and E_x, E_y, E_z are x, y and z components of the magnetic and electric field in the laboratory coordinate system, correspondingly. A rigorous definition of the coordinate system and the detailed derivation of the equations of motion in this system (1.1.8)–(1.1.13) can be found in [18].

Once the fields and the reference trajectory are known, these equations can be directly integrated (analytically for simplest cases, numerically for most real-life problems) in order to determine the dynamics of the particles. More efficiently, map methods, mentioned earlier, can be used for this purpose. The latter approach is used by the Beam Physics package for *COSY Infinity* scientific computing code [22, 23]. In this framework the Taylor expansion of the map is actually an array of Taylor expansions of the dependencies of the final coordinates on the initial coordinates. Employing notation that is frequently used in optics in order to emphasize the nature of the coordinate dependencies, it can be written in the following form:

$$\begin{aligned} x_{\mathrm{f}} &= (x|x)x_{\mathrm{i}} + (x|a)a_{\mathrm{i}} + (x|y)y_{\mathrm{i}} + (x|b)b_{\mathrm{i}} + (x|l)l_{\mathrm{i}} + (x|\delta)\delta_{\mathrm{i}} \\ &+ (x|xx)x_{\mathrm{i}}^2 + (x|xa)x_{\mathrm{i}}a_{\mathrm{i}} + (x|xy)x_{\mathrm{i}}y_{\mathrm{i}} + (x|xb)x_{\mathrm{i}}b_{\mathrm{i}} + \ldots. \end{aligned}$$

In this notation the Taylor expansion for the map of the system (1.1.8)–(1.1.13) takes the form:

$$x_{\mathrm{f}} = \sum (x|x^{i_1}a^{i_2}y^{i_3}b^{i_4}l^{i_5}\delta^{i_6}) x_{\mathrm{i}}^{i_1}a_{\mathrm{i}}^{i_2}y_{\mathrm{i}}^{i_3}b_{\mathrm{i}}^{i_4}l_{\mathrm{i}}^{i_5}\delta^{i_6} \tag{1.1.14}$$

$$a_{\mathrm{f}} = \sum (a|x^{i_1}a^{i_2}y^{i_3}b^{i_4}l^{i_5}\delta^{i_6}) x_{\mathrm{i}}^{i_1}a_{\mathrm{i}}^{i_2}y_{\mathrm{i}}^{i_3}b_{\mathrm{i}}^{i_4}l_{\mathrm{i}}^{i_5}\delta^{i_6} \tag{1.1.15}$$

$$y_{\mathrm{f}} = \sum (y|x^{i_1}a^{i_2}y^{i_3}b^{i_4}l^{i_5}\delta^{i_6}) x_{\mathrm{i}}^{i_1}a_{\mathrm{i}}^{i_2}y_{\mathrm{i}}^{i_3}b_{\mathrm{i}}^{i_4}l_{\mathrm{i}}^{i_5}\delta^{i_6} \tag{1.1.16}$$

$$b_{\mathrm{f}} = \sum (b|x^{i_1}a^{i_2}y^{i_3}b^{i_4}l^{i_5}\delta^{i_6}) x_{\mathrm{i}}^{i_1}a_{\mathrm{i}}^{i_2}y_{\mathrm{i}}^{i_3}b_{\mathrm{i}}^{i_4}l_{\mathrm{i}}^{i_5}\delta^{i_6} \tag{1.1.17}$$

$$l_{\mathrm{f}} = \sum (l|x^{i_1}a^{i_2}y^{i_3}b^{i_4}l^{i_5}\delta^{i_6}) x_{\mathrm{i}}^{i_1}a_{\mathrm{i}}^{i_2}y_{\mathrm{i}}^{i_3}b_{\mathrm{i}}^{i_4}l_{\mathrm{i}}^{i_5}\delta^{i_6} \tag{1.1.18}$$

$$a_{\mathrm{f}} = \sum (a|x^{i_1}a^{i_2}y^{i_3}\delta^{i_4}l^{i_5}\delta^{i_6}) \delta_{\mathrm{i}}^{i_1}a_{\mathrm{i}}^{i_2}y_{\mathrm{i}}^{i_3}b_{\mathrm{i}}^{i_4}l_{\mathrm{i}}^{i_5}\delta^{i_6} \tag{1.1.19}$$

$$\tag{1.1.20}$$

where the summation is performed on all indices $i_1, i_2, \ldots, i_6$ such that $\sum_{k=1}^{6} i_k \leq n$, n is the Taylor expansion order.

Normal Form Methods

Repetitive systems such as synchrotrons and storage rings are the main component of the most modern high-energy particle accelerators. In those circular lattices, particles ought to remain confined for many turns. Hence their trajectories should be stable, which usually requires them to be bounded in some way. The study of the dynamics of particles in these structures and the stability of the dynamics is very important both theoretically and practically. The advantages of map methods for such studies lie in the ability to calculate a map of motion (see section 1.1.1) representing the action of all accelerator elements in one full revolution. Then the repeated application of this map, the so called Poincaré map, can be studied to evaluate the stability of the entire device for large number of turns.

The linear theory of repeated motion has been fully developed since its introduction by Courant and Snyder [49]. It relies on the well-known matrix methods from Linear Algebra (see [17] for detailed treatment). Since the transfer map to first order is a matrix, the so called transfer matrix, the stability of the motion is determined by its eigenvalues: if any of the eigenvalues has absolute value > 1, then the motion is unstable. Since the motion in such structure is volume preserving, the product of the eigenvalues of the transfer matrix must be one. This means that if there exists an eigenvalue > 1, there must be another inversely proportional to it and thus < 1. However, such an arrangement would make the motion unstable. Therefore for the motion to be stable and volume preserving, all eigenvalues must have a magnitude of one. However, real-valued eigenvalues that have the magnitude of one can be perturbed from this value under a small perturbation in the system parameters rather easily, hence they should be complex. In sum, for our motion to be stable we need the system's linear transfer matrix to have only complex conjugate eigenvalues with magnitudes of one. Further development of this theory and other conditions imposed on the matrix and its eigenvalues by stability considerations can be found in any of the sources mentioned earlier.

Nonlinear motion is in general much more difficult to study. In accelerator theory nonlinear studies are usually divided into the study of the parameter-dependent linear motion, where parameters include the particle energy spread, magnet misalignments, etc., with perturbation theory; and the study of fully nonlinear dynamics. Many useful properties of nonlinear motion can be obtained exactly by employing the method of normal forms, first introduced (to low orders, no more than 3) by Dragt in 1979 [54] and developed over almost two decades, then brought to its full practical power (maximum order is theoretically infinite, i.e. limited only by the available computational resources) by Berz in 1992 within the Differential Algebra framework [16, 17]. This approach provides an algorithm to build a nonlinear change of variables to remove all removable non-linearities and present motion in the set of variables where it is circular with amplitude-dependent frequency.

Assume, that we obtained the nonlinear transfer map of a particle optical system under consideration:

$$\mathbf{z}_f = \mathcal{M}(\mathbf{z}_i, \delta\,), \tag{1.1.21}$$

where $\mathbf{z}$ is the $2v$-dimensional vector of the phase space coordinates, δ is the vector of the system parameters and the indices i and f correspond to initial and final coordinates. We want to build a sequence of the coordinate transformations $\mathcal{A}$ of the map

$$\mathcal{A} \circ \mathcal{M} \circ \mathcal{A}^{-1} \tag{1.1.22}$$

to remove all nonlinearities of every order up to the desired.

The first transformation is performed in order to make the map origin-preserving for any δ:

$$\mathcal{M}(\mathbf{0}, \delta) = \mathbf{0}.$$

DA methods are employed to move the map to the new parameter-dependent fixed point $\mathbf{z}_\mathrm{F}$ so that

$$\mathbf{z}_\mathrm{F} = \mathcal{M}(\mathbf{z}_\mathrm{F}, \delta\,). \tag{1.1.23}$$

It is possible if and only if 1 is not an eigenvalue of the linear part of the map. For stable repetitive systems such a condition always holds as we mentioned earlier.

The diagonalization of the linear part of the map, so called linear diagonalization, is performed on the next step. From Linear Algebra we know that in this case diagonalization is possible if the matrix has exactly $2v$ distinct eigenvalues, which is true for most modern circular accelerators. In this case it is possible to represent all eigenvalues as complex conjugate pairs $r_j e^{\pm i\mu_j}$. It is easy to show that for symplectic systems [17], the condition for the determinant to be unity entails

$$r_j = 1, \quad \mu_j \in \mathbb{R}, \quad \text{for } j = 1, \ldots, v.$$

If we now transfer the matrix to the new basis of complex conjugate eigenvectors $\mathbf{v}_j^\pm$ corresponding to complex conjugate eigenvalues, it assumes the diagonal form

$$\mathcal{R} = \begin{pmatrix} r_1 e^{+i\mu_1} & 0 & \ldots & 0 & 0 \\ 0 & r_1 e^{-i\mu_1} & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & r_v e^{+i\mu_v} & 0 \\ 0 & 0 & \ldots & 0 & r_v e^{-i\mu_v} \end{pmatrix}. \tag{1.1.24}$$

On subsequent steps we iteratively build a sequence of non-linear transformations of the form (1.1.22), such that on each step the constructed transformation tries to remove one particular order of nonlinearity. The ultimate goal is to remove all nonlinearities up to a specified order but, as it turns out, this is not always possible. Nonlinearities that can be removed by means of this transformation are called removable, all other nonlinearities are called non-removable. Non-removable nonlinearities usually characterize the non-linear nature of the system under consideration.

It is worth noting that all these transformations are nonlinear and thus they do not affect the diagonal form of the linear map obtained in the first two steps. Since the process is iterative, it is sufficient to describe the algorithm to make the m-th step in order to fully determine it. Having one step of the algorithm we can proceed in applying it from order 2 to the desired order.

On the m-th step we try to remove nonlinearities of the order m only. In order to achieve it, we start by splitting the map $\mathcal{M}$ into a linear part $\mathcal{R}$ and a nonlinear part $\mathcal{S}_m$: $\mathcal{M} = \mathcal{R} + \mathcal{S}_m$. Then we perform a transformation using a map that to the m-th order has the form

$$\mathcal{A}_m = \mathcal{I} + \mathcal{T}_m, \tag{1.1.25}$$

where $\mathcal{I}$ is a linear unity map, $\mathcal{T}_m$ has only zero terms up to order $(m-1)$. The linear part of the transformation is a unity matrix that is invertible, hence the map $\mathcal{A}_m$ itself is invertible. Using relations for transfer map inversion from [17], we obtain the inverse to order m:

$$\mathcal{A}_m^{-1} =_m \mathcal{I} - \mathcal{T}_m. \tag{1.1.26}$$

Applying the transformation from (1.1.22), we obtain

$$\begin{aligned} \mathcal{A}_m \circ \mathcal{M} \circ \mathcal{A}_m^{-1} &=_m (\mathcal{I} + \mathcal{T}_m) \circ (\mathcal{R} + \mathcal{S}_m) \circ (\mathcal{I} - \mathcal{T}_m) \\ &=_m (\mathcal{I} + \mathcal{T}_m) \circ (\mathcal{R} + \mathcal{S}_m - \mathcal{R} \circ \mathcal{T}_m), \\ &=_m \mathcal{R} + \mathcal{S}_m + (\mathcal{T}_m \circ \mathcal{R} - \mathcal{R} \circ \mathcal{T}_m) \end{aligned} \tag{1.1.27}$$

where we used the fact that any nonlinear map composed with $\mathcal{T}_m$ is zero to the order m since $\mathcal{T}_m$ is of the order m and does not have smaller-order terms. If we now could choose $\mathcal{T}_m$ so that for communicator

$$\mathcal{C}_m = \{\mathcal{T}_m, \mathcal{R}\} = (\mathcal{T}_m \circ \mathcal{R} - \mathcal{R} \circ \mathcal{T}_m) \tag{1.1.28}$$

the following condition holds

$$\mathcal{C}_m = -\mathcal{S}_m, \tag{1.1.29}$$

then result of the transformation in (1.1.27) can be simplified to

$$\mathcal{A}_m \circ \mathcal{M} \circ \mathcal{A}_m^{-1} =_m \mathcal{R} \tag{1.1.30}$$

and the transformation $\mathcal{A}_m$ defined by (1.1.25) removes all nonlinearities of the map up to the order m. However, such choice of $\mathcal{T}_m$ is usually not possible.

In order to find the conditions for its existence, we consider the Taylor expansion of the $\mathcal{T}_m$ in the coordinates $s_j^{\pm}$ in the eigenvector basis $\mathbf{v}_j^{\pm}$. The Taylor expansion for the j-th component of the $\mathcal{T}_m$, has the form

$$\mathcal{T}_{mj}^{\pm} = \sum \left(\mathcal{T}_{mj}^{\pm} | \mathbf{k}^+, \mathbf{k}^-\right) \cdot \left(s_1^+\right)^{k_1^+} \left(s_1^-\right)^{k_1^-} \ldots \left(s_v^+\right)^{k_v^+} \left(s_v^-\right)^{k_v^-}, \tag{1.1.31}$$

where

$$\left(\mathcal{T}_{mj}^{\pm} | \mathbf{k}^+, \mathbf{k}^-\right)$$

are Taylor expansion coefficients for corresponding exponents of $s_j^{\pm}$; $\mathbf{k}^+$ and $\mathbf{k}^-$ are vectors of exponents

$$\begin{aligned} \mathbf{k}^+ &= (k_1^+, \ldots, k_v^+) \\ \mathbf{k}^- &= (k_1^-, \ldots, k_v^-). \end{aligned}$$

Now if we, using the same notation for Taylor expansion of $\mathcal{C}_m$, substitute relations (1.1.31) for $\mathcal{T}_m$ and the exact expression (1.1.24) for the $\mathcal{R}$ into the definition of the communicator (1.1.28), and then use the fact that polynomials are equal if the corresponding coefficients are equal to equate coefficients of the corresponding exponents in Taylor polynomials, we obtain the expression for the Taylor expansion coefficients of $\mathcal{C}_m$ components:

$$\left(\mathcal{C}_{mj}^{\pm}|\mathbf{k}^+,\mathbf{k}^-\right) = \left(\left(\prod_{l=1}^{v} r_l^{k_l^+ + k_l^-}\right) \cdot e^{i\mu(\mathbf{k}^+ - \mathbf{k}^-)} - r_j \cdot e^{\pm i\mu_j}\right) \cdot \left(\mathcal{T}_{mj}^{\pm}|\mathbf{k}^+,\mathbf{k}^-\right). \tag{1.1.32}$$

Substituting this expression into the condition (1.1.29) and solving the resulting equation for the coefficients of the Taylor expansion (1.1.31) of $\mathcal{T}_m$, we obtain:

$$\left(\mathcal{T}_{mj}^{\pm}|\mathbf{k}^+,\mathbf{k}^-\right) = \frac{-\left(\mathcal{S}_{mj}^{\pm}|\mathbf{k}^+,\mathbf{k}^-\right)}{\left(\left(\prod_{l=1}^{v} r_l^{k_l^+ + k_l^-}\right) \cdot e^{i\mu(\mathbf{k}^+ - \mathbf{k}^-)} - r_j \cdot e^{\pm i\mu_j}\right)}. \tag{1.1.33}$$

Now we see that the existence of the transformation (1.1.30) depends on conditions for which the expression in the denominator in the formula (1.1.33) is not zero. If it is zero for certain values of ($\mathbf{k}^+$, $\mathbf{k}^-$), then the corresponding Taylor expansion term of $\mathcal{S}_m$ cannot be removed. Some special cases, such as symplectic systems, which often arise in accelerator physics, as well as quantities of interest (in particular resonances of different kinds) that can be obtained from the normal form transformation, are discussed in details in [17].

Normal Form Defect Function

Normal forms (see [17], section 1.1.2) are a valuable part of the map methods in Differential Algebra (DA) framework and a powerful tool in studying the dynamics of the particles in circular accelerators. As was mentioned, the motion in these coordinates follows nearly perfect circles around a fixed point (Fig. 1.1).

If the motion has perfectly circular nature, it entails constancy of the radii that are thus invariants of motion. This, in turn, demonstrates the stability of the motion for the infinite time and number of turns. If the motion is non-perfectly circular, a measure of the defect in the invariants of the motion I is:

$$d = \max(I(\mathcal{M}) - I), \tag{1.1.34}$$

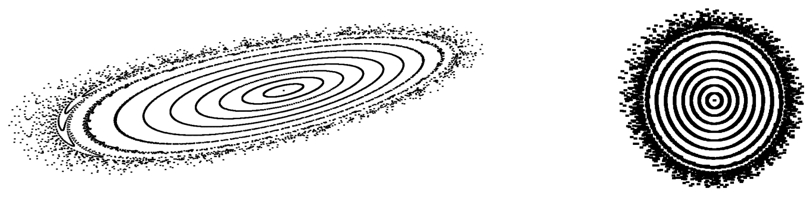


Figure 1.1: *Phase space trajectories in FODO cell, obtained for 1000 turns by applying one turn map to the vector with initial coordinates 1000 times; in conventional (left) and normal form (right) coordinates*

where $\mathcal{M}$ is the Poincare map (see section 1.1.2), and can be introduced using Nekhoroshev-type estimates [140]. It then could be studied to estimate the time and the number of turns that particles stay in the accelerator and make assertions on motion stability [16, 18, 20]. Note, however, that the presented approach to calculation of the invariants of motion involves Taylor expansions of maps to a specified order, hence it allows one to obtain the expansions of the invariant radii to that order only. This means that with this method we can obtain only approximate invariants from (1.1.34). Nevertheless, the quality of the approximation for the weakly nonlinear systems that particle accelerators are in most cases, improves rapidly with the order of the approximation order.

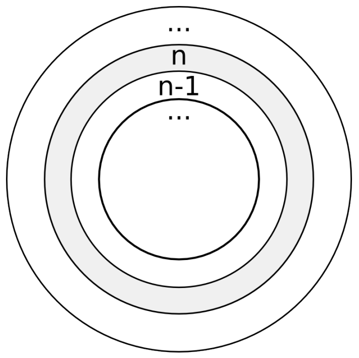
Suppose all trajectories in normal form coordinates are perfect circles. Then we know we have found an invariant of the system for all degrees of freedom. Now, if the transformation from normal form coordinates to conventional coordinates is continuous, then the set of trajectories is bounded and the motion is stable for an infinite time. For most systems under consideration this, however, is not the case. One reason for this is that the normal form defects can be very small and, being calculated on a computer, can be caused by floating point operations errors. Another reason is that the particle dynamics is calculated by means of Taylor expansions up to a specified order, so the invariants we obtain are only approximate. Here the defects of this approximation (decreasing with increasing order) produce the deviation from circularity. A third reason that many systems are non-integrable, i.e. they do not have an invariant for every degree of freedom. In this case the motion is non-circular even if the dynamics are calculated exactly, with no approximations or numerical errors. The non-integrability of the system indicates itself in the form of small denominators in (1.1.33) in some step of the normal form transformation algorithm applied to its map (see section 1.1.2). The circularity of the motion in the Figure 1.1 is disturbed for all three reasons but the non-integrability of the system under consideration most likely has the largest impact.

Any real physical system has some construction defects and real values of parameters can deviate from designed. Rigorous estimations of the stability ranges for perturbed motion exist, but stability predictions are possible for only for very small perturbations and totally dominated by realistic construction errors. While the defective nature of the invariants of motion prevents us from making statements on global stability for an infinite period of time, it is still possible to estimate stability for a finite, but still practically useful period of time, applying principles established by Nekhoroshev [140].

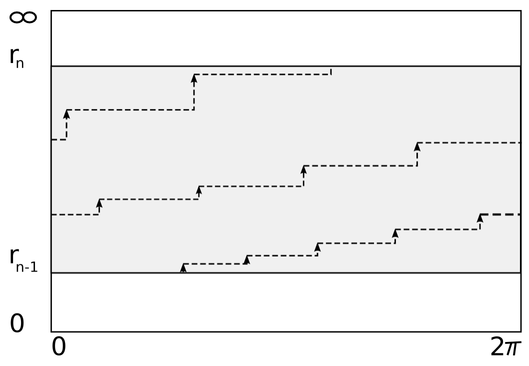
In order to do so, we divide the normal form coordinate space for each degree of freedom into a set of rings such that in each of them motion is almost circular, as demonstrated in Figure 1.2(a). Suppose that for the ring n the defect is not larger than Δr_n. Then all particles launched from ring $(n-1)$ need to make at least

$$N_n = \frac{r_n - r_{n-1}}{\Delta r_n} \tag{1.1.35}$$

turns before they reach the n-th ring (see Figure 1.2(b)). If we want to estimate the minimal number of turns it would take particles to get from the inner circle bounded by r_{min} (initial region) to the outer ring bounded by r_{max} (restricted region, particles



(a) Normal form coordinate space divided into a set of rings where we estimate the maximum defect



(b) Particles motion in the ring: arrows point at particle positions, step corresponds to one turn, height of the step corresponds to the defect, maximum height corresponds to Δr_i, number of steps is the number of turns N_i particle stay in this ring. Note that the defect gets larger towards the outer radii.

Figure 1.2: *Normal form coordinate space divided into rings and schematic view of particles motion in one of those rings*

that have reached it are considered lost), we can perform the subdivision

$$r_{\min} = r_1 < r_2 < \cdots < r_n = r_{\max}.$$

Then if maximal defects on each of the rings bounded by those radii are Δr_i, $i = 2, \ldots, n$ correspondingly, the total minimal number of turns, particles need to get to the restricted region from the initial region, is given by

$$N = \sum_{i=2}^{n} \frac{r_i - r_{i-1}}{\Delta r_i}. \tag{1.1.36}$$

Usually the normal form defect function grows quickly with radii (as can be seen, for example, in Figure 4.7), hence large values of n help us to get a better estimate of N. Since in most cases of interest the Δr_i are small, motion stability can be assured for a large number of turns.

The practical usefulness of this method heavily depends on the ability to determine tight and rigorous bounds for the defects of Δr_i. In practice, defect functions (Figs. 4.7, 4.11) are multi-dimensional polynomials of high order, with many of the high-order elements canceling each other out. Thus they pose difficulties for conventional interval methods. Studies on obtaining rigorous bounds for the maxima of normal form defect functions [20,26,126] have lead to many interesting numerical algorithms applicable to much wider class of problems (see section 2.2.2 for description of a rigorous global optimizer based on the DA framework and Taylor Models [118], section 4.2 on its application to the normal form defect function). The behaviour of these functions is highly oscillatory, the number of local extrema is high so they also present significant difficulty for conventional minimization methods. These properties are employed in section 4.2 to test the GATool genetic algorithm-based heuristic optimizer described in section 2.3 and in appendix B.

1.2 Neutrino Factories

1.2.1 Purpose and History

A neutrino is a special kind of elementary particle that was believed not to have any mass, charge or color. Recent studies, however, have demonstrated that neutrinos have a very small (estimated to be much less than 1 MeV), but non-zero mass. They are the most abundant constituent of the universe and have an important impact on astrophysical processes, from the first minutes after the Big Bang itself to supernovae explosions observed today. Neutrinos are created as a result of certain types of radioactive decay or nuclear reactions such as those that take place in the sun, in nuclear reactors, or when cosmic rays hit atoms. There are three types, or "flavors", of neutrinos, named after their partner leptons in the Standard Model: electron neutrinos ν_e, muon neutrinos ν_μ and tau neutrinos ν_τ. Each of those types also has an antimatter partner, called an antineutrino ($\bar{\nu}_e$, $\bar{\nu}_\mu$, $\bar{\nu}_\tau$ — electron, muon and tau antineutrinos, correspondingly). In 1998, experiments began to show that solar and atmospheric neutrinos change flavors. The processes leading to these unexpected masses and mixing parameters are suggested to take place at energies never seen since the Big Bang, perhaps connected to the unification of all forces. Precise determinations of the masses and mixing angles of the three families of neutrinos opens a unique window of observation into these early times [63]. These fascinating questions of physics require an ambitious accelerator-based neutrino experimental program [1,2].

The Neutrino Factory is a very important facility for the long-term neutrino physics program. Modern technologies of particle accelerators, both already developed and being researched, open the possibility of building an accelerator complex to produce and capture more than 10^{20} muons per year. The idea of an accelerator where the pions are injected into a storage ring, decay to produce muons captured within the same ring, and then further decay into a neutrino beam was proposed several times by different researchers starting from Koshkarev in 1974, but it has the basic problem that the resulting neutrino beam intensity was low [39,106]. The Neutrino Factory idea in its current form was proposed by Geer in 1997 [75]. He suggested creating muons from an intense pion source at low energies, then compressing their phase space to produce a bright beam which is then accelerated to the energies of several tens of GeV and injected into a storage ring with long straight sections where they decay into highly intense neutrino beams

$$\mu^- \to e^- \nu_\mu \bar{\nu}_e \, , \quad \mu^+ \to e^+ \bar{\nu}_\mu \nu_e. \tag{1.2.1}$$

Beams of such brightness can be used for the extensive study of the neutrino oscillations [5] and neutrino interactions with the required high precision.

In the U.S., the Neutrino Factory and Muon Collider Collaboration [146] is a collaboration of 130 scientists and engineers engaged in carrying out the accelerator R&D that is needed before a Neutrino Factory can be actually built. Much technical progress has been made over the last few years, and the required key accelerator experiments are now in the process of being proposed and approved. In addition to the U.S. effort, there are active Neutrino Factory R&D groups in Europe and Japan, and much of the R&D is performed and organized as an international endeavor. Neutrino Factory R&D is an important part of the present global neutrino program. The Neutrino Factory requires an intense multi-GeV proton source capable of producing a primary proton beam with a beam power of 2 MW or more on the target. This is the same proton source required in the near future for Neutrino Superbeams [33]. Therefore, there is a natural evolution from Superbeam experiments to Neutrino Factory experiments over time. Studies performed so far have shown that the Neutrino Factory gives the best performance among all considered neutrino sources over virtually all of the parameter space. Its practical possibility and cost remain, however, important questions that are being actively researched. Numerous articles and technical reports on the progress are published. The summary reports, including international ones, are produced every year [1,3,86,150,178].

1.2.2 Design Overview

The Neutrino Factory is a secondary beam machine; that is, a production beam is used to create secondary beams that eventually provide the desired flux of neutrinos. For the Neutrino Factory, the production starts from a high intensity proton beam that is accelerated to a moderate energy (beams of 2-50 GeV have been considered by various groups) and impinges on a target, typically made from a high-Z material (baseline choice is a liquid Hg jet). The collisions between the proton beam and the target nuclei produce secondary beams of pions that quickly decay (26.0 ns) into longer-living (2.2 μs) muon beams. The remaining part of the Neutrino Factory is used to condition the muon beam, rapidly accelerate it to the desired final energy of a few tens of GeV, and then store it in a decay ring with long straight sections where the intense beam of neutrinos is produced from the decaying muons (1.2.1). The resulting beam can then be used, for example, to hit a detector located hundreds or thousands of kilometers from the source.

The Feasibility Study II [150] that was carried out jointly by the Brookheaven National Laboratory (BNL) and the U.S. Neutrino Factory and Muon Collider Collaboration, established most of the current Neutrino Factory design ideas. Although a number of other ideas or variations of existing ones was proposed since FS II, later studies mainly concentrated on the exploration of already proposed concepts and their combinations. Their main goals were conducting a cost/performance analysis and developing consensus on a baseline design for the facility [10]. It is worthwhile to note that the details of the FS II design are highly influenced by a specific scenario of sending a neutrino beam from BNL to a detector in Carlsbad, New Mexico. The results that came out of the Feasibility Studies demonstrated technical feasibility of the Neutrino Factory, established its cost baseline, and the expected range of performance. Another important feature of this design is that such a Neutrino Factory could be comfortably constructed on the site of an existing U.S. laboratory, such as BNL or Fermi National Accelerator Laboratory (FNAL).

Here we list the main components of the Neutrino Factory (see example of the RLA-acceleration based variant of the Study IIa design in Figure 1.3) and their primary functions:

- **The Proton Driver** provides ≈ 2 MW beam of a moderate energy (several

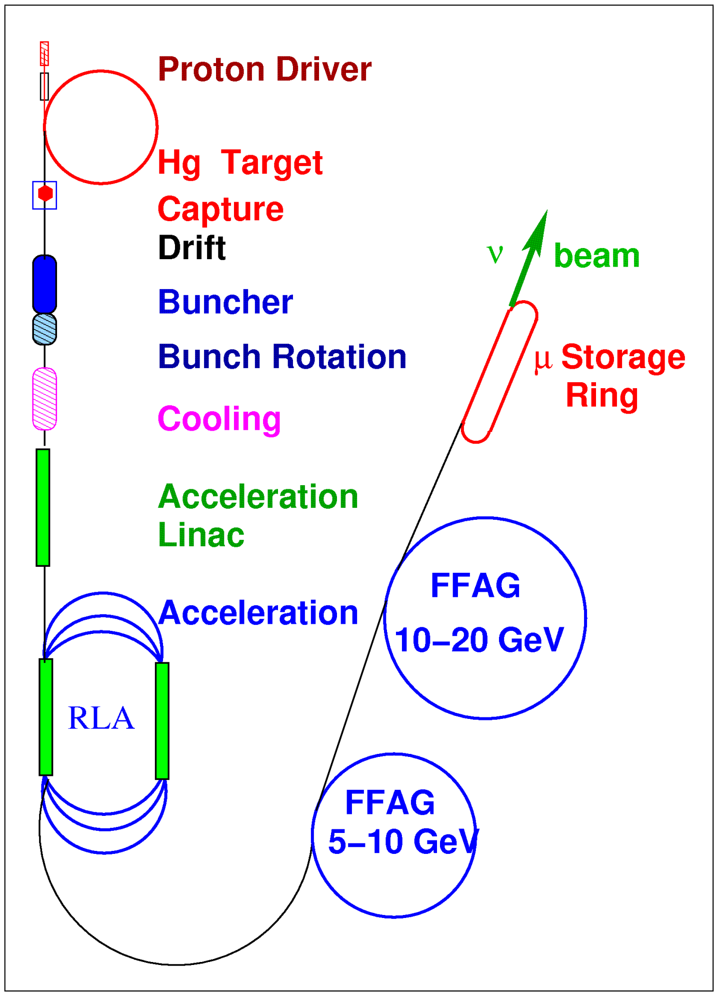


Figure 1.3: *Neutrino Factory schematics from the Feasibility Study IIa (RLA acceleration variant) [10]*

GeV) protons on target.

- **Target.** A high-Z target is put inside a 20 T solenoidal field (superconducting solenoid) to capture pions produced in the interactions of the inciding proton beam with the nuclei of the target material (liquid Hg jet) (see the longitudinal distribution of the particles 12 m from the target obtained from the MARS simulation code [138] in Figure 1.4).

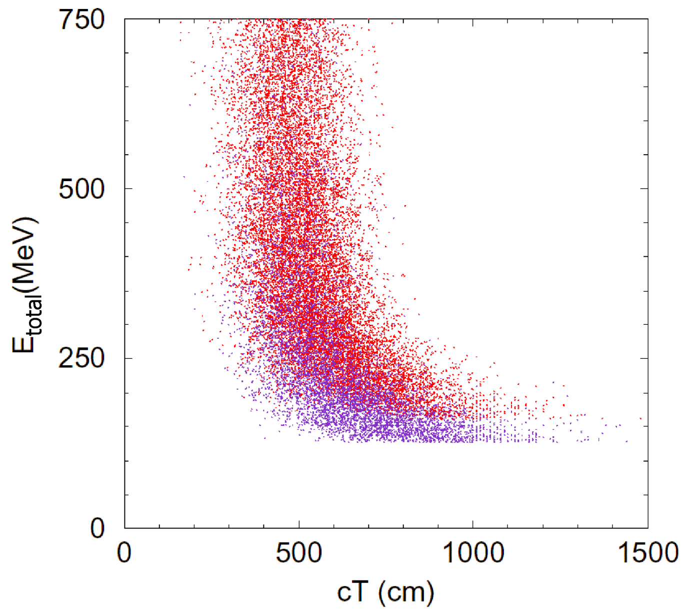


Figure 1.4: *Distribution of particles energies 12m from the target calculated by MARS,* $E_{\mathrm{total}} = E_0 + T$*, where* E_0 *is a rest energy (105.6 MeV for muons),* T *— kinetic energy*

- **The Front End** consists of the parts of the Neutrino Factory between the target and the acceleration section. It collects the pions coming from the target, conditions them to form a beam of the muons that are produced by pion decay, and then manipulates this beam to prepare it for the acceleration by efficiently matching the beam to the accelerator acceptance (see example of the longitudinal dynamics of a beam with a relatively small initial phase space in Figure 1.5, courtesy of David Neuffer [143]). It consists of the following subsystems:

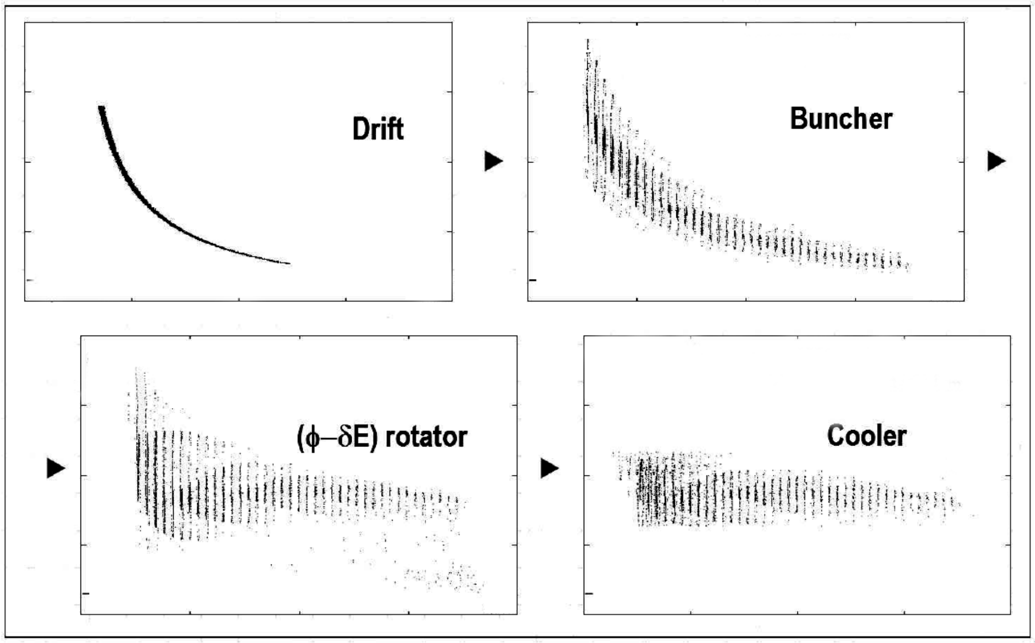


Figure 1.5: *Example of the longitudinal beam dynamics in the Front End (phase-energy plane), courtesy of David Neuffer [143]*

- **Capture.** The magnetic field at the target is smoothly tapered down to a much lower value, 2 T, which is then maintained through the bunching and phase rotation sections to keep the beam confined in the channel.

- **Decay.** This region is just an empty magnetic lattice where pions are allowed to decay to muons and where the particles of the resulting beam develop a correlation between a temporal coordinate and an energy.

- **Bunching and Phase Rotation.** First the large beam of muons is bunched with RF cavities of modest gradient, whose frequencies decreases as we proceed down the beam line. After bunching, another set of RF cavities, with changing frequencies, is used to rotate the beam in longitudinal phase space in order to reduce its energy spread and match the frequency to the one of the downstream RF cavities for efficient acceleration.

- **Ionization Cooling.** A solenoidal focusing channel, filled with high-gradient RF cavities and LiH absorbers, cools the transverse normalized RMS emittance of the beam [60]. In this stage muons in the momentum range of 150–400 MeV/c pass through the absorbers (from LiH in this design) thus reducing the total momentum (both longitudinal and transverse components). They are then reaccelerated in RF cavities to regain the longitudinal momentum component only. The total effect is a decrease in the transverse momentum spread and, therefore, the transverse emittance.

- **Acceleration.** Increases the beam kinetic energy from $\approx$ 138 MeV to a final energy in the range of 20–50 GeV. A superconducting pre-acceleration linear accelerator (linac) with solenoidal focusing is used to raise the muon beam energy to 1.5 GeV. It is then followed by a Recirculating Linear Accelerator (RLA), arranged in a dogbone geometry, that increases the beam energy to 5 GeV. Finally a pair of cascaded Fixed-Field, Alternating Gradient (FFAG) rings with combined-function doublet magnets, is used to bring the beam energy up to 20 GeV. Additional FFAG stages could be added to reach a higher beam energy, deemed necessary for physical reasons.

- **Storage and Decay Ring.** A compact racetrack-shaped superconducting storage ring in which $\approx 35\%$ of the stored muons decay to neutrinos and are sent toward the detector located approximately 3500 km from the ring. Muons survive in a ring for $\approx$ 500 turns.

1.2.3 Front End

Since the focus of our research is primarily in the exploration and optimization (see section 4.3) of the Neutrino Factory Front End section, we describe its design here in more detail. Since there are different variations of the Front End suggested by different research groups, including a Japanese FFAG study [81] and CERN linear channel studies [30], here we describe only the scheme based on the Neuffer phase rotation [74,141,142,145] and $\approx$ 201 MHz RFs in cooling and acceleration. The latest "International scoping study of a future Neutrino Factory and super beam facility" accepts this design as the currently preferred scheme. Its additional advantage is that it captures muons of both signs with equal efficiency. As most neutrino experiments are aimed at both neutrino and anti-neutrino studies, such setup doubles the overall efficiency. Finally, it replaces expensive induction linac-based design with a relatively inexpensive array of high-frequency RF cavities thus making the overall scheme better in terms of cost/performance.

As can be seen in Figure 1.4, pions that are produced by the nuclear collisions on target occupy a significantly large longitudinal phase space. The transverse phase space is mainly determined by the magnetic field strength of the solenoidal capture channel. According to the properties of the dynamics of particles in a solenoid [119], particles with the transverse momentum satisfying the following condition

$$p_{\perp} < 0.3\frac{BR}{2},$$

where B is the solenoidal field strength and R is the radius of the solenoid are captured after the target. In order to effectively accelerate the beam, it needs to be preconditioned to be fully contained within the capture transverse acceptance (30 π mm·rad) and the longitudinal acceptance (150 mm) of the subsequent accelerating section. Another constraint that the resulting beam has to satisfy is that only the particles that are contained within the longitudinal bucket of the accelerating system (bucket area depends on the RF frequency, phase and a field gradient) are captured into the accelerating regime. Transverse emittance should be decreased by cooling in order to achieve optimal intensity. Hence the main figure of merit for the Front End is the number of captured muons at the exit per incoming pions.

1.2.4 Decay, Bunching and Phase Rotation

Pions, and the muons that are produced by their decay, are generated in the target over a very wide range of energies (see, for example, Fig.1.4), but in a short time pulse (≈ 3 ns rms). Preparation of the muon beam for acceleration thus requires significant conditioning that includes reducing the energy spread and forming the beam into a train of bunches. Beam splitting into multiple bunches demonstrated itself as the best technique since in this case the bucket areas are significantly larger than the beam area hence a very good acceptance is expected [62].

First, the beam is allowed to drift to develop an energy correlation, with higher energy particles at the head and lower energy particles at the tail of the beam. Next, the long beam is separated into a number of short bunches suitable for capture and acceleration in a 201-MHz RF system. This is done with a series of RF cavities that have decreasing frequencies and increasing gradients along the beam line, separated by a suitably chosen drift spaces. The resultant bunch train still has a substantial energy correlation, with the higher energy bunches places first and progressively lower energy bunches coming behind. The large energy tilt is then phase rotated into a bunch train with a longer time duration and a lower energy spread using additional RF cavities of decreasing frequencies but constant gradient and drifts.

And example 2D simulation of the dynamics of the particles in the structure is shown in Fig.1.5. The beam at the end of the buncher and phase rotation section has an average momentum of about 220 MeV/c. The proposed [142] system is based on standard RF technology, and is expected to be much more cost effective than the induction-linac-based system considered in [150]. An additional benefit of the RF-based system is the ability to transport both signs of muons simultaneously. Finally, we note that there are many variations of the proposed scheme in order to study performance/cost relations and/or better fit to different designs of other sections of the Neutrino Factory. Examples include low-frequency rotation, phase rotation with a scaling FFAG, variations of gradients, phases, number of different RF frequencies, geometry of windows in RF cavities, gas-filled cavities, and other alternative designs.

The baseline Front End schematic from the latest International Scoping Study [62] is demonstrated in Figure 1.6. The baseline proton driver has an energy of 10 GeV. The capture system is a 12 m long channel with the solenoidal field dropping from initial 20 T to 2 T and the channel radius increasing from 75 mm to 250 mm. It is followed by a 100 m long decay section where the pions decay to muons and develop a correlation between the temporal position and an energy. This correlation is then employed by the 50 m long bunching section to split the beam into a train of bunches via a set of RF cavities of a modest gradient and decreasing frequencies. Then another set of RF cavities with higher gradients in the 50 m long rotator section are employed to rotate the beam in the longitudinal phase space to reduce its energy spread. We describe the logic behind the choice of the frequencies of the buncher and phase rotator later in detail later in this section. The final RMS energy spread in this scheme is ≈ 10.5 %. Then an 80 m long channel filled with high-gradient 201.25 MHz RF cavities and LiH absorbers in the solenoidal field is used to cool the transverse normalized RMS emittance from 17 π mm·rad to $\approx$ 7 mm·rad at a central muon momentum of $\approx$ 220 MeV/c.

To set up buncher parameters we choose some ideal particle to be the main central particle of the beam. Usually this is a particle with coordinates in the center of the

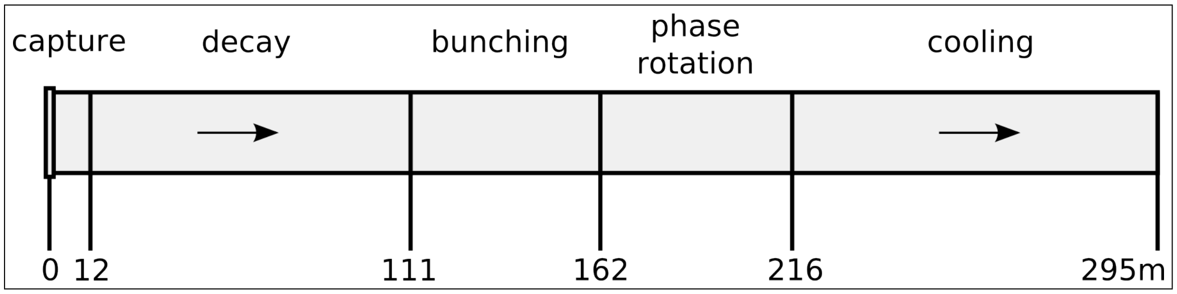


Figure 1.6: *The baseline Front End schematics from the latest International Scoping Study [62]*

beam particles coordinates distribution. We then set phases of RF cavities in such a way that this particle enters every cavity in the same phase ($\varphi_s = 0$) of EM field oscillations. By the virtue of the equations of motion in such a structure (see, for example, Chapters 13-14, [170]), particles near the central one in ($\varphi - \delta$E) phase space are then formed into a stable group called a "bunch". This group then oscillates around the central particle in the longitudinal phase space along with the motion of this particle in the accelerator. As a consequence of our choice of the main central particle, phase and cavity parameters, other particles are passing all cavities in the same $\varphi_s = 0$ phase and thus forming bunches around themselves. In the following text we will call them *central particles* and the one chosen first the *main central particle.* Of course, all central particles are not real particles, they are just an idealization chosen to make equations of motion simpler.

Each cavity in the buncher has its frequency set to maintain the following condition: the difference in the time of arrival of any two central particles in a place of RF field application remains equal to a fixed integer number of RF oscillation periods and this condition is maintained as the beam propagates through the buncher

$$\Delta t = t_{\mathrm{n}} - t_{\mathrm{c}} = z\left(\frac{1}{v_{\mathrm{n}}} - \frac{1}{v_{\mathrm{c}}}\right) = nT_{\mathrm{rf}} = n\frac{\lambda_{\mathrm{rf}}}{c}, n \in \mathbb{Z}\,, \tag{1.2.2}$$

where n is the number of the bunch counted from the main central particles, t_{c}, t_{n} and v_{c}, v_{n} are time-of-arrival of main central and n-th central particle (main central particle has $n = 0$) and their velocities respectively, T_{rf} is the period of RF field oscillations, λ_{rf} is the RF wavelength, c is the speed of light.

As the E field phase in the RF cavities is zero for the main central particle, it is also zero for other central particles since they pass the RFs when the field has zero strength and therefore their energies stay constant through the buncher. We keep the final frequency of the buncher and rotator fixed to match the beam into 201.25 MHz cooling and/or accelerating sections. Thus, setting $n = 1$, $\lambda_{\mathrm{rf}} = \bar{\lambda}$, $z = \bar{L}$ in (1.2.2), where z is the longitudinal coordinate with $z = 0$ at the beginning of the drift, $\bar{\lambda}$ is the final RF wavelength in buncher (defined by matching to the following cooler), $\bar{L}$ is the longitudinal coordinate of the last RF in buncher, we can define

$$\delta\left(\frac{1}{\beta}\right) = \left(\frac{1}{\beta_1} - \frac{1}{\beta_{\mathrm{c}}}\right) = \frac{\bar{\lambda}}{\bar{L}}\,, \tag{1.2.3}$$

where β_{c}, β_{n} are the main central particle and n-th central particle's normalized velocities, and then rewriting (1.2.2) we get

$$\frac{1}{\beta_{\mathrm{n}}} = \frac{1}{\beta_{\mathrm{c}}} + n\delta\left(\frac{1}{\beta}\right)\,. \tag{1.2.4}$$

Therefore for kinetic energies of central particles in the buncher we have following relation

$$T_n\left(\beta_{\mathrm{c}}, \delta\left(\frac{1}{\beta_{\mathrm{c}}}\right)\right) = W_0\left(\left(1 - \left(\frac{\beta_{\mathrm{c}}}{1 + n\beta_{\mathrm{c}}\delta\left(\frac{1}{\beta_{\mathrm{c}}}\right)}\right)^2\right)^{-1/2} - 1\right)\,, \tag{1.2.5}$$

where W_0 is the rest energy of the particle and T_n is the kinetic energy of the n-th central particle. From (1.2.3),(1.2.4) it follows that in order to keep the time of arrival difference between two central particles constant, the frequencies of RFs in a buncher should depend on the longitudinal coordinate through

$$\lambda_{\mathrm{rf}}(z) = z \cdot \delta\left(\frac{1}{\beta}\right) \Rightarrow \nu_{\mathrm{rf}}(z) = \frac{c}{z \cdot \delta\left(\frac{1}{\beta}\right)}\,, \tag{1.2.6}$$

In the buncher the RF gradient is adiabatically increased over the length of the buncher. The goal here is to perform an *adiabatic capture*, in which the beam within each bunch is compressed in phase such that it is concentrated near the central particle's phase. We arbitrarily choose this gradient to be increasing quadratically

$$V_{\mathrm{rf}}(z) = B\frac{(z - z_{\mathrm{D}})}{L} + C\frac{(z - z_{\mathrm{D}})^2}{L}, \qquad (1.2.7)$$

where V_{rf} is RF voltage, z_{D} is the longitudinal coordinate of the beginning of the buncher (equal to the drift length), B and C are positive constants, defined by chosen initial and final RF gradients in a buncher, L is the length of the buncher. Note that, since each of the bunches is centered at a different energy, they all have different longitudinal oscillation frequencies, and a simultaneously matched compression for all bunches is not possible. Instead a quasi-adiabatic capture is performed in order to achieve an approximate bunch length minimization in each bunch.

Following the buncher is the $(\varphi - \delta\mathrm{E})$ *vernier rotation system* in which the RF frequency is almost fixed to the matched value at the end of the buncher and the RF voltage is constant. In this system the energies of the central particles of the low-energy bunches increase, while those of the high-energy bunches decrease. So the whole energy spread reduces to the point where the beam is a string of similar-energy bunches, which are captured into the ~201 MHz ionization cooling system matched to the central energy of the beam.

We now describe the rotator parameters calculation in more detail. At the end of the buncher we choose two reference particles (n_1 and n_2) which were kept $(n_2 - n_1)$ RF periods from each other along the buncher, and the *vernier offset* δ. We then keep the second central particle at $((n_2 - n_1) + \delta)\lambda_{\mathrm{rf}}$ wavelengths from the first one through the rotator. With this choice, the second central particle passes all RF cavities in a constant accelerating phase φ_{n_2} having constant energy change ΔT_{n_2}.

Ffter $\left|T_{\mathrm{n}_1} - T_{\mathrm{n}_2}\right| / \Delta T_{\mathrm{n}_2}$ cavities, the energies of the first particle (usually we choose main central particle as first central particle) and the chosen second central one will be nearly equal. From this consideration we can derive the relation between the energy change of the n-th central particle in each cavity of the rotator and the rotator parameters:

$$\Delta T_n(E_{\mathrm{rf}}, \delta, n_1, n_2) = E_{\mathrm{rf}} \sin\left(2\pi\delta \frac{n - n_1}{n_2 - n_1}\right) , \tag{1.2.8}$$

where ΔT_n is the energy change of the n-th central particle, δ is the vernier parameter, n_1 and n_2 are the numbers of chosen central particles, E_{rf} is the RF gradient of the cavities in rotator. This process also aligns the energies of other central particles and their bunches, hence at the end of the rotator we have the beam rotated in $(\varphi - \delta\mathrm{E})$ space with a significantly reduced energy spread. A simulation of the process in $(\varphi - \delta\mathrm{E})$ phase space is shown in Figure 1.5.

Combining equations (1.2.5) and (1.2.8) we can obtain the formula for the central energy of the n-th bunch after buncher and phase rotator in terms of their design parameters:

$$T_n^{\mathrm{fin}}\left(\beta_{\mathrm{c}}, \delta\left(\tfrac{1}{\beta_{\mathrm{c}}}\right), E_{\mathrm{rf}}, \delta, n_1, n_2\right) = \tag{1.2.9}$$
$$W_0\left(\left(1 - \left(\frac{\beta_{\mathrm{c}}}{1 + n\beta_{\mathrm{c}}\delta\left(\frac{1}{\beta_{\mathrm{c}}}\right)}\right)^2\right)^{-1/2} - 1\right) + mE_{\mathrm{rf}} \sin\left(2\pi\delta\frac{n-n_1}{n_2-n_1}\right)$$

where m is the number of the RF cavities in rotator.

1.3 Optimization Problems

1.3.1 Introduction

We live and work to reach our goals in a world where all available resources are restricted. There is always a limit on the amount of time, money or technologies that we have a control on. Most often we want to achieve our goals with maximum satisfaction, spending a minimal amount of the limited resources, and producing a minimal amount of unwanted side effects. In vague terms this is a formulation of the optimization problem. Many design problems of science and industry can be formulated as the optimization of a certain objective function under a particular set of constraints. Many problems of accelerator and beam physics can be formulated as optimization problems. As of the last feasibility study [62] most of the questions of the Neutrino Factory design R&D (see section 1.2.2) were questions of optimization, e.g. optimum beam energy, repetition rate, and bunch length for the Proton Driver. Target material, optimal production of the Front End (delivery of the most muons fully contained within the capture transverse acceptance) and the longitudinal acceptance of the Accelerating section (see section 4.3), were investigated. The simple question of how to design of the one of the basic accelerator lattice building blocks (see section 4.1) and the complicated problem of the stability estimating for the particle dynamics in large and complex circular accelerators (see section 4.2) are solved using the framework of optimization. Thus we see that optimization problems and, of course, optimization methods, are of great importance for many areas of the modern science.

If we can construct a function that maps the properties we control to the measures of the properties we want to optimize (minimize unwanted, maximize wanted), we get a mathematical *optimization problem.* Properties under control are called *control variables* or *parameters*, the function of measures is called an *objective function.* If we seek the minimum of the objective function, it is typically called a *cost function*; if the minimum is known to be zero, it is called an *error function.* In cases where the maximum is sought, it is referred to as a *fitness function.* Since the maximization can be turned into minimization by flipping the sign of the objective function, without loss of generality we can restrict our consideration to minimization problems.

Optimization of even a single performance measure is already a hard and complex problem, but simultaneous optimization of several measures poses additional qualitative complexity. Here optimal values of different measures are frequently achievable with different combinations of control parameters and even the definition of the optimal solution itself is a non-trivial problem. Therefore it is usually preferable to use mathematical models with a scalar objective function which provides some combined performance characteristic of the system under consideration (see [102] for a discussion on constructing combined objective functions). The problem of optimizing a single objective function is typically called an optimization problem or a *single-objective optimization* problem while the problem of optimizing several objective functions is referred to as a *multi-objective optimization* problem.

As was mentioned earlier, resources in real-life problems are typically limited. Those limits can usually be modelled as equality and inequality constraints imposed on control variables. An optimization problem in their presence is called a *constrained optimization* problem.

There exist many problems that can be formulated as optimization problems and many methods to build a mathematical model for a problem. Such a variety of methods produces a large variety of optimization problems. Apart from the properties mentioned earlier they are often characterized with respect to:

- *Parameter types:* on/off, discrete, continuous, functions of a certain type, etc.

- *Dimensionality:* number of control parameters.

- *Presence of noise:* noise could be present in parameters and in the objective function values.

- *Properties of the objective function:* modality, time-dependence, continuity, differentiability, smoothness, separability, etc.

Classification of optimization problems is done with respect to these properties. For example, combinatorial optimization deals with discrete control variables from the space that usually contains a finite number of candidate solutions. Continuous optimization typically considers control parameters from the space of real numbers thus the number of potential solutions is infinite. Continuous optimization, in turn, is divided into linear programming (linear objective function, linear constraints) [167] and quadratic programming (quadratic objective function, linear constraints) [147]. Here well-known polynomial algorithms can solve a problem provided that certain solution existence conditions are met. Au contrary, in non-linear programming (non-linear non-quadratic objective function, nonlinear constraints) there are no general algorithms with guaranteed convergence and estimate on the number of operations required.

In this work we restrict our consideration to the optimization problems for which the control parameters are real values, and the objective function is scalar and is generally nonlinear. A large number of design problems can be formulated as optimization problems of this type (see sections 4.2, 4.1, 4.3). In accelerator design control parameters usually represent physical properties of the accelerator components, e.g. magnet positions, strengths, lengths and apertures.

1.3.2 Unconstrained Optimization

Under our assumptions we can formulate the optimization problem as follows. Let $S \subseteq \mathbb{R}^v$ be a search domain, $\mathbf{x} \in S$ be a vector of v control parameters assuming real values, and

$$f : S \longmapsto \mathbb{R} \tag{1.3.1}$$

be an objective function. Let $\mathbf{x}$ be subjected to equality and inequality constraints

$$g_i(\mathbf{x}) = 0,\; i = 1, \ldots, n \tag{1.3.2}$$

$$h_i(\mathbf{x}) \leq 0,\; i = 1, \ldots, m \tag{1.3.3}$$

Then the general problem of mathematical optimization is to find $f^* \in \mathbb{R}$ such that

$$f^* = \min_{\mathbf{x} \in S} f(\mathbf{x}) \tag{1.3.4}$$

and corresponding $\mathbf{x}^* \in S$:

$$f^* = f(\mathbf{x}^*)$$

which is usually written as

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in S} f(\mathbf{x}), \tag{1.3.5}$$

such that it satisfies constraints (1.3.2), (1.3.3).

In some cases the optimization problem (1.3.4), (1.3.5) is softened. It is considered solved if we find $\mathbf{x}^* \in S$ such that there exist $\delta \in \mathbb{R}$ and

$$f^* = f(\mathbf{x}^*) = \min_{\mathbf{x} \in S_\delta} f(\mathbf{x}), \tag{1.3.6}$$

where $S_\delta \subseteq S$ is a delta-neighborhood of $\mathbf{x}^*$. In this case f^* is called a local minimum and the problem is called a *local optimization* problem. The original problem (1.3.4), (1.3.5) is called *global optimization* problem and the corresponding f^* is called a global minimum ($\mathbf{x}^*$ is also called a global minimum or, to make a distinction, a minimizer). In this work we primarily consider global minimization problems therefore for all considered optimization problems a global minimum is sought unless it is explicitly stated otherwise. We cannot make any assumptions about the uniqueness of this minimum for an arbitrary optimization problem. Therefore by a minimum in these problems we mean any of the non-unique ones, if there are several, unless stated otherwise.

In some cases a global optimization problem is restricted by adding additional conditions. If it is required to prove that the found minimum is global, and provide rigorous bounds for its value, then such subfield of optimization is called a *rigorous global optimization* (covered in more detail in section 2.2). If the constraints (1.3.2), (1.3.3) are defined, the problem is called a *constrained optimization* problem, otherwise it is called an *unconstrained optimization* problem. In Chapter 2 we review of the methods of unconstrained optimization with Evolutionary Algorithms, describe the implemented GATool continuous unconstrained optimization EA, present studies on its performance and potential of the integration with the rigorous optimization package COSY-GO.

1.3.3 Constrained Optimization

In this section we consider constrained optimization problems, i.e. problems (1.3.1), (1.3.5), and (1.3.4) in the presence of constraints (1.3.2) and (1.3.3), in more detail and introduce the relevant terminology.

When constraints are imposed, the set

$$F = \left\{ \mathbf{x} \in S \subseteq \mathbb{R}^v \,\middle|\, g_i(\mathbf{x}) = 0,\ h_j(\mathbf{x}) \le 0,\ i = 1, \ldots, n\,,\ j = 1, \ldots, m \right\} \tag{1.3.7}$$

is called a *feasible set*. It contains all vectors from the search domain that simultaneously satisfy all constraints. Such vectors $\mathbf{x} \in F$ are called *feasible*, all other vectors are called *unfeasible*. If at some point $\mathbf{x} \in S$ the inequality constraint $h_j(\mathbf{x})$ holds as an equality ($h_j(\mathbf{x}) = 0$), it is called *active* at $\mathbf{x}$. Equality constraints are considered active everywhere in S. Using these definitions we can rewrite a constrained optimization problem formulation as

$$\begin{aligned} f^* &= \min_{\mathbf{x} \in F} f(\mathbf{x}) \\ \mathbf{x}^* &= \arg\min_{\mathbf{x} \in F} f(\mathbf{x}), \end{aligned} \tag{1.3.8}$$

where a sought minimum is also called a *feasible minimum*.

Search domain S is usually given as a v-dimensional box

$$S = \left\{ \mathbf{x} \in \mathbb{R}^v \,\middle|\, \underline{x_l} \le x_l \le \overline{x_l},\ l = 1, \dots, v \right\} \tag{1.3.9}$$

and thus can be treated as a set of inequality constraints included into the feasible set definition (1.3.7) (note that by this definition $F \subseteq S$). However, for most real-world problems S can be determined rather easily by estimating physically reasonable ranges for control parameters and thus has a simple convex structure with linear boundaries. Feasible set, in contrast, can be specified by a large number of complex non-uniform constraints and therefore can have an extremely complex structure. Depending on the problem F can have nonlinear boundaries, be non-convex, not connected, have measure of zero, or even be empty (in this case the constrained optimization problem has no solution) and is thus hard to study and visualize. Moreover, for many test and real-life problems $|F| \ll |S|$, hence the distinction between a search space and a feasible set is fully justified. The quantity

$$\rho = \frac{|F|}{|S|} \in [0, 1] \tag{1.3.10}$$

is often used as one of the measures of the difficulty of the constrained optimization problem. Empirically it can be viewed as a measure of the difficulty that the constrains are adding to the problem, in comparison with the difficulty of the unconstrained problem with the same objective function. Generally, the smaller ρ is, the harder it is for the algorithm to find feasible points in the search space. Note, however, that for an arbitrary problem this factor is hard to estimate because of the unknown and frequently complex structure of F. Random sampling of the search space is usually employed for such estimation [130].

It is worth noting that the ρ factor alone does not determine the constrained problem's difficulty completely. However, the theoretically developed framework for such analysis and comparison of different problems does is not established yet. Most of the difficulty ratings are assigned heuristically and are derived from the practice. For example, it is well-known that the problems with convex feasible sets are easier to solve than the ones with non-convex feasible sets; that the problems with disjoint F are harder to solve than the ones with connected F, etc [135]. It is also known that the difficulty of the problem often increases as the number of constraints that are active at the sought feasible minimum increases (for an arbitrary problem this information is not available before the minimum is found). It is also worth noting, that the optimization performance is algorithm-dependent (see section 2.1.4), hence it cannot be measured for the problem itself without the considerations on the algorithm. Some work towards characterizing constrained problems and determining if they are EA-hard can be found in [57].

Inequality constraints (1.3.3) can be transformed into equality constraints by introducing "dummy" variables $\xi_j,\ j = 1, \ldots, m$. In this case each inequality constraint

$$h_j(\mathbf{x}) \leq 0$$

is converted into an equivalent equality constraint

$$h_j(\mathbf{x}) + \xi_j^2 = 0.$$

Each equality constraints (1.3.2) can in turn be transformed into two inequality constraints:

$$\begin{aligned} -g_i(\mathbf{x}) &\leq 0,\ i = 1, \ldots, m \qquad (1.3.11) \\ g_i(\mathbf{x}) &\leq 0,\ i = 1, \ldots, m, \end{aligned}$$

or, for the methods that do not rely on smoothness of the constraint functions to one inequality constraint

$$|g_i(\mathbf{x})| \leq 0,\ i = 1, \ldots, m. \qquad (1.3.12)$$

For practical purposes of non-rigorous optimization

$$|g_i(\mathbf{x})| - \varepsilon \leq 0,\ i = 1, \ldots, m, \qquad (1.3.13)$$

where ε is an acceptable tolerance for equality constraint satisfaction is also frequently used. Using these transformation we can limit our consideration to the problems with either equality-only or inequality-only constraints without loss of generality. For simplicity we consider only inequality constraints, i.e. constraints of the type (1.3.3), treating n as a total number of constraints. In this case the feasible set (1.3.7) is defined as

$$F = \left\{ \mathbf{x} \in S \subseteq \mathbb{R}^v \,\middle|\, h_j(\mathbf{x}) \leq 0\,,\ j = 1, \ldots, n \right\}. \qquad (1.3.14)$$

If certain conditions on the constrained problem are satisfied, methods to solve it analytically can be applied. For example, the well-known and widely applied Lagrange Multipliers Method requires an objective function and constraint functions be written in an algebraic form, be deterministic, and differentiable. The generalization of these conditions are Karush-Kuhn-Tucker (KKT) conditions [110] which formulate the necessary conditions for a point to be a conditional extrema. Additional assumptions formulated in a variety of different forms and called regularity conditions assure that the solution is non-degenerate. Under additional assumptions about constraint functions, for example, when inequality constraint functions are affine and equality constraint functions are convex, sufficient conditions for the point to be a global minimum can be formulated [7].

Unfortunately, many real-life problems are posed in such a way that their objective and/or constraint functions make the KKT conditions not applicable. In these cases various numerical optimization methods are usually employed. Constraints are often incorporated into an objective function or used to transform the problem into a multi-objective optimization problem with help of penalty and barrier functions (see Chapter 3) or the Lagrange multipliers method. After such simplifying transformation the optimization methods for the unconstrained problems can be applied to solve the constrained problems. In general, most optimization methods for constrained problems are based on the methods designed for unconstrained problems [147]. In Chapter 3 we review constrained optimization methods (mostly those used in Evolutionary Algorithms), propose a new method for a constrained optimization and study its performance.

CHAPTER 2

Unconstrained Optimization

2.1 Optimization Methods

Once the mathematical model of the problem is developed, the types of the control parameters are chosen and the objective function is constructed, the problem is most frequently need to be solved. For all but rigorous optimization problems, point-based iterative methods are most often used. Each step, they operate on a population of points (for single-point methods it consists of one point) to generate the next, supposedly better population in order to eventually converge to the sought minimum. Single-point methods (also called descent methods) typically use the greedy iterative search strategy from Figure 2.1.

2.1.1 Derivative-based Methods

If the objective function is sufficiently differentiable, derivative-based methods can be utilized. The classic and the most well-known among them are: Newton's, Steepest Descent, Conjugate Gradient and Quasi-Newton methods, which all are iterative

```
1. Start from the initial guess $\mathbf{x}_0$.
2. Compute the search direction $\mathbf{p}_k$.
3. Choose the step $\lambda_k$ to achieve $\varphi(\lambda_k) = f(\mathbf{x}_k + \lambda_k \mathbf{p}_k) < f(\mathbf{x}_k) = \varphi(0)$
   (more or less extensive line search).
4. If the move is successful, move to the next point $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{p}_k$.
5. Repeat steps 2-4 until $\|f(\mathbf{x}_k) - f(\mathbf{x}_{k+1})\| < \varepsilon$, where $\varepsilon$ is a required
   precision.
```

Figure 2.1: *One-point greedy iterative search strategy*

single-point local minimizers. Multi-start and clustering techniques were developed in an attempt to turn them into multi-point global minimizers [172]. These techniques increase the method's chances of finding a global minimum rather than being trapped in one of the local minima.

If the derivative cannot be obtained or is too expensive to calculate, derivative-free direct search heuristic methods can be employed. These methods can be deterministic or stochastic depending on the usage of random numbers in the search procedure. Examples of the direct search methods are: Random Walk, Simulated Annealing, and Hooke-Jeeves, which are single-point methods, and Nelder-Mead (nonlinear simplex), Evolutionary Algorithms, and Particle Swarm Optimization, which are multi-point methods. Rigorous optimization typically employs interval methods (see section 2.2.2).

If the objective function f is analytic, then denoting the gradient of the function $f(\mathbf{x})$ at the point $\mathbf{x}_0$ as $\mathbf{g}(\mathbf{x}_0)$ and its Hessian matrix at the same point as $\mathbf{H}(\mathbf{x}_0)$, we can write its Taylor expansion at $\mathbf{x}_0$ in the form:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \mathbf{g}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^{\mathrm{T}} \mathbf{H}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \dots \quad . \tag{2.1.1}$$

If we then differentiate this expression, we get the expansion for the gradient:

$$\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{x}_0) + \mathbf{H}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \dots . \tag{2.1.2}$$

Taking into account that the necessary condition for the point to be a minimum is that $\mathbf{g}(\mathbf{x}^*) = 0$, substituting into (2.1.2) and neglecting terms of the order 3 and higher since the function is approximately quadratic near the minimum, we obtain the formula for the minimizer

$$\mathbf{x}^* = \mathbf{x}_0 - \mathbf{H}^{-1}(\mathbf{x}_0)\mathbf{g}(\mathbf{x}_0) \tag{2.1.3}$$

that is obviously exact for quadratic $f(\mathbf{x})$. For non-quadratic twice differentiable functions, it can be turned into the iterative procedure given in Figure 2.1. By replacing $\mathbf{x}^*$ with $\mathbf{x}_{k+1}$, $\mathbf{x}_0$ with $\mathbf{x}_k$ and multiplying the direction by the step size λ_k, we obtain the multi-dimensional *Newton's method* formula:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \cdot \mathbf{H}^{-1}(\mathbf{x}_k)\mathbf{g}(\mathbf{x}_k). \tag{2.1.4}$$

This method basically approximates a function at the current point with the quadratic part of the Taylor polynomial and then makes a step to the minimum using the exact formula (2.1.3). Since the formula is exact for the quadratic functions only, this step does not reach the minimum but hopefully produces a next point that is closer to it. In the sufficiently small neighborhood of the minimum the function is dominated by the quadratic terms of the expansion (2.1.1) so the approximation gets more accurate and the convergence speed increases.

However, the calculation of the Hessian matrix on each step is computationally expensive. Its inversion is also an expensive and, moreover, numerically unstable operation. Various methods like Gauss-Newton, Fletcher-Reeves, Davidon-Fletcher-Powell, Broyden-Fletcher-Goldfarb-Snanno and Levenberg-Marquardt [147] were developed to avoid this problem.

One of the simplest of the gradient-based methods, the method of the *Steepest Descent* (also called *Gradient Descent*), is based on the fact that the function decreases with the largest rate in the direction opposite to the direction of the function's gradient at this point. Hence the step direction $\mathbf{p}_k$ is chosen as

$$\mathbf{p}_k = -\mathbf{g}(\mathbf{x}_k) \tag{2.1.5}$$

and the iterative formula is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \cdot \mathbf{g}(\mathbf{x}_k). \tag{2.1.6}$$

Formula (2.1.6) can also be viewed as formula (2.1.4) with inverse Hessian approximated by the identity matrix. However, such an approximation is very crude and leads to the step size control problem due to the loss of the information about the function curvature contained in the Hessian. This leads to very slow convergence rates for functions like Rosenbrock's function (see section C.4). The anti-gradient in the narrow valleys seen on its contour plot is directed towards another wall, while the direction that leads to the minimum positions itself along the walls, i.e. almost orthogonal to the direction calculated by the Steepest Descent method. Hence a typical path to the minimum consists of a series of zigzags from one wall to another, the overall progress to the minimum is slow and the search process could be terminated prematurely by spending all its budgeted number of steps. This problem is well-known in optimization and is referred to as the “error valley” problem.

Other derivative-based methods designed to solve the problems of slow convergence and high computational cost can roughly be divided into two categories. *Quasi-Newton methods* use formula (2.1.4) with various approximations for the inverse Hessian while *Conjugate Gradient methods* employ another scheme to select directions $\mathbf{p}_k$ (based on the conjugate gradient method developed for fast minimization of quadratic functions). Some methods combine both approaches adding heuristics to determine directions and step sizes. The disadvantage of these derivative-based methods is that in order to work they require the differential of the objective function or the second differential which are often not defined or are expensive to obtain. Also, while their convergence on the quadratic functions is very fast, it generally does not hold for arbitrary nonlinear functions. Finally, they are all local minimization methods so they are best suited for unimodal objective functions. For multimodal objective functions several approaches were developed: *Multi-start techniques* where optimization with one of the iterative methods from Figure 2.1 is started several times from different initial points and *Clustering methods* which attempt to identify basins of attraction (or clusters) for each extremum in order to determine the number of initial points needed to find all minima.

Here we describe one of the heuristic Quasi-Newton methods, namely the Levenberg-Marquardt method. It forms the core of the LMDIF optimizer, one of the built-in *COSY Infinity* [23] optimization methods. We start with the *Gauss-Newton method* that is designed to solve nonlinear least squares problems, i.e. problems where the objective function has a special form:

$$S(\mathbf{x}) = \sum_{i=1}^{m} \left(f_i(\mathbf{x})\right)^2 . \tag{2.1.7}$$

Here $\mathbf{x}$ typically consists of v parameters to be fitted, $f_i,\ i = 1, \ldots, m$ are the functions of $\mathbf{x}$, typically experimental results. If we denote $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}))^{\mathrm{T}}$, we can write the objective function as

$$S(\mathbf{x}) = \mathbf{f}(\mathbf{x})\mathbf{f}(\mathbf{x})^{\mathrm{T}}. \tag{2.1.8}$$

Then its gradient is given by

$$\mathbf{g}(S(\mathbf{x})) = 2\mathbf{J}_{\mathbf{f}}(\mathbf{x})^{\mathrm{T}}\mathbf{f}(\mathbf{x}), \tag{2.1.9}$$

where

$$\mathbf{J_f}(\mathbf{x}) = \mathrm{Jac}_{\mathbf{f}}(\mathbf{x}) = \left\{ \frac{\partial f_i(\mathbf{x})}{\partial x_j} \right\}_{i=1,\ldots,m,\ j=1,\ldots,v} \tag{2.1.10}$$

and its Hessian is given by

$$\mathbf{H}_S(\mathbf{x}) = 2\mathbf{J_f}^{\mathrm{T}}(\mathbf{x})\mathbf{J_f}(\mathbf{x}) + 2\sum_{i=1}^{m} f_i(\mathbf{x})\mathbf{H}_{f_i}(\mathbf{x}), \tag{2.1.11}$$

where $\mathbf{H}_{f_i}$ is the Hessian of f_i.

Usually the objective function S is constructed so that its minimum value is zero. By (2.1.7) it is attained only at the point $\mathbf{x}^*$ where all f_i are zeros. For continuous f_i this means that in the neighborhood of the minimum the second term in the expression (2.1.11) for the Hessian of $S(\mathbf{x})$ is getting close to zero and the Hessian can be approximated as

$$H_S(\mathbf{x}) \approx 2\mathbf{J_f}(\mathbf{x})^{\mathrm{T}}\mathbf{J_f}(\mathbf{x}). \tag{2.1.12}$$

Substituting expressions (2.1.9) and (2.1.12) into Newton's method iterative formula (2.1.4), we obtain an iterative formula for the Gauss-Newton method:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \cdot \left(\mathbf{J_f}(\mathbf{x}_k)^{\mathrm{T}}\mathbf{J_f}(\mathbf{x}_k)\right)^{-1}\mathbf{J_f}(\mathbf{x}_k)^{\mathrm{T}}\mathbf{f}(\mathbf{x}_k). \tag{2.1.13}$$

The advantage of this method is that while it does not require a computation of the second derivatives, it still uses information from them (although only approximately). In cases where the sought minimum is greater than zero, the neglected term in expression (2.1.11) for the Hessian can become significant thus making the approximation (2.1.12) crude and decreasing the quality of the search procedure (2.1.13). In this case the *Levenberg-Marquardt algorithm*, which is a heuristic combination of the Gauss-Newton algorithm and the Gradient Descent could be a better approach. It is generally more robust (albeit sometimes slower) than the Gauss-Newton or the Gradient Descent algorithms alone [77], in the sense of reliably finding solutions even if the initial guess is far from the resulting minimum.

The Levenberg-Marquardt algorithm iterative formula is a slightly changed version of the Gauss-Newton formula (2.1.13):

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \cdot \left(\mathbf{J_f}(\mathbf{x}_k)^{\mathrm{T}}\mathbf{J_f}(\mathbf{x}_k) + \gamma\mathbf{I}\right)^{-1}\mathbf{J_f}(\mathbf{x}_k)^{\mathrm{T}}\mathbf{f}(\mathbf{x}_k). \tag{2.1.14}$$

Here I is the identity matrix, γ is a non-negative value called damping parameter. It is adjusted on each iteration using the following logic: if the value of the objective function S decreases rapidly, the damping factor is decreased to make the algorithm's behaviour closer to that of the Gauss-Newton algorithm. If an iteration results in insufficient change of an objective function value, the damping factor is increased to make algorithm's behaviour closer to the one of the Steepest Descent. The choice of the damping parameter and a scaling strategy is usually a matter of heuristic and might require fine-tuning for the problem.

Formula (2.1.14) is actually a development of Levenberg, while the insight of Marquardt was to replace the identity matrix with the diagonal of the approximated Hessian (2.1.12) in order to use information contained in it even when the damping factor is high and the method behaves like the Gradient Descent. This helps to avoid the classic "error valley" problem mentioned earlier. The Levenberg-Marquardt method is essentially heuristic, which makes it hard to theoretically prove its convergence, but it is known to work extremely well in practice and thus is often considered as one of the standard methods of the nonlinear optimization. Note, however, that for higher-dimensional problems its performance is significantly reduced by the expensive and ill-conditioned matrix inversion needed on each step of the iteration (2.1.14). Several initial steps of the LMDIF minimization process (which is a heuristically enhanced implementation of Levenberg-Marquardt algorithm) on the 2-dimensional Sphere test function (see section C.1) starting from the initial guess $(10, 10)$ are shown in Figure 2.2.

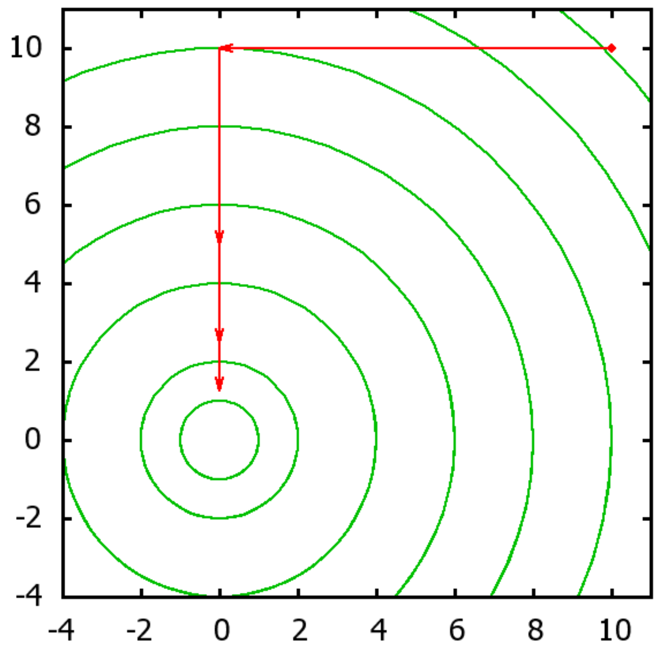


Figure 2.2: *First steps performed by the LMDIF (*COSY Infinity *built-in optimizer) on the* 2*-dimensional Sphere test function (see section C.1) starting from the initial guess* $(10, 10)$

2.1.2 Direct Search Methods

The methods described so far require the objective function to be at least one time differentiable and its derivatives to be cheap to obtain. Unfortunately, for most real-life problems these conditions do not hold. Therefore we cannot use derivatives to select the direction and the step size for the greedy strategy from Figure 2.1. The *Direct Search* heuristic algorithms, also known as “generate-and-test” methods, are heavily used for such problems. Their distinctive feature is that they divide the next point search into generation and selection phases. Possible moves generated during the first phase are either accepted on the second phase and the iteration advances or they are rejected and then a new move is generated.

The simplest direct search method is probably the *Brute Force method*. Here the search domain is covered by a grid which is then visited point by point and the best found minimizer is updated every time a better one is found. Due to its search method, this algorithm is also called the *Naïve Sampling method*. It suffers from several obvious drawbacks: strong dependence of the optimal grid size on the problem and an exponential growth of the number of the points in the grid with dimension. Since the algorithm visits all points in the grid during search, it leads to an exponential growth of the search time.

The *Random Walk method* uses a v-dimensional Gaussian distribution to generate trial step vectors $\mathbf{\Delta x}$ randomly. Then the trial points on k-th step are given by

$$\mathbf{x}_{k,\text{trial}} = \mathbf{x}_{k-1} + \mathbf{\Delta x}.$$

The selection is greedy, i.e. the first trial point such that

$$f(\mathbf{x}_{k,\text{trial}}) < f(\mathbf{x}_{k-1})$$

is accepted as the new iterate. While the method seems to not be as badly affected by the "dimensionality curse", the problem of determining the optimal parameters for the Gaussian distribution that is used to generate step sizes still remains.

Hooke and Jeeves method also known as *Pattern Search* attempts to dynamically adjust step size by exploring coordinate axes separately using per-axis step sizes, which are reduced if the trial move is unsuccessful. In practice, this approach is more effective than the Random Walk and Brute Force methods. More sophisticated techniques to find the direction and step size exist but methods of this type are still typically used only in combination with other methods.

Greediness of the selection process in direct search algorithms often leads to their convergence at a local minimum once they get into its basin of attraction. To avoid this problem, the *Simulated Annealing* algorithm [105] modifies the selection criteria to also accept some "uphill" moves on the function's landscape. This method is frequently used for metaheuristic algorithms and is also one of the built-in *COSY Infinity* optimizers called ANNEALING.

Strictly speaking, Simulated Annealing is not a method, it is a selection strategy that replaces the greedy selection from step 3 of the greedy iterative search algorithm from Figure 2.1. It helps to avoid being trapped in a local minimum, which is a common case for greedy methods, and increases the chances of finding the global minimum. As such it is often used in conjunction with direct search methods, most frequently with the Random Walk method.

The inspiration for the method is the annealing process from metallurgy when a material is first heated (recovery phase) and then slowly cooled to change its properties such as strength and hardness (recrystallization phase). The heating causes atoms to move freely and the slow cooling gives them time to find configurations with minimal energy. A strategy built on this analogy allows the current search point to move to the next point with a probability that depends on the difference of the function values at the current and the candidate points (energy difference)

$$d_k = f(\mathbf{x}_{k,\text{trial}}) - f(\mathbf{x}_{k-1})$$

and the value of a parameter T (temperature) that is gradually decreased as the search progresses. There are several parameters that influence the performance of the method:

1. *Probability of acceptance:* defines the probability with which the next move is accepted. It has to posses the following properties: be non-zero for any values of d and T, but such that the probability of accepting a move with $d > 0$ (function value increases) decreases as temperature decreases, while the probability of accepting moves with $d < 0$ (function value decreases) increases

or stays constant (in classical Simulated Annealing it is equal to 1 for all such moves). Frequently used is the following formula for the probability satisfying these requirements:

$$P(d,T) = \begin{cases} 1 & , d < 0 \text{ or } T = 0 \\ e^{-\frac{d}{T}} & , \text{otherwise} \end{cases} . \qquad (2.1.15)$$

Since there is a non-zero probability of accepting moves that worsen the final result, the best found point is typically saved separately.

2. *Annealing temperature schedule:* determines the change of temperature with iterations $T = T(k)$. If the temperature decreases too fast, the method might converge prematurely. If it decreases too slowly, calculations might take an unnecessarily long time. Also note that for $T = 0$ the strategy turns to greedy.

3. *Trial point generation method:* is not a part of the Simulated Annealing algorithm itself but since the choice of the probability of acceptance and annealing schedule depend both on a problem and on an exploration method, all three control parameters should be selected and fine-tuned together.

Further developments and enhancements of the algorithm like Adaptive Simulated Annealing, Boltzman Annealing, Simulated Quenching, Fast Annealing and Reannealing are discussed in [91]. Several initial steps of the ANNEALING minimization process (which is an implementation of the Random Walk method with Simulated Annealing selection strategy) on the 2-dimensional Sphere test function (see section C.1) starting from initial guess $(10, 10)$ are presented in Figure 2.3.

Another heuristic search method that tries to avoid a convergence to a local minimum is the well-known *Nelder-Mead method*, also known as the *Deforming Polyhedron Search* or the *Nonlinear Simplex Method* [92]. The main idea of the method is

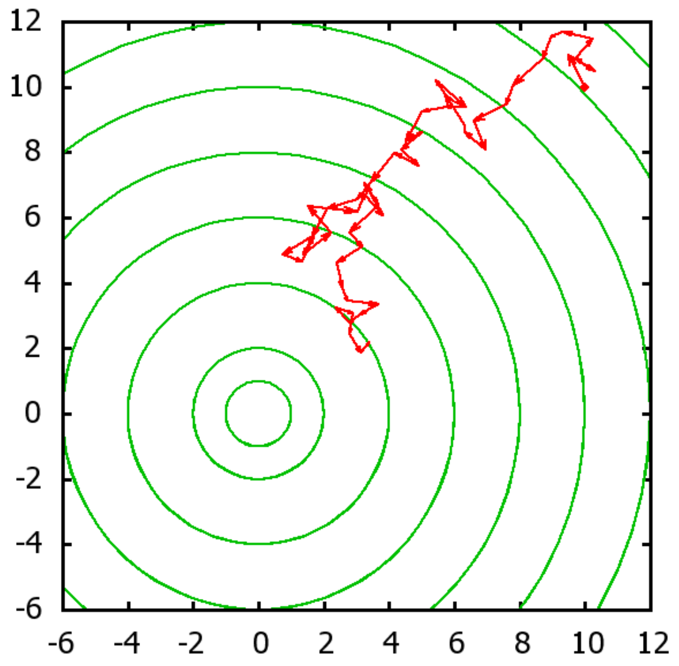


Figure 2.3: *First steps performed by the ANNEALING (*COSY Infinity *built-in optimizer) on the* 2*-dimensional Sphere test function (see section C.1 starting from the initial guess* (10, 10)

to use a "search object" which in this case is a polyhedron with $(v + 1)$ vertices in v-dimensional search space called a simplex. After creation of the initial polyhedron, search proceeds with the search object being transformed and moved in the search space in order to reach the minimum. Objective function evaluations in its vertices are used to measure the performance and to select appropriate transformations and movements. The initial simplex is either generated randomly, or is set as one of the parameters. Note that if it is too small, the algorithm can be trapped in a local minimum. After the simplex is generated, the iterative search process given in Figure 2.4 can be initiated. There is a certain level of flexibility in defining the control flow and conditional transitions, which sprouted different variations of the method, so Figure 2.4 demonstrates only one of those existing variations. Classic values for $\alpha, \beta, \gamma, and \sigma$ — the reflection, expansion, contraction and shrinking coefficients — are 1, 2, 0.5, and 0.5, respectively.

```
0. All vertices are ordered and relabeled according to the corresponding
   function values:
       $f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq \ldots \leq f(\mathbf{x}_{v+1})$,
   centroid of all points except the worst one is calculated:
       $\mathbf{x}_m = \frac{1}{v}\sum_{i=1}^{v} \mathbf{x}_i$.
1. Reflection is performed:
       $\mathbf{x}_r = \mathbf{x}_{v+1} + \alpha(\mathbf{x}_m - \mathbf{x}_{v+1})$.
       If $f(\mathbf{x}_r) < f(\mathbf{x}_1)$, reflection improved the best point,
          go to step 2.
       If $f(\mathbf{x}_1) < f(\mathbf{x}_r) < f(\mathbf{x}_v)$, reflection improved the next worst point,
          $\mathbf{x}_{v+1} = \mathbf{x}_r$,
          go to step 0.
       If $f(\mathbf{x}_v) < f(\mathbf{x}_r) < f(\mathbf{x}_{v+1})$, reflection improved the worst point,
          go to step 3.
       Else reflection failed to improve the worst point,
          go to step 4.
2. Expansion is performed:
       $\mathbf{x}_e = \mathbf{x}_r + \beta(\mathbf{x}_m - \mathbf{x}_{v+1})$.
       If $f(\mathbf{x}_e) < f(\mathbf{x}_r)$, expansion improved reflection,
          $\mathbf{x}_{v+1} = \mathbf{x}_e$.
       Else expansion failed to improve reflection,
          $\mathbf{x}_{v+1} = \mathbf{x}_r$.
       Go to step 0.
3. Contraction is performed:
       $\mathbf{x}_c = \mathbf{x}_{v+1} + \gamma(\mathbf{x}_m - \mathbf{x}_{v+1})$.
       If $f(\mathbf{x}_c) \leq f(\mathbf{x}_r)$, contraction improved reflection,
          $\mathbf{x}_{v+1} = \mathbf{x}_r$,
          go to step 0.
       Else contraction failed to improve reflection,
          go to step 4.
4. Shrinking of the whole simplex around the best point is performed:
    $\mathbf{x}_i = \mathbf{x}_1 + \sigma(\mathbf{x}_i - \mathbf{x}_1), i = 2, \ldots, v+1$.
    Go to step 0.
```

Figure 2.4: *Nelder-Mead method iteration*

The advantage of the simplex is that it can adapt to the objective function surface and thus efficiently control the step size. However, for complicated objective functions $(v + 1)$ points might not be enough to build a good model of the landscape, hence there exist methods that use different "search objects" with more sample points, for example, complex, which contains $2v$ points [136]. The Nelder-Mead method is the last one of the built-in *COSY Infinity* optimizers and is called SIMPLEX. Several initial steps of the SIMPLEX minimization process on the 2-dimensional Sphere test function (see section C.1) starting from the initial guess $(10, 10)$ are shown in Figure 2.5.

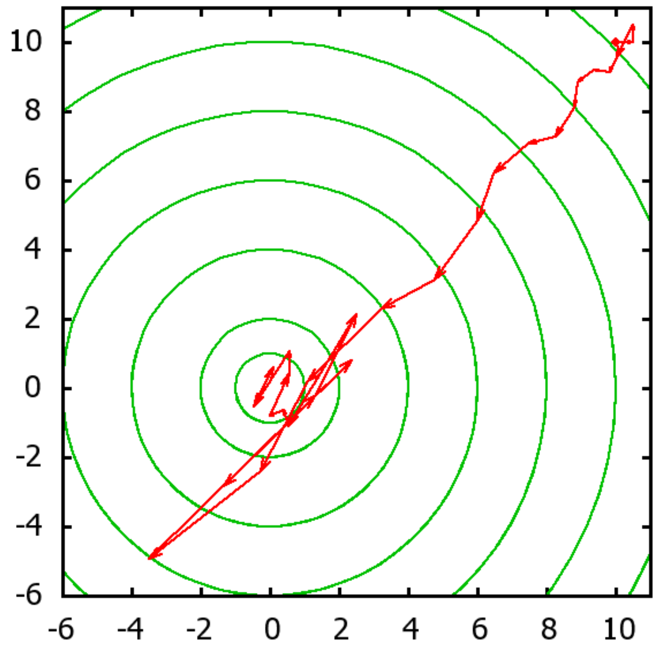


Figure 2.5: *First steps performed by SIMPLEX (*COSY Infinity *built-in optimizer) on the 2-dimensional Sphere test function (see section C.1) starting from the initial guess* $(10, 10)$

2.1.3 Evolutionary Algorithms

Another family of methods that use many points to explore the objective function landscape is inspired by the process of evolution described by Darwin in his revolutionary work “Origin of Species”, first published in 1859 [51]. According to it, the main driving forces of evolution are the variability in living organisms and the natural selection implicitly performed on them by the environment. Over time these forces shape different species to be very sophisticated inhabitants of the environment, i.e. make them fit to it.

If we view an objective function as an environment and points in a search space as organisms evolving to find the best places in this environment (which are for our purposes minima), we can easily sketch a general model of evolution suitable for optimization which is called an *Evolutionary Algorithm* (EA) (see Figure 2.6). Having the evidence of the efficiency of this algorithm in a variety of very well-fit organisms on Earth, there emerged a strong belief that its main principles can be applied to function optimization problems equally successfully.

```
Generate initial population, evaluate fitness
While stop condition not satisfied do
    Produce next population by
        Selection
        Recombination
    Evaluate fitness
End while
```

Figure 2.6: *Evolutionary Algorithm*

Note that EA is actually a meta-algorithm and that all algorithms described earlier can be formulated in this form. For single-point algorithms the population consists of just one individual, selection and recombination are not applicable, the process of producing a next search point can be viewed as a mutation. Multi-start methods differ from the single-point ones only in the size of the population. The Nelder-Mead method (see Figure 2.4) resembles EA more closely: it maintains a population of

$(v+1)$ points and uses both selection (by sorting its points using function values and replacing the worst on each step) and recombination (during reflection a new point is generated by means of other points in the population) to perform the optimization. Expansion, contraction and shrinking here can be considered as different types of mutation.

Because of such generality, it is commonly agreed that the family of Evolutionary Algorithms includes only the ones that directly imitate the processes of evolution and use evolutionary terminology to describe their search strategies. A particularly important distinctive feature of EAs is that the members of the population actively exchange information about the search space. Despite these distinctions, the boundary is still blurry and some EAs, for example the Differential Evolution [171] algorithms, are closer to multi-point direct search methods than to the "true" Evolutionary Algorithms.

It is worth noting that EA does not pose any restrictions on the search space and members of the population, which, multiplied by a variety of different approaches to define fitness, selection, recombination and mutation, leads to a very broad field of applications. Examples include a wide variety of optimization problems: numerical optimization, combinatorial optimization, circuit design, scheduling problems, video and sound quality optimization, control systems [53], image analysis [50], marketing [164] and economics [12], traffic control [35], manufacturing [90], and many others. While EAs do not explicitly guarantee to find even a local minimum, practical applications demonstrate that frequently they are able to find a global minimum or at least produce a practically acceptable solution.

Each of these applications is usually tied to a particular flavour of the Evolutionary Algorithms. Genetic Algorithms (GAs) [78, 79] often encode parameters as strings of bits and modify them with logical operators and thus are better suited for combinatorial optimization, for example for the class of problems equivalent to the famous Traveling Salesman Problem [158]. Genetic and Evolutionary Programming [107] evolve computer programs and are used, for example, to design snippets of code to filter out unimportant events from the flow of the events coming from a detector in high-energy physics. Evolution Strategies (ES) [82] and Differential Evolution (DE) [171] both use real numbers and arithmetic evolutionary operators for continuous function optimization. It is also worth noting rapidly increasing interest in the development of the optimizers mimicking various optimization and search processes of nature: *Particle Swarm Optimization*, *Ant Colony Optimization*, *Tabu Search*, *Cultural algorithm*, etc. [165] and their successful application to many real-world problems.

In this work we consider EAs primarily in the context of the real-valued functions optimization. However, the overabundance of the variations of the Evolutionary Algorithms suitable for the task does not allow us to cover them all. Our primary goal was to find and implement the one that is proven to be robust and efficient, with a default set of parameters effective for many applications and then to assess its applicability to our problems. The description of the Evolutionary Algorithm we implemented is presented in section 2.3.

2.1.4 No Free Lunch Theorems for Optimization

With such a large variety of optimization methods, having described only a few well-known ones, one can conclude that there is no need for new algorithms. However, each method has its own strengths and weaknesses and the variety of optimization problems is much larger than the variety of methods. Thus development actively continues, especially along with the rapid growth of the available computing resources, which is opening the possibility to solve increasingly complex problems.

To combine the best features of different methods and compensate for their weaknesses, combinations of methods are sometimes used. The Levenberg-Marquardt method described earlier, is a heuristic combination of the Steepest Descent and Gauss-Newton methods, and it works very well in practice. Even more sophisticated combinations can be constructed using several direct search methods. Consider, for example, a perfectly viable combination: a multi-start method, sampling points of the search space by using them as initial points for the greedy direct search method combined with the Simulated Annealing to compensate for its greediness. Combinations of rigorous methods with good heuristic methods could significantly speed up their convergence (see section 2.3.6) and increase their robustness (see section 3.3.5). The number of parameters for state-of-the-art methods and for their combinations can be so large and their interactions so complex, that a separate optimization technique to adapt method parameters to the problem might be used.

Much controversy was caused by the so called "No free lunch theorems for optimization" [176] which in rough terms states that the difference in performance of all search algorithms averaged over all optimization problems is negligible. Every method can perform better at certain classes of problems only at the price of performing worse on other classes and this performance depends on the amount of information about the problem incorporated into algorithm, i.e. on fine-tuning of the algorithm for the problem.

While this means that theoretically there is no algorithm that outperforms all other algorithms on all problems, in practice we are always solving a particular problem or a class of problems. Therefore, we can interpret these theorems as an additional reason to find the best matching optimizer for each circumstance. No efficient general purpose optimization method exists, hence there is always a scope for improving algorithms for better performance on particular problems. It must be noted, however, that since human time is getting more and more expensive with computation time becoming increasingly cheap and since we usually need a good solution in a reasonable time rather than the best solution in an optimal time, running an algorithm which is not optimal but capable of finding satisfactory solution in an acceptable time might overall be more beneficial than spending precious human time on finding the optimal method.

2.2 Rigorous Global Optimization

2.2.1 Conventional Interval Methods

While local optimization methods solve only the problem of finding the extremum of the function and in most cases they settle at a local extremum, global optimization methods have an additional goal to achieve. They need to prove that the extremum they find is global in the given search domain. Rigorous, verified or validated global optimization is dealing with the problem of finding mathematically rigorous enclosures for the extrema and the points in which they are attained, accounting also for numerical computational inaccuracies. Note that for this purpose the function under consideration is usually required to be sufficiently smooth on the search domain.

The most straightforward approach to validated global optimization is a version of the well-known general "divide and conquer" strategy: the large problem that is hard to solve as a whole, is divided into a set of smaller problems which are easier to solve (or conquer, following the terminology). For the global optimization this strategy takes the form of the branch-and-bound method that divides the original search domain (1.3.9) into a stack of sub-boxes and conquers them one-by-one, removing each box, proven not to contain a minimum, from further consideration, reducing its size, or dividing into even smaller boxes for subsequent processing. This strategy is applied on each step until the desired accuracy is reached. A rigorous estimate of the upper bound of the minimum, a so called cutoff value, is frequently employed to speed up the elimination. It can be calculated, for example, using interval arithmetic evaluation of the function at some point in the box. Typically, interval calculus is also used to obtain rigorous estimates for the maximum and minimum values of the function on a box.

Algorithms based on the outward rounding interval arithmetic allow one to obtain a rigorous estimate of the global extrema. However, there are several problems connected to their usage, of which the two most important are [26]:

1. *Dependency Problem:* resulting bound is not tight due to a significant overestimation introduced by the complexity of the function and features of the interval arithmetic operations (these features at the same time make them rigorous).

2. *Cluster Effect:* the number of boxes in the stack that are located near the local extrema remains almost constant for a prolonged period of algorithm execution thus slowing down the elimination process.

The field of global optimization is very wide and there exist many different approaches and algorithms which we do not cover in detail in this work. A detailed survey of interval global optimization methods can be found, for example, in [56,101].

2.2.2 Taylor Methods

Efficient Taylor model methods that solve the dependency problem were first developed in [125] for the problem of rigorously bounding a particularly complicated function, which was introduced to solve a practical problem from the field of nonlinear dynamics (normal form defect function optimization is discussed in section 4.2). Detailed coverage of the current Taylor model methods research status, theory, implementation, applications of the Taylor models, and numerous Taylor model-based methods to solve many important problems of scientific computing, can be found in [117]. Here we just outline the basic ideas behind these methods and describe their application to verified global optimization.

For the function f that is $(n+1)$ times continuously partially differentiable on a domain D, a Taylor model of the order n consists of the Taylor polynomial P for f, expanded at the point $x_0 \in D$ up to n-th order and a remainder error bound interval I such that

$$f(x) \in P(x, x_0) + I, \ \forall x \in D. \tag{2.2.1}$$

Given the Taylor model for the function on a search domain, one could perform naïve bounding by merely evaluating the polynomial P in interval arithmetic and then summing results with remainder interval I. Even such a simplistic approach outperforms naïve interval methods or more advanced Centered Form methods applied to a function f directly [117]. However, more sophisticated and efficient range bounders based on the Taylor model representation (2.2.1) were developed and implemented. Two of them, namely the Linear Dominated Bounder (LDB) and the Quadratic Fast Bounder (QFB), outlined in this section and covered in detail in [26, 117], form the core of the Taylor model-based verified global optimization package COSY-GO, implemented in *COSY Infinity* [23].

Linear Dominated Bounder

The *Linear Dominated Bounder* (*LDB*) is based on the observation that in the Taylor model representation (2.2.1) the linear part of the Taylor polynomial P frequently dominates model's behaviour and thus plays the main role in range bounding. This linear part is utilized by the LDB in order to reduce the search domain that encloses extrema. To find the lower bound of M, the minimum of the Taylor model (2.2.1) in D, it performs the steps given in Figure 2.7. All re-expansion errors and point function evaluations from step 3 are accounted for by including them into the remainder error bound interval. Note, that even if there is no linear part in the original Taylor model, it can often be introduced by shifting the expansion point.

```
1. Re-expand $P$ at $c$, mid-point of $D$, to obtain polynomial $P_m$ on the
   centered domain $D_1$.

2. Flip the coordinate directions to make all the linear coefficients $L_i$ of
   the $P_m$ positive so as to obtain the polynomial $P_+$.

3. On the $n$-th step compute the bound of the linear ($I_{\mathrm{L}}$) and nonlinear ($I_{\mathrm{N}}$)
   parts of $P_+ = L + N$ on $D_n$. Then, according to the rules of interval
   arithmetic, the minimum is bounded by $[M, M_{\mathrm{in}}] = I_{\mathrm{L}} + I_{\mathrm{N}}$.
   If possible, reduce $M_{\mathrm{in}}$ using the minimum of the current cutoff value,
   the function values calculated at the left endpoint and the mid-point.

   4. Calculate width of the resulting range: $d = \mathrm{width}(M, M_{\mathrm{in}})$.
   a) If $d < \varepsilon$, desired accuracy $\varepsilon$ is reached, stop. $M$ is a lower bound of
      the minimum.
   b) Else, if $L_i \neq 0$, the domaini containing the minimum is reduced by
      setting
         $\overline{D}_{n+1,i} = \min(\underline{D}_{n,i} + d/L_i, \overline{D}_{n,i})$.
      Perform steps 1 and 2 on $D_{n+1}$ to obtain new $P_+$. Return to step 3
      to perform $(n+1)$-st iteration.
```

Figure 2.7: *LDB range bounding algorithm based on Taylor model* (2.2.1)

Quadratic Fast Bounder

The *Quadratic Fast Bounder* (*QFB*) is designed to work in cases where the linear part of the Taylor model (2.2.1) is not dominant (most importantly in the vicinity of the local minimizer) and it utilizes the quadratic part of P. Range bounding of a quadratic polynomial on an arbitrary interval generally has exponential complexity growth with the dimension and thus can be very expensive. However, obtaining a lower bound of the quadratic polynomial in an isolated neighborhood of the local minimum (which is also an important problem of global optimization) turns out to be a much simpler task. Indeed, in such a neighborhood the Hessian of a function f is positive definite, so the purely quadratic part of the Taylor model (2.2.1) has a positive definite Hessian matrix H. Positive definiteness itself can be tested in a verified way via a common LDL or extended Cholesky decomposition, as demonstrated in [26]. If a purely quadratic part is, indeed, positive definite, QFB provides a lower bound of the Taylor model rather cheaply based on the following observations.

Suppose we obtain the Taylor model (2.2.1) for the given function f on the given domain D. Let H be the Hessian matrix of P and let it be positive definite. We now represent P as

$$P + I = (P - Q) + I + Q \tag{2.2.2}$$

and observe that then the lower bound for $P + I$ is obtained as

$$l(P + I) = l(P - Q) + l(I) + l(Q). \tag{2.2.3}$$

If we now choose Q such that

$$Q = Q_{x_0} = \frac{1}{2}(x - x_0)^{\mathrm{T}} H (x - x_0) \tag{2.2.4}$$

for any $x_0 \in D$, then, since H is positive defined, $l(Q) = 0$ in (2.2.3) and is actually attained at $x = x_0$. By this choice of Q, the remaining polynomial $(P - Q)$ does not contain purely quadratic monomials: it consists only of first (linear), third and higher order terms. If we now choose x_0 to be the minimizer of the quadratic part P_2 of P in D, then, by the consequence of the Kuhn-Tucker conditions, it is also a minimizer of the remaining linear part, so the lower bound (2.2.3) is dominated by the orders ≥ 3 and is thus optimally sharp.

Hence, by choosing x_0 to be sufficiently close to a minimizer of P_2 in D, we can make the contribution of $P_2 - Q$ to the lower bound (2.2.3) sufficiently small. To determine this minimizer we can utilize the fast iterative directional minimization method descending in the direction of $-\nabla P_2$ limiting at the same time the obtained values to remain inside D.

Validated Global Optimization Package COSY-GO

The methods described earlier, combined together, form the basis for the verified global optimization COSY-GO package for *COSY Infinity*. It uses a branch-and-bound scheme to manage the list of boxes that represents the current state of the search space. The core of the package itself consists of two global optimization methods: interval bounding and interval bounding with Centered Forms which is roughly equivalent to the first order TM method (used mainly as an auxiliary tool for simple tasks). The other, most sophisticated, efficient and thus heavily employed method is based on the Taylor model methods described earlier in this section [26,117,122,125]. At every step, the algorithm from Figure 2.8 is applied to each box from the list. Then the search process either stops if the desired accuracy is achieved, or reduces the search space volume by re-splitting it to smaller boxes and inserting them back into a list and continues. An example output of the COSY-GO rigorous minimization of the 2-dimensional Rosenbrock's function (see section C.4) is presented in Figure 2.9.

It demonstrates different methods COSY-GO utilized to eliminate and reduce boxes in order to obtain a rigorous enclosure of the minimum.

```
1. A lower bound is obtained by applying the various available bounding
   schemes sequentially in the order described below. If the obtained lower
   bound is below the cutoff value, the box is eliminated, otherwise it
   is bisected. Each subsequent method is applied only if the previous
   one fails. The following bounding methods are used:

   a) Simple interval bounding of the function $f$.
   b) Naive Taylor model bounding based on the evaluation of the Taylor
      polynomial $P$ in interval arithmetic.
   c) LDB bounding. If it fails, the LDB domain reduction is performed as
      described earlier.
   d) QFB bounding, if the quadratic part of the $P$ is positive
      definite.

2. The cutoff value is heuristically updated using following methods:
   a) The result of the function evaluation at the midpoint of the
      current box.
   b) The linear and quadratic parts of $P$ are utilized to obtain a
      potential cutoff update.
```

Figure 2.8: *COSY-GO Verified Global Optimizer box processing algorithm*

```
# COSY-GO result for Rosenbrock: ID=0, NV=2, NO=5, GORUN=1
#                                  Output level=3
#
#  COSY-GO Computation time     :  0.2002880000000001 sec
#
#  Initial search volume        :   9.000000000000016
#  Remaining volume             :  0.7655209967109666E-19
#
#  Total number of steps        :            119
#
#  Number of boxes eliminated
#   By retained lower bound     :              0
#   By interval lower bound     :             43
#   By naive TM lower bound     :              0
#   By LDB cutoff               :              3
#   By QFB cutoff               :              7
#   By splitting                :             53
#
#  Number of boxes reduced
#   By LDB                      :             32
#   By QFB                      :             23
#    - QFB and no box splitting :             12
#
#  Number of remaining boxes
#   In the master list          :              0
#   In the local lists          :              0
#   Smaller than SIZE           :              1
#
#   SIZE= 0.1000000000000000E-05
#
#  Maximum number of active boxes in the local list : 14
#
#  Enclosure of the minimum
# [-0.2004168360008975E-291, 0.1112491091587934E-026]
```

Figure 2.9: *Output of the results of the minimization of the 2D Rosenbrock's function (see section C.4) by COSY-GO*

2.3 GATool Evolutionary Optimizer

As it was mentioned in section 2.1, Evolutionary Algorithms (EAs) are successfully employed to solve many different optimization problems of science and industry. Their distinctive advantages include:

- relative ease of implementation,
- ability to efficiently find global optima avoiding local ones even in very large search spaces (see section 2.3.6),
- no requirements on the objective function other than the ability to calculate its value at every point of the search space,
- good tolerance to noise (see section 2.3.5),
- ability to work even when the traditional search methods fail.

General interest in the field of EAs is steadily growing. Active research on the development of EAs and their applications has produced a large number of publications; bibliography on Evolutionary Computation as of now contains more than 4000 entries on Evolutionary Computation and related areas [65]. To save space and avoid repeating work we describe only the concepts of Evolutionary Algorithms that essential to the considered optimizer. For more detailed treatment we refer the reader to an excellent introduction to Evolutionary Algorithms in [128].

2.3.1 Principles, Concepts and Building Blocks

A generic scheme of the Evolutionary Algorithm is presented in Figure 2.6. Because of the EA's generality, in order to select or construct a specific EA for the problem or a class of problems, we need to make certain design decisions. Usually the EA design process starts with selecting an appropriate representation of the possible solutions from the search space. Effective encoding of the solutions is the first and one of the most important components of successful EA usage. For the Travelling Salesman Problem [6], it can be a string of numbers denoting the visited cities, for the physical device design optimization it can be an array of the control parameters in the floating point format.

Some algorithms make a distinction between *genotypic* and *phenotypic* representations. Genotypic representation here is an encoding of the solution that has to be decoded to a phenotypic representation before evaluation. Typical example of such encoding is a genotypic binary encoding of the phenotypic real-valued parameters that is frequently employed by Genetic Algorithms [78]. Most frequently used representations include binary encoding, binary gray encoding, integer, real or even standard data structures such as lists, trees, etc.

Each step of the execution, EA works with a set of the representations of the current potential solutions called a *population* (P). Individual members of the population are called *individuals*:

$$P = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}, \tag{2.3.1}$$

where N is the population size. As the algorithm progresses with the search, the population changes. The population on the i-th step of the EA is called i-th *generation.*

The definition of the solution representation is, of course, meaningless without a method to measure the performance of the represented solution. The *fitness* function is used for this purpose (typically fitness is a real, non-negative number):

$$\text{fitness} : P \longmapsto \mathbb{R}^+. \tag{2.3.2}$$

The fitness function serves as the main connection between the objective function and the Evolutionary Algorithm. Its main purpose is to rank individuals according to the optimization goals. Hence it is constructed such that the individuals that are better in terms of the underlying optimization problem have higher fitnesses than the ones that are worse. Typically the fitness is calculated from the value of the objective function via a process called *fitness scaling.* This process can, for example, convert the function values obtained during the minimization process where smaller values correspond to better solutions, to fitnesses in $[0, 1]$ such that a larger fitness corresponds to a better individual:

$$\text{fitness}(\mathbf{x}) = \text{fitness}\big(f(\mathbf{x})\big),\ \mathbf{x} \in P. \tag{2.3.3}$$

Fitness scaling plays an important role in a successful EA application. It can be used to increase or decrease the evolutionary pressure by influencing the selection methods (especially the proportional selection described later in this section) increasing or decreasing the difference in fitness between the members of the population that have different objective function values.

In order to progress in the evolutionary search the selection and reproduction processes must also be designed. The selection process selects individuals for reproduction with a probability that is related to their fitnesses. If we want the evolution to progress, we must select better individuals more often. One class of selection methods is called proportional selection. It includes roulette, stochastic remainder, universal stochastic, deterministic sampling and other methods where individuals are selected with a probability that is directly proportional to their fitness. Another class of methods includes tournament selection where a series of the tournaments among the fixed size (two or more) samplings of the individuals of from the population is held. The fitnesses of all the tournament participants are compared in order to determine the fittest one that is the winner. The important problem here is to choose and tune the selection method so that it exerts an optimal amount of evolutionary pressure in order to keep the population diverse and avoid premature convergence of the algorithm on the one hand but, on the other hand, not to suppress the convergence at all and keep it at a reasonable level.

During the reproduction phase, the next generation is produced from the current one and the results of the selection phase. Two *evolutionary operators* usually employed for reproduction are *mutation*, which produces a new individual (mutant) via modification of the single selected individual, and *crossover*, which uses two or more individuals (parents) to produce a new individual (child). These two operators are typically connected with two main processes of the EA search procedure: *exploration* and *exploitation*. The first of them is a search of the potentially interesting zones of the search space, i.e. zones where the location of the optima is suspected. The second is an examination of these zones in order to find the optima. Mutation is usually responsible for the exploration while crossover is driving the exploitation.

An important concept that influences reproduction is *elitism*. It is an operator employed by the EA algorithm to preserve a certain number of the best members of the current population and transfer them to the next population intact, without mutation or crossover (though they also participate in the selection and can be selected to produce both mutants and children). Elitism guarantees that the best found value of the next generation is not worse than the best found value of the current one and thus is very important for steady convergence. Care must be taken, however, to keep the number of elite members relatively small in order to allow an EA to explore the search space even when some very good potential solutions have already been found and thus to avoid the convergence to a local optimum. Under additional assumption about the run time being infinite (or a maximum number of generations, depending on the selected stopping criteria) the convergence to the global optimum can be proven for the Evolutionary Strategies that are a type of EAs [161].

To start the search we also need a method to generate the initial population (typically by generating random samples uniformly over the whole search space), stopping criteria (typical criteria include maximum number of generations, maximum number of stall generations and maximum run time) and the algorithm parameters (population size N, tolerances mutation and crossover rates, number of elite members, etc.). Only when the design process is completed can the algorithm be used.

Here we should note that despite all their attractive features, Evolutionary Algorithms also have certain weaknesses and complications:

- it is possible to choose a “right” representation but “wrong” genetic operators or to set method parameters to non-optimal values, which results in degraded performance in both speed and quality,
- extensive fine-tuning via trial-and-error and intuition might be required to tune the method for reasonable performance on specific problems,
- no complete theoretical methods to unambiguously select or design the Evolutionary Algorithm for the problem are developed up to date.

2.3.2 Design and Implementation

The algorithm we implemented uses the best features of Evolutionary Strategies (ES), Genetic Algorithms (GA) and Differential Evolution (DE). We named the algorithm and its implementation in *COSY Infinity* system GATool because it most closely resembles the logic of GA (while it is definitely not a classic GA). It is worth noting that the widely popular Matlab scientific computations package [139] includes Genetic Algorithms Toolbox that provides a very similar algorithm in its standard distribution. This helps demonstrate that this algorithm is, indeed, well-tested and proven to be efficient.

From Evolutionary Strategies we adopted the representation of a potential solution as a vector of real numbers, i.e. a vector of problem arguments:

$$\mathbf{x} = \big(x_1, x_2, \ldots, x_v\big)^{\mathrm{T}}. \tag{2.3.4}$$

Then the population members are:

$$\mathbf{x}_i = \big(x_{i1}, x_{i2}, \ldots, x_{iv}\big)^{\mathrm{T}},\ i = 1, \ldots, N$$

and

$$\mathbf{f} = \big(f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_N)\big)^{\mathrm{T}} = \big(f_1, f_2, \ldots, f_N\big)^{\mathrm{T}} \tag{2.3.5}$$

is a vector containing function evaluations for the members of the population and $\underline{f}$ and $\overline{f}$ denote the minimum and maximum function values of the population members correspondingly. Noting the success of the ES and DE (see references in section 2.1) both using such a representation, we suggest that it is more adequate for the optimization of the problems with real-valued parameters than the binary encoding frequently used in GAs. Note that this representation is phenotypic, i.e. a member of the population does not need to be decoded in order to be evaluated.

Since we implemented the algorithm for the minimization of real-valued functions, fitness scaling mapping had to satisfy two requirements:

- smaller function values mapped to larger real fitness values and
- resulting fitness is non-negative for all function values.

For convenience, after the function values are mapped to fitnesses they are normalized to be in the $[0, 1]$ range. Several fitness scaling functions are currently used:

- Linear:
$$\text{fitness}(\mathbf{x}_i) = \text{fitness}_i = \overline{f} - f_i \geq 0 \tag{2.3.6}$$

- Proportional: first the following transformation
$$\text{fitness}_i = \left(\frac{\overline{f} + \underline{f}}{2} - f_i \right), \tag{2.3.7}$$
which effectively rotates the function values around the center of the function values range, is applied. Then, if $\underline{f} < 0$, it is added to the resulting fitness to make it non-negative.

- Rank: function values are sorted in ascending order and then the fitness is assigned to individuals according to the indices of their fitnesses in the resulting array. Practically, the square root of the inverse of the index has demonstrated itself as an efficient formula. This technique ensures that the best members (with smaller indices) are further apart than the worst members (with larger indices). Hence the evolutionary competition between best members is stronger.

From these fitness scaling methods, rank scaling has demonstrated itself as the most efficient. Note that albeit being most computationally expensive of the implemented methods, it is also the most numerically stable since it does not involve the operation of subtraction which can lead to a cancellation effect and thus to the loss of accuracy if the function values are close to machine precision.

The initial population in EAs is generated by producing uniformly random points from the initial box (the most common method). Here we assume that the search domain is given as a v-dimensional box:

$$S = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_v, b_v]. \tag{2.3.8}$$

GATool makes a distinction between the global search box and initial box, which is usually (but not necessarily) contained within the global box or is equal to it. As it was mentioned, the initial box is utilized to generate the initial population, while the global box is employed to control the population for the presence of the outside members. If the elimination mode is on (by default), all members of the population that are initially generated or produced during the search outside of the global box, are killed and then regenerated in the initial box. This strategy can be easily extended on a collections of boxes, both local and global. It can be used if we want to direct the search to certain zones of the search domain and is particularly important for the COSY-GO rigorous optimization package interaction (see section 2.3.6).

The initial population can also be seeded, i.e. initialized with predefined members. For example, if we have reasons to suspect certain zones of the search space for the location of the minima, we might want to pre-generate some members of the population located there in order to direct the search process. In most cases, however, seeding is not recommended because additional pressure can decrease GATool's chances to find a global minima and in the worst case trap GATool in the local minima. It must also be noted that a search space that does not enclose the sought minima with enough tightness can produce similar effects since the uniformly randomly generated initial population can be too sparsely distributed over the search space and thus be to distant from the minima for a successful search.

The process of selection of individuals for reproduction is also of great importance for the algorithm's efficiency. For ease of notation we denote by

$$\xi = \mathrm{rand}\langle a, b\rangle, \tag{2.3.9}$$

where "$\langle$" is either [or (; "$\rangle$" is either] or), a uniformly distributed random number from the corresponding interval: $\xi \in \langle a, b\rangle$. For example $\xi = \mathrm{rand}[a, b)$ denotes a uniformly distributed random number ξ from $[a, b)$. We denote the number of members we need to select as N_{select}. The following frequently used methods of selection are used b GATool:

- *Roulette Wheel:* suppose

  $$S_k = \sum_{i=1}^{k} \text{fitness}_i,\ k = 1, \dots, N, \tag{2.3.10}$$

  $S_0 = 0$ and $t = \mathrm{rand}[0, S_N]$. Then the index ξ such that

  $$S_{\xi-1} < t < S_\xi$$

  denotes the member selected by a random turn of the roulette wheel. Note that here

  $$S_k - S_{k-1} = \text{fitness}_k,$$

  i.e. sizes of the sectors of the wheel are equal to the fitnesses of the corresponding members. Thus members with larger fitnesses have higher chances to be selected. The procedure is repeated N_{select} times.

- *Stochastic Uniform:* suppose partial sums of fitnesses are defined as in the Roulette Wheel selection method (2.3.10). Let h be a selection step:

  $$h = \frac{S_N}{N_{\text{select}}}.$$

  Let $t_1 = \mathrm{rand}[0, h]$ and

  $$t_j = t_1 + (j-1)h,\ j = 2, \dots, N_{\text{select}}.$$

Then

$$t_j \in [0, S_N],\ j = 1, \ldots, N_{\text{select}}$$

and we select N_{select} indices ξ_j of the population members by choosing those that satisfy the relation

$$S_{\xi_j - 1} < t_j < S_{\xi_j},\ j = 1, \ldots, N_{\text{select}}.$$

- *Tournament:* suppose T is a number of the members participating in a tournament. We randomly select T members of the population with equivalent probability for each member to be selected. Suppose the indices of the chosen members form the set I. Then the index of the member that is a result of the tournament selection is determined as

  $$\xi = \arg\max_{i \in I}\{\text{fitness}_i\}.$$

  The procedure is repeated N_{select} times.

From these selection methods Stochastic Uniform stood out as the most efficient and robust in practice. Elite members are selected as the N_{elite} members of the population with the largest fitnesses. Members needed for the evolutionary operators in order to produce the next generation are selected by the chosen selection method. It must be noted that one member of the population can be selected several times.

The next population is produced during the reproduction phase on the basis of the current one and the results of the selection. Elite transfer and two evolutionary operators — mutation and crossover — are employed. While elite transfer is a simple process of copying the best individuals from the current population to the next one (evolution of neutrality, preservation of already found results), mutation (variability, innovation, exploration) and crossover (ancestry, information exchange, exploitation) are more advanced. The number of members of the next generation generated by each of these methods is determined by two parameters: mutation rate and the number of elite members. Mutation rate is a number in the $[0, 1]$ range that determines the percentage of the next population that is generated by mutation. The number of the elite members or elite rate determines the number or percentage of elite members transferred to the next population. The remaining members of the next population are generated by the crossover. Hence the next generation completely replaces the current one with the exception of elite members.

There are two types of mutation used:

- *Uniform:* usually employed by classic GAs [78], it involves random change in one or several genes that occurs with a certain probability. For classic GAs operating on binary codes, it involves flipping the bit value or several bit values. In our case for every member $\mathbf{x}_i$ selected for mutation for each coordinate x_{ij}, $j = 1, \ldots, v$, the random number $\xi_{ij} = \mathrm{rand}[0, 1]$ is generated. If it is less or equal to the predefined coordinate mutation rate $p_\mathrm{c} \in [0, 1]$, the coordinate is replaced with the random value of this coordinate from the search domain:

  $$x_{ij,\mathrm{m}} = \mathrm{rand}[a_j, b_j].$$

  Since the distribution of ξ_{ij} is uniform, the coordinate mutation rate is also a probability for exactly one coordinate of the member scheduled for mutation to change. Hence the case of $p_\mathrm{c} = 0$ corresponds to no mutations at all while the case of $p_\mathrm{c} = 1$ corresponds to a mandatory mutation in every coordinate of every member selected for mutation. Frequently the value of $p_\mathrm{c} = 1/v$ is used so that on average only one coordinate is changed per the mutation of the member. From probability theory we know that the probability of the combination of the independent events (replacement of the certain coordinate) is equal to the product of the probabilities of the mutations of the individual events (replacement of the several coordinates). Since usually p_c is chosen such that $p_\mathrm{c} \ll 1$ and it is shared between all coordinates, the probability for several coordinates of one member to mutate is rapidly decreasing as the coordinate mutation rate raised to the power of the number of coordinates. This also means that the probability for the mutant of any point from S to be any other point from S (i.e. the probability of transition between any two pairs of points in S during the search) is positive if p_c is positive (even if it is very small). This, in turn, means that any point of the search space has a certain probability to be considered during optimization, which is important to ensure that the search for the optimum is global (in the search space). The example of the uniform mutation is demonstrated in Figure 2.10. Here $\mathbf{x}$ is the population member selected for mutation, $\mathbf{x}_{\mathrm{m},1}$ (the ones on the dashed lines) are the mutants with only one coordinate changed, $\mathbf{x}_{\mathrm{m},2}$ are the mutants with two coordinates changed. Here for every $\mathbf{x}$ scheduled for mutation, the probability of mutating to any of the $\mathbf{x}_{\mathrm{x},1}$ is p_c while the probability of mutating to any of the $\mathbf{x}_{\mathrm{x},2}$ is $p_\mathrm{c}^2 \le p_\mathrm{c}$.

- *Gaussian:* usually employed by the ESs [161]. In this type of mutation a difference vector is added to each member selected for mutation in order to produce a mutant member of the new generation. Each coordinate of the difference vector is generated as a random number from the Gaussian distribution with the mean of 0 and a standard deviation equal to half the length of the corresponding range of the search domain S:

$$N(\mu, \sigma^2) = N\left(0, \frac{b_j - a_j}{2}\right).$$

Due to the properties of the Gaussian distribution this means that even though all coordinates are changed for every mutant, most changes in coordinates are small compared to a search range for the corresponding coordinate. We note that in this case the result of the mutation is not guaranteed to remain in S hence additional suppression of these cases might be needed if such requirement is imposed. In our case we do not perform this check since the outside members can still be a valuable source of information about the search space and they will transfer this information during the recombination phase. We also added an option for this mutation type to be adaptive by defining the shrinking schedule for the standard deviation:

$$\sigma^2 = \sigma^2(g) = \left(1 - \alpha \frac{g}{g_{\max}}\right), \qquad (2.3.11)$$

where g is a generation number, $g_{\max}$ is the maximum allowed number of generations and α is the shrinking factor (typically in the $[0, 1]$ range). Of course, adaptive parameters are defined only if the maximum allowed number of generations is set. The mutant produced by the Gaussian mutation can be any point of the search space (even outside of it) with the probability that is decreasing with the growth of the distance from the mutated point. Due to the form of the Gaussian distribution there is a positive probability for any point in the search space S to be generated by the mutation of this type.

Practical experience shows that there is no universally best mutation type and that different problems benefit from different approaches. Some problems might perform better with both types working simultaneously. For Gaussian mutation with adaptation, the shrinking factor must be chosen carefully to allow enough time for exploration in the beginning of the search by generating large jumps but to, allow the algorithm to converge at the end of the search by producing relatively small deviations in the neighborhood of the potential result location.

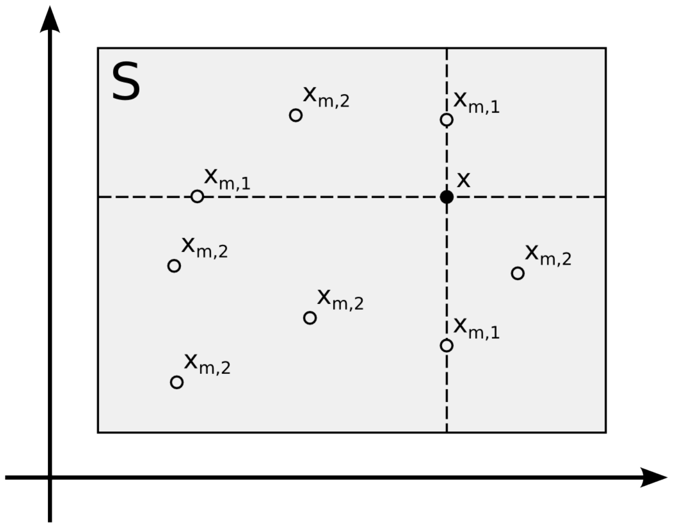


Figure 2.10: *Uniform mutation example:* $\mathbf{x}$ *is a member scheduled for mutation,* $\mathbf{x}_{\mathrm{m},1}$ *are some of the possible one-coordinate mutants (all such mutants are located on one of the two dashed lines),* $\mathbf{x}_{\mathrm{m},2}$ *are some of the possible two-coordinate mutants (can be anywhere in S)*

The process of the generation of new, untested solutions is essential for the success of the method. But the process of the exchange of the information about already explored points of the search space is also an integral part of the algorithm. The evolutionary operator of crossover is designed to perform this task. A usual crossover method employed by GAs is the *Uniform Crossover* (also called *n-point Crossover*). The child is produced from the two selected parents and the process starts from randomly generating n (typically 1 or 2) different random integers from $[1, v]$ to serve as the crossover points. These points divide the member vector into $(n + 1)$ zones. For each of these zones the contributing parent is selected randomly or by some other means. Then the genes in each such zone are copied from the contributing parent into the corresponding zone of the child. Therefore the child have a chance to share a part of its genes with one parent and another part with another. The number of points in n-point crossover determines how parents' genes are being mixed in a child.

Despite the success of this crossover method used in a variety of the different EAs we decided to use another crossover method, which could be viewed as a very simple form of the line search method of the classic continuous optimization methods (step 3, Figure 2.1). This method is usually called *Arithmetic* or *Continuous Crossover*. Suppose two parents are selected for crossover: $\mathbf{x}_{\mathrm{p},1}$ and $\mathbf{x}_{\mathrm{p},2}$. We compare their fitnesses to determine the better fit and the worse fit parents and relabel them as $\mathbf{x}_{\mathrm{p,b}}$ and $\mathbf{x}_{\mathrm{p,w}}$, correspondingly. Then their child is generated by the following formula:

$$\mathbf{x}_{\mathrm{c}} = \mathbf{x}_{\mathrm{p,w}} + \beta(\mathbf{x}_{\mathrm{p,b}} - \mathbf{x}_{\mathrm{p,w}}), \tag{2.3.12}$$

where β is a scaling factor. It can be shared between all coordinates or selected individually for each coordinate (then the formula (2.3.12) is applied coordinate-wise). Since all parents are selected independently, the situation when one population member is chosen for the role of both parents can occur. Particularly often this happens when the population size is small. From (2.3.12) it follows that in such a case the child duplicates this member, which leads to a stagnation of the search process. We implemented a process of the suppression of such situations: in case it occurs, another parent may be re-selected until different parent is chosen or the budgeted number of trials is exhausted.

From (2.3.12) it can be seen that the child is generated on the line connecting two parents and the scaling factor β (typically $\beta \in (0, 2]$) determines its location on this line relative to the better and worse parents. If $\beta < 0.5$ then the child is generated closer to the worse parent, if $0.5 < \beta < 1$, then the child is generated closer to the better parent, between parents, if $\beta > 1$, then the child is generated closer to the better parent, outside the segment of line between parents and the case of $\beta = 0.5$ corresponds to the intermediate crossover when the child is generated in the middle of this segment (in this case, parents' fitness values are effectively ignored). Examples of the different crossover children can be seen in Figure 2.11.

Equivalent notes can be made for per-coordinate scale factors, only coordinate-wise, with the line replaced by a box with parents occupying opposite vertices of its main diagonal (see examples in Figure 2.12). In this case, it is possible, for example, to generate a child that is closer to the worse parent in the first coordinate but closer to the better parent in the second coordinate, or with all coordinates closer to the better parent's coordinates but each one with its own scale. This feature can be useful for the fine adjustment of the algorithm to a problem, if an additional knowledge about its landscape is available.

Since this type of crossover for $\beta > 0.5$ introduces additional pressure to converge (although in practice it does work better) additional randomization can be added to preserve the diversity of the population and avoid the stagnation of the search process:

$$\mathbf{x}_{\mathrm{c}} = \mathbf{x}_{\mathrm{p,w}} + \mathrm{rand}(0,1) \cdot \beta(\mathbf{x}_{\mathrm{p,b}} - \mathbf{x}_{\mathrm{p,w}}). \tag{2.3.13}$$

Random multiplier can be generated once for a member or generated anew for each coordinate. Values of $\beta \in [0.75, 0.85]$ have demonstrated a reliable performance in most practical cases.

It is interesting to note that for the DE optimization approach a very similar method is employed to perform the differential mutation which is an essential feature of the algorithm (there the crossover is of the n-point type). The difference is that in classic DE all three vectors in the right hand side of (2.3.12) are different.

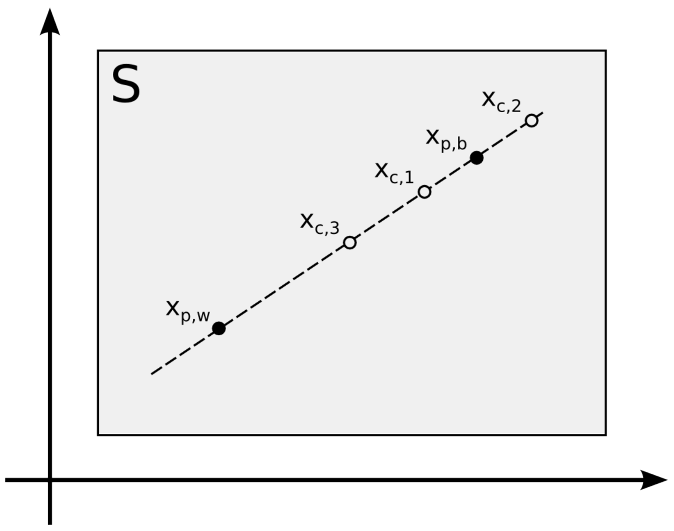


Figure 2.11: *Continuous Crossover examples for the common scaling factor* β*. Points* $\mathbf{x}_{\mathrm{p,b}}$ *and* $\mathbf{x}_{\mathrm{p,w}}$ *are the parents with the better and worse fitnesses, correspondingly;* $\mathbf{x}_{\mathrm{c,i}}$ *for various* i *are the children generated with different values of the scaling factor:* $i = 1$ *corresponds to* $\beta \in (0.5, 1)$*,* $i = 2$ *corresponds to* $\beta > 1$*,* $i = 3$ *corresponds to* $\beta = 0.5$ *(intermediate crossover, values of the fitnesses neglected)*

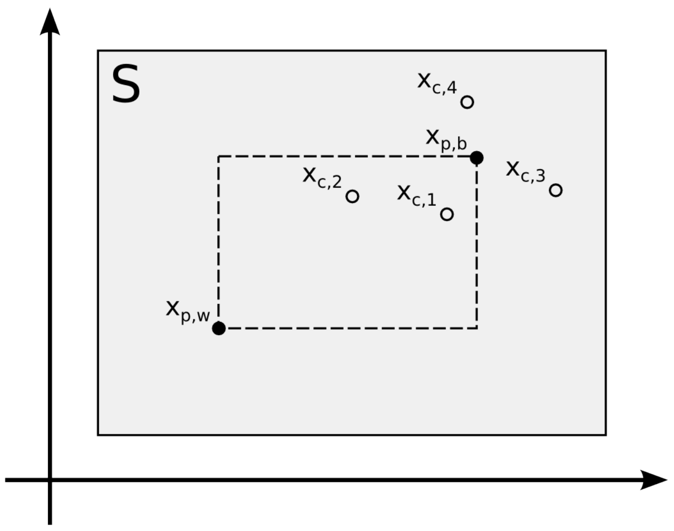


Figure 2.12: *Continuous Crossover examples for the per-coordinate scaling factors* β_i*. Points* $\mathbf{x}_{\mathrm{p,b}}$ *and* $\mathbf{x}_{\mathrm{p,w}}$ *are the parents with the better and worse fitnesses, correspondingly;* $\mathbf{x}_{\mathrm{c,i}}$ *for various* i *are the children generated with different values of the scaling factor for different coordinates. Here the dotted rectangle contains all the children generated with* $0 < \beta_i < 1$*,* $i = 1, \ldots, v$

2.3.3 Statistics, Diversity and Convergence

Several types of statistics are gathered during the search process. Some of them are used to determine if the stopping condition is met and some provide various measures of the algorithm's performance. One important performance characteristic of any EA is its ability to maintain a *diversity* in the population in order to avoid the convergence to a local minimum. The population is considered diverse if it contains members from different parts of the search space and therefore corresponds to different objective function values. Of course the range of the objective function values $\Delta f = \overline{f} - \underline{f}$ depends on the objective function landscape as well as on the search space. But if we have a way to estimate this range on the whole search domain (for example by applying interval arithmetic, see section 2.2), we can use the ratio of the range of function values of the population members to this estimated range as a measure of diversity.

It must be noted, however, that Δf alone is not a good estimate of population diversity. Consider, for example, a population of $(N - 1)$ equal members where only one of them lies very far apart and has a significantly different function value. This population is obviously not diverse yet Δf is large. Instead, another measure of diversity, average distance between the members of the population, should be considered. In our case the distance is Euclidean; for GAs with binary encoding it is Hamming. Since the population can have a large number of members (especially for high-dimensional problems) only a rough estimate of this distance is calculated by sampling 5–10% of the population. Note, however, that there is no information about the "right" amount of the diversity, only experimental observations are available.

However, we are not interested in keeping diversity at all cost all the time. Towards the end of the search we want an optimizer to converge at the sought result.

Convergence here means that the diversity drops down and the search process *stalls*, i.e. the improvement achieved by a next generation remains less than the required tolerance. Therefore we actually want the diversity to decrease but we do not want it to decrease prematurely in order to keep the balance between the exploration of the search space and the exploitation of the potentially interesting zones. Achieving the balance between the fast but possibly, premature convergence and slow but more robust performance is a matter of GATool parameters settings, particularly those of the selection and reproduction. The minimum, average and maximum function values in the current population and their change from generation to generation measure the performance of the search. If the minimum is known, the distance to a minimum both in the search space and in the space of the objective function values can also be used (see section 2.3.6).

An example of the values of these performance characteristics and their behaviour during the search is demonstrated in Figure 2.13. These results are gathered during one run of the GATool on the 10-dimensional Sphere function test problem (see section C.1). Even though GATool is a stochastic search method and thus the results of the different runs are different if the random number generator generates different random numbers, the qualitative picture generally remains the same. Two stages of the search process can be clearly recognized. The first stage, which typically takes 1–20 generations, is exploration, i.e. the search for the zones of interest. It can be observed on the graphs of the statistics from Figure 2.13 as relatively fast in the beginning, slowing towards the end improvement of the minimum, average and maximum function values, as well as the average distance between the members of the population. After the exploration is done, exploitation performs its job of finding the minimum in the zones of interest (typically one zone) found during exploration.

This can be noticed for all statistics as the relatively slow change of the minimum function value while other statistics randomly oscillate aroun an equilibrium state.

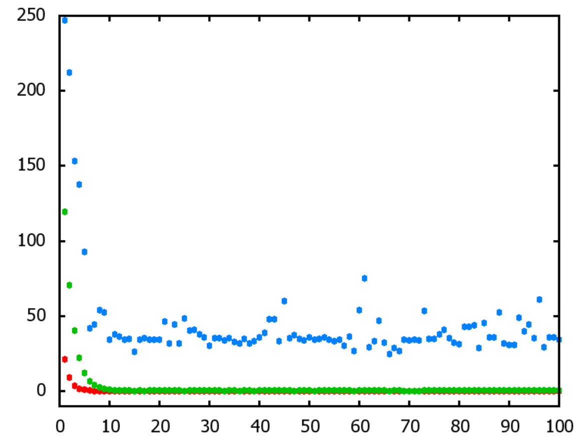
Now that we have considered all the building block of the algorithm, in order to be able to start the search, we must define the stopping criteria. We implemented 4 stopping criteria commonly used by EAs:

- *Maximum stall generations:* maximum allowed number of stall generations, i.e. generations with the improvement to the obtained minimum function remains less than the desired tolerance. Usually it means that the search converged.
- *Desired objective function value:* useful for design optimization when we know we will be satisfied and will not need further search if the values of the parameters that produce the desired value of the objective function are found. If the algorithm preserves the best found values by elitism, this criteria can be less useful since the smaller values of the objective function that can still be found even after the desired value is reached are usually more preferable.
- *Maximum total generations:* maximum allowed number of generations, g_{max}.
- *Maximum run time:* maximum CPU time allowed to be used by a search process.

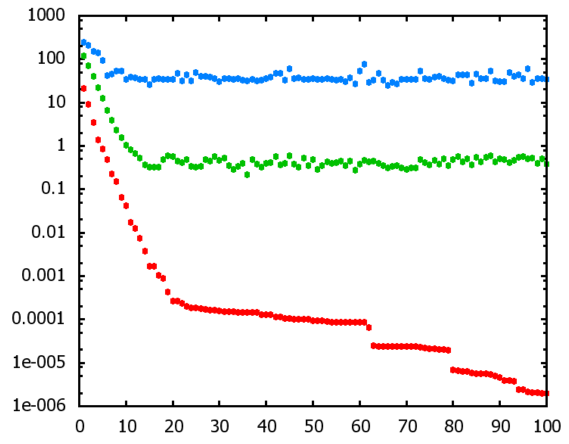
By default GATool uses the maximum number of stall generations and maximum total number of generations stopping criteria.

2.3.4 Summary, Notes on Performance and Parallelization

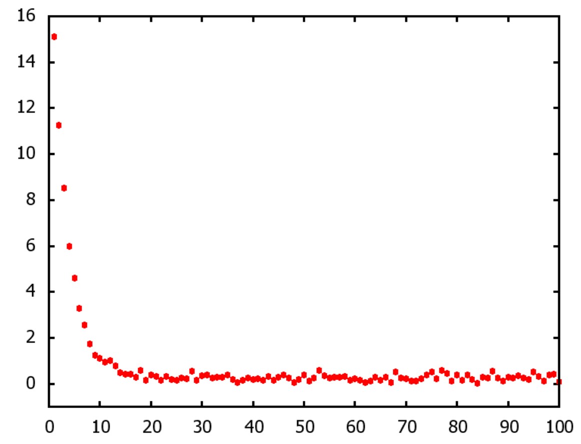
The GATool search algorithm is demonstrated in Figure 2.14, technical details of its implementation in *COSY Infinity* [23], user interface and default values of the



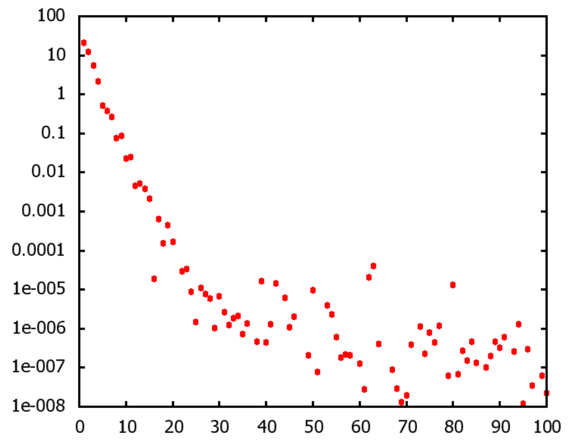
(a) Max/avg/min function values, normal axis



(b) Max/avg/min function values, logarithmic axis



(c) Estimated average Euclidean distance between population members



(d) Min function value improvement (absolute value)

Figure 2.13: *Statistics gathered during one run of GATool on the* 10*-dimensional Sphere function problem (see section C.1). The horizontal axis for all plots is the generation number.*

parameters are described in Appendix B. Performance of the method on standard test problems is assessed in section 2.3.6, on real-life problems from Accelerator Physics — in sections 4.2, 4.1, 4.3, noisy data handling methods are discussed in section 2.3.5, and constrained optimization in pair with the proposed REPA repair algorithm is examined in section 3.3. Results of several example runs on the 10-dimensional Rastrigin function problem (see section C.2) performed with different GATool settings are summarized in Table 2.1. Even though the results obtained from different runs would be different, it can be seen that the results of the optimization highly depend on the choice of parameters.

Since the EAs are heuristic in many ways and since most of their parameters are interdependent in a non-trivial way, a rigorous analysis of the method is too complex and not available for the general case. Rather the statistical approach to performance evaluation on the various test problems is adopted. However, as it is discussed in section 2.1.4, good performance of the algorithm on the arbitrary large class of the optimization problems does not guarantee good performance on problems that do not belong to this class. Therefore we suggest considering these tests only as an estimate of the algorithm's behaviour and always performing tests on the problems that are to be studied by any EA including GATool.

Table 2.1: *Results of one run of GATool on the* 10*-dimensional Rastrigin function problem (see section C.2) performed with different GATool settings. The values of the parameters that are different from the default ones (see Figure B.1) are given in boldface*

Scaling	Elite	Mutation	Crossover	Result	Time
Rank	10	Unif(0.1)	Heur(0.8, 1)	0.593E-02	0h 4m 30s
Rank	10	**Unif(0.01)**	Heur(0.8, 1)	0.196	0h 4m 27s
Rank	10	**Gauss(1, 1)**	Heur(0.8, 1)	3.082	0h 4m 25s
Rank	10	Unif(0.01)	Heur(0.8, **0**)	0.100E-01	0h 4m 43s
Rank	**0**	Unif(0.1)	Heur(0.8, 1)	0.125E-03	0h 4m 29s
Linear	**0**	Unif(0.1)	Heur(0.8, 1)	7.4327	0h 4m 1s

Note that, in principle, GATool can be parallelized almost trivially. If the objective

```
Randomly generate initial population, set predefined members, if any
Calculate objective function values, scale to fitnesses
Update statistics
While any of the stop conditions is not satisfied do
    Perform Roulette Wheel/Stochastic Uniform/Tournament Selection
    Generate next population
        Produce mutants by Uniform/Gaussian Mutation
        Produce children by Continuous Crossover
        Copy elite members
    Replace old population with a newly generated one
    Calculate objective function values, scale to fitnesses
    Update statistics
End while
```

Figure 2.14: *GATool search algorithm*

function evaluation is expensive relative to the GATool's computational expenses, it can be distributed over several (several hundreds or thousands) computers, evaluated there in parallel and then gathered and passed back to the GATool running sequentially for processing. This can be done with no extra effort using the `PLOOP` "parallel loop" construct that was recently added to *COSY Infinity* [103, 104]. In cases where a large population is needed or any other purpose, the algorithm itself can be parallelized by using a co-evolutionary model where several populations co-evolve together starting from different initial populations, possibly using different algorithm parameter values. Each of these populations can be evolved on a separate machine. After certain number of generations (or in predefined time quants) a transfer of members between these populations is performed in order to exchange the obtained information. This model of the parallel EA execution is frequently called the *Island Model*, and the process of exchange, the *Migration Operator*.

Also it should be noted that GATool is a modular algorithm, i.e. it allows easy modification or substitution of its operators. Other types of selection, mutation, crossover, initial population generation and stopping criteria can be added if deemed necessary. Another important note is that GATool is a very general-purpose optimizer with almost no requirements imposed on the considered problem. In fact, the only requirement is the ability to evaluate an objective function value at any point in search space. However, because of this generality, if there exists a method of optimization that is specialized for a particular class of problems and it uses the extra information about this class, then it is likely to outperform than GATool. On the other hand, even in this case it can still be advantageous to build a hybrid algorithm (see section 2.3.6) or at least use GATool to cheaply explore the search space and generate some good starting points for the specialized method.

2.3.5 Noisy Data Handling

All physical devices operate with errors: they could not be manufactured exactly as designed without construction errors and the operational information coming from various detectors contain noise and due to imperfections and limited precision. Here we discuss how such problems can be treated with GATool.

We consider two classes of problems both containing noise in the function values:

- *Static:* the function values contain errors but these errors remain constant across function evaluations:

$$f(x) = f_{\text{true}}(x) + \Delta f(x),\ \forall x.$$

- *Dynamic:* the function values contain errors that change every time the function is evaluated:

$$f(x) = f_{\text{true}}(x) + \text{rand}(-\Delta f(x), +\Delta f(x)),$$

where rand is a random number whose distribution is determined by the considered problem. For simplicity, in this section we consider only uniformly distributed random numbers.

First we consider static noise problems. Since the noise in the function values is usually several orders smaller than the function range, for this class of the problems we can use the same GATool parameter values that we would utilize for the undisturbed function values, possibly averaging results to decrease the contribution of noise. Of course, if the function range is of the same order as the noise itself, then its presence significantly changes the properties of the function and the true value of the minimum is unlikely to be recovered.

Many test functions in Appendix C can be viewed as a sum of the "main function" that determines the large-scale behaviour, main properties and a global minimum, and a "noise function" that adds oscillatory behaviour of a smaller scale, thus introducing many local extrema and making optimization harder. Consider, for example, the Sphere function (see section C.1):

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$$

and the Rastrigin function (see section C.2):

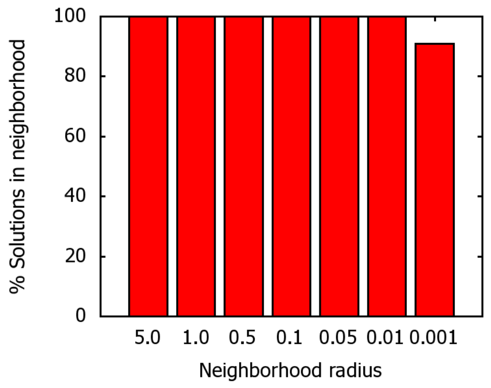
$$f(\mathbf{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) \right).$$

The "main function" here is a Sphere function and the "noise function" is

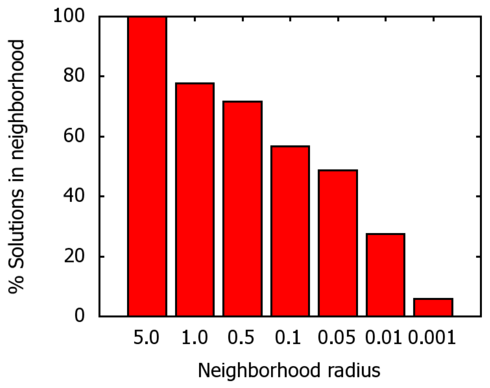
$$10n + \sum_{i=1}^{n} \left(-10\cos(2\pi x_i) \right).$$

Both Sphere and Rastrigin's functions have the same global minimum: $f(\mathbf{0}) = 0$, $\mathbf{x}^* = \mathbf{0}$.

We ran GATool on both test problems in 5 dimensions 100 times using a default set of parameters (see Figure B.1), population size = 10*dimension = 50. The statistical distributions of the solutions by the neighborhoods of the known global minima is presented for both functions in Figure 2.15. The average run time for the Sphere function is 4.09 seconds, for Rastrigin function — 5.22 seconds; quality reduction is clearly noticeable. However, if we increase the population size to 20*dimension = 100 and repeat the simulation, we can bring the quality back (see Figure 2.16), i.e. compensate for the noise at the cose of an increased average runtime increased of 11.92 seconds. Hence, our recommendation for noisy problems with the static noise is to increase the population size and the maximum number of generations allowed to retain the quality at the expense of an increased run time.



(a) Sphere



(b) Rastrigin

Figure 2.15: *Distribution of the results of 100 runs of GATool on the 5-dimensional Sphere (left) and Rastrigin (right) test function problems (see Appendix C) with the default set of parameters (see Figure B.1), population size = 10*dimension, by the ε neighborhoods of the global minimum.*

Dynamic noise in the function values can occur, for example, in the problem of on-line optimization of the control parameters of some complicated physical system: a car autopilot, a self-tuning nuclear reactor or particle accelerator. Here the primary source of the noise is the limited accuracy of the physical measurements. Here our population members are vectors that store sets of control parameters. The objective

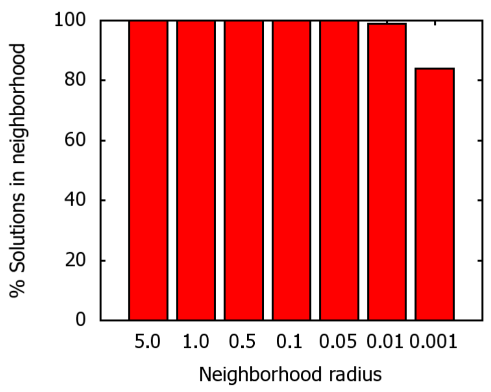


Figure 2.16: *Distribution of the results of 100 runs of the GATool on the 5-dimensional Rastrigin (right) test function problem (see Appendix C) with the default set of parameters (see Figure B.1), population size = 20*dimension, by the ε neighborhoods of the global minimum.*

function is evaluated by taking measurements of the performance from detectors. Different measurements performed with the same set of control parameters will most likely return different values of the objective function, so the noise is indeed dynamic.

The main problem here is that the mechanism of the elite members that preserves the best values found to this step and assures convergence does not work since function values change between generations and the best members might not remain best after re-evaluation. This effect does not let the method converge which can be seen in Figure 2.17. Here we use the sum of the coordinate-wise differences of the best found minimizer with the true one instead of their squares or modules in order to demonstrate that the current minimizer's position relative to the true minimum is changing not just the distance. Note that for the problem without noise, the method converged in the 52-nd generation, while for the noisy problem it reached the maximum number of generations oscillating around the true minimizer without convergence.

The approach recommended earlier for the static problem (increased population size and/or maximum number of generations in evolution) can also be utilized in this case. Another alternative is to use the mean value of the population's best member

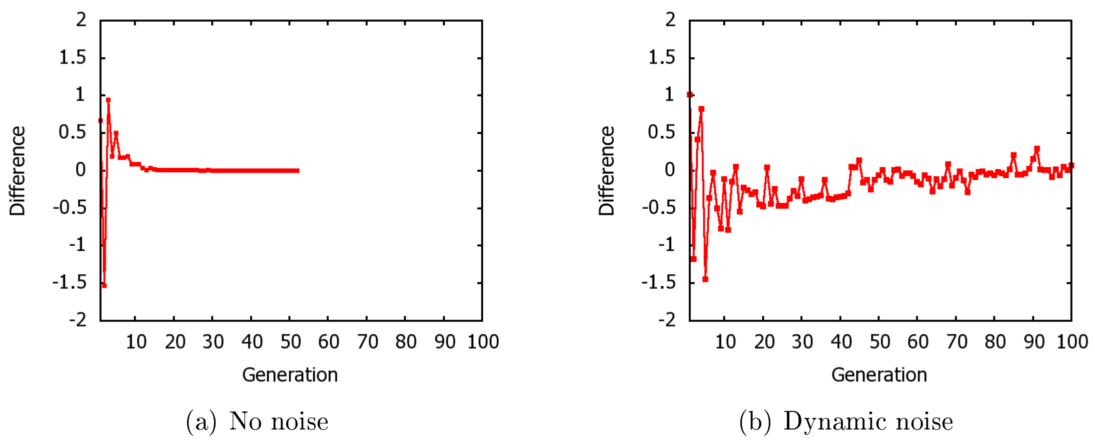


Figure 2.17: *GATool's performance in the 5-dimensional Sphere function problem (see section C.1), population size 50, default set of parameters (see Figure B.1), without noise (left) and with the dynamic noise in the range* $[-1, 1]$ *(right). Generation number versus* $\sum_{i=1}^{v}(x_i^* - x_{i,\mathrm{true}})$*, where* $\mathbf{x}^*$ *is the best minimizer found by GATool and* $\mathbf{x}_{\mathrm{true}}$ *is the true global minimizer (in this case* $\mathbf{0}$*), is plotted.*

averaged over generations as suggested in [102]:

$$\mathbf{x}^* = \overline{\mathbf{x}^*} = \frac{1}{g_2 - g_1 + 1} \sum_{i=g_1}^{g_2} \mathbf{x}_i^*, \quad 1 \le g_1 \le g_2 \le g_{\max}. \tag{2.3.14}$$

As can be seen in Figure 2.17, the best value obtained by GATool oscillates around the true minimum. Hence, by running GATool for a sufficient number of generations and skipping several initial ones (typically $g_1 = 5 \ldots 20$) where the method is searching for the area of interest and then averaging the best obtained minimizer using (2.3.14), we can reduce the effect of noise. Note that the noise distribution must be taken into account when averaging. In our case we used a uniform noise distribution, hence we employ the simple arithmetic mean.

Minimizing the 5-dimensional Sphere function problem with the true minimizer $\mathbf{0}$ (see section C.1) by GATool with the default set of parameters (see Figure B.1), population size = 10*dimension= 50 and the uniform dynamic noise in the range $[-1, 1]$ with a different maximum number of generations we obtain the results summarized in Table 2.2. While GATool's stochastic nature and the fact that noise is dynamic make

Table 2.2: *Euclidean distance from the true minimizer to the current best found objective function value and the objective function value averaged by formula* (2.3.14) *with* $g_1 = 5$*, for the* 5*-dimensional Sphere test problem (true minimizer* $\mathbf{0}$ *see section C.1) minimized with GATool with the default set of parameters (see Figure B.1), population size* $= 10$**dimension*$= 50$ *and dynamic noise in the range* $[-1, 1]$

Generation	Distance to minimizer	
	current	averaged
100	0.18567	0.22973
200	0.17075	0.31166
500	0.13479	0.07508
1000	0.21228	0.06281

the results run-dependent, we can repeat these simulations to statistically observe that the averaged value of the minimizer indeed gets better with an increase in the number of generations GATool is allowed to run. It should be noted, however, that if the noise is dynamic but its level is of the same order as the required precision, the number of generations where the positive effect of the averaging starts to show up can get very large.

2.3.6 Studies on Integration with COSY-GO Rigorous Global Optimizer

The COSY-GO verified global optimizer is based on the box processing algorithm described in section 2.2.2. In the second step of the algorithm (see Figure 2.8), it employs various heuristics in order to update the current cutoff value, which is the best rigorous upper bound for a minimum. This value is then used in the first step to help trimming branches of the search tree when they are evaluated to lie above the cutoff in the objective function range space. The better the cutoff, the more boxes can be eliminated from the relatively expensive processing performed by COSY-GO to maintain the rigor. Thus the overall execution time is also getting better. In the current version, the heuristics used for this purpose are an evaluation of the function value at the middle of each box, a gradient line search, and a representation of the function as a convex quadratic form to update the cutoff value [26].

For small-dimensional problems, even such a naïve approach works fine but for larger problems where days of computational time on thousands of CPUs are needed to finish the search, it can be beneficial to use a better heuristic algorithm. While there exists a plethora of heuristic optimization methods (see section 2.1), we claim that GATool is particularly suitable for this purpose since it is fast, robust, multi-purpose and tolerant to noise (see sections 2.3, 2.3.5, 4.2, 4.1, 4.3). We should note, however, that while generally EAs are capable of finding good estimates of extrema even for very complicated functions, they are not guaranteed to succeed and may require extensive fine-tuning to the problem in order to achieve good results.

In this section we describe the problems that rigorous optimization with COSY-GO is dealing with as the dimension of the problem increases and discuss how the GATool optimization method can be utilized to reduce their negative impact. We also demonstrate some example simulations and overview some ideas on the optimal choice of GATool parameters in order to obtain a good balance between computation time and the quality of the result.

In order to rigorously bound the optimum, COSY-GO represents the current search space as a list of boxes and employs Taylor model methods [4] to eliminate or split them, effectively reducing the search space volume. The package was efficiently applied to rigorous global minimization of the well-known test problems for the global optimizers [26, 117, 122] as well as to solve some real life problems such as spacecraft trajectory optimization [11] and normal form defect function optimization [26,103]. A collection of the boxes generated during the 2-dimensional Rosenbrock's test problem (see section C.4) minimization and the spacecraft trajectory optimization are shown in Figures 2.23 and 2.18, correspondingly (courtesy of Roberto Armellin). Performance of the COSY-GO on some of the test problems listed in Appendix C with increasing dimensionality is summarized in Table 2.3.

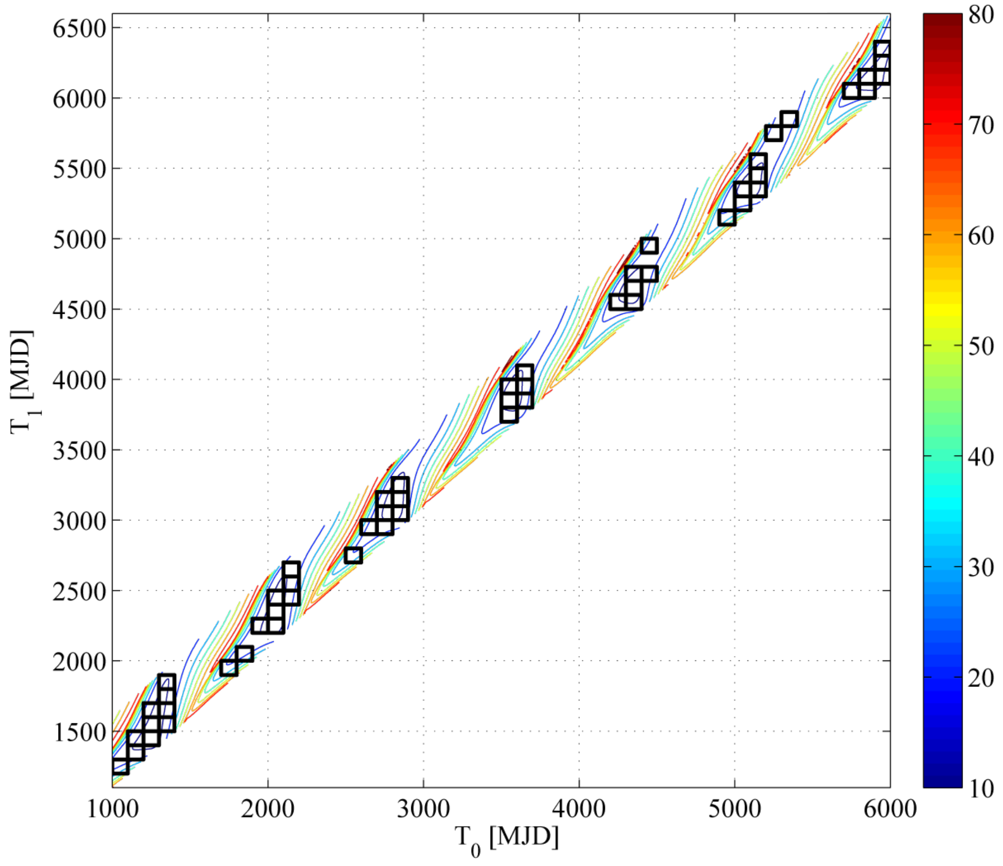


Figure 2.18: *Global optimization of the spacecraft trajectories: pruned search space in the epoch/epoch plane (courtesy of Roberto Armellin) [11])*

It should be noted that even for such a small sampling of the test problems, some of the results demonstrate a visible correlation between the dimension and the minimization time. Some problems scale well with dimension (An), i.e. the time required to find the minimum does not grow as the dimensionality increases, some scale not so well (CosExp), and some scale poorly (Paviani, SinSin). Note that for the An function, computation time is very small and is dominated by the input/output time rather than by actual numerical computations.

Generally, we cannot expect problems to scale well with dimension. Only when they have some special properties and symmetries are such expectations justified. There are several reasons for that. First, as the dimensionality of the problem increases, so does the volume of the search space. Suppose, for simplicity, that our search space is a v-dimensional cube with the length of one side equal to d. Then its volume is given by

$$V = d^v. \tag{2.3.15}$$

so the volume of the space searched for the extrema grows exponentially (see Figure 2.19(a)). Note that if $0 < d < 1$ then the search space volume actually decreases with the dimension, which is the case for the An test function from Table 2.3. As the volume increases, in general the number of the boxes that COSY-GO needs to eliminate to enclose the minima with the desired accuracy increases. The number of local minima that should be reviewed and rejected in order to find the global one, can grow exponentially as well (as it does for most test problems from Appendix C). Hence the complexity of global optimization generally grows exponentially with dimension. Note, that for the local optimizers, complexity often grows polynomially [26].

Another problem, directly connected to the Taylor methods used by COSY-GO, is that the number of monomials in the Taylor expansion of the function grows with the dimension:

$$M = \frac{(n+v)!}{n!v!}. \tag{2.3.16}$$

Here v is the number of variables (dimension) and n is the order of the expansion (NO in Table 2.3). The graph of the number of monomials against the dimensions obtained for different Taylor expansion orders is shown in Figure 2.19(b). As the number of monomials increases, so does the evaluation time for P from the Taylor

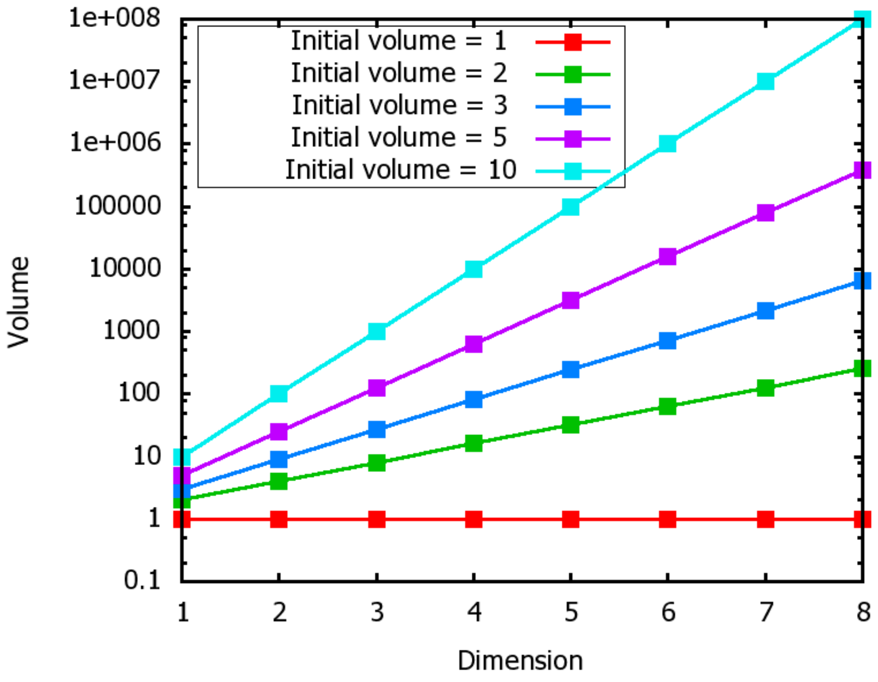
Table 2.3: *COSY-GO performance on the test problems (see Appendix C) with increasing dimensionality; V is the volume of the search space and t is the execution time in seconds. NO is the Taylor model expansion order (see section 2.2). SinSin problem with NO = 8 for* 8 *and* 9 *dimensions takes too long to compute and is thus omitted*

Problem	Parameters	Dimension							
		2	3	4	5	6	7	8	9
Paviani, NO = 8	V	6.30e+1	5.11e+2	4.09e+3	3.27e+4	2.61e+5	2.09e+6	1.67e+7	1.33e+8
	t	0.04	0.19	0.65	7.43	50.93	290.17	2481.88	13524.51
CosExp, NO = 5	V	6.40e+1	5.12e+2	4.09e+3	3.27e+4	2.62e+5	2.09e+6	1.67e+7	1.34e+8
	t	0.03	0.08	0.4	1.19	5.78	24.6	90.79	337.31
SinSin, NO = 8	V	4.00	8.00	1.60e+1	3.20e+1	6.40e+1	1.28e+2	2.56e+2	5.12e+2
	t	0.17	1.37	18.62	395.53	2546.82	7677.42	-.-	-.-
An, NO = 2	V	2.50e-1	1.25e-1	6.25e-2	3.13e-2	1.56e-2	7.81e-2	3.91e-3	1.95e-3
	t	0.02	0.04	0.02	0.05	0.03	0.03	0.01	0.04

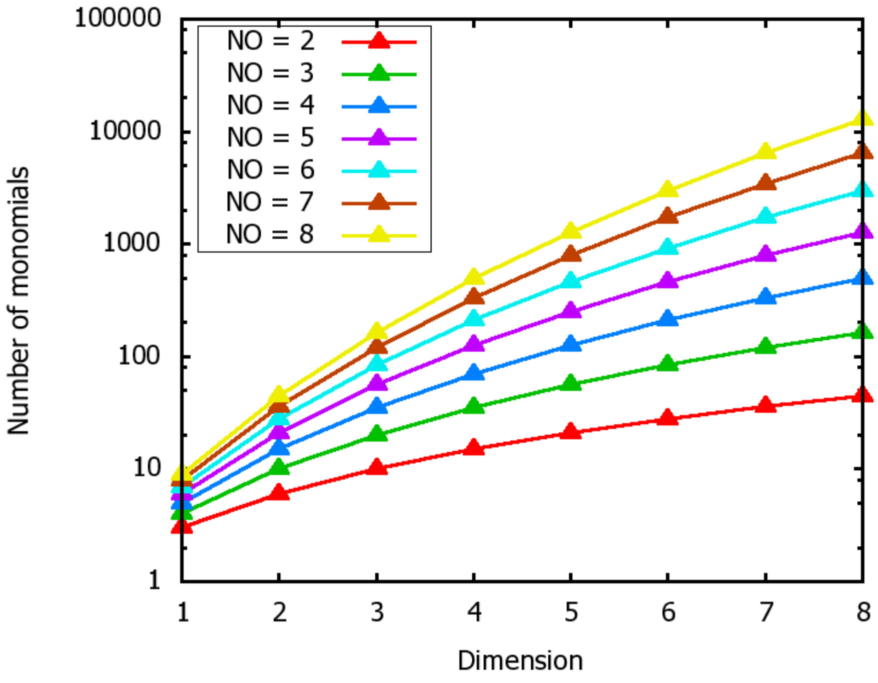
model representation (2.2.1). Since in high-order multi-dimensional Taylor models, many higher-order monomials are frequently zero, an efficient technique to handle this sparsity helps to reduce the impact of this factor. Such a technique is efficiently implemented in *COSY Infinity*: monomials that are equal to zero are not stored and do not participate in calculations [13]. This allows *COSY Infinity* to be memory and computation-efficient and allows it to operate on Taylor expansions up to higher orders with a reasonable amount of the computing resources. A good heuristic cannot help in solving the problem of growing numbers of monomials, but it could help to reduce the bad impact of exponential growth of the search space volume by providing good cutoff values and thus allowing elimination of more boxes on each step of COSY-GO.

Here we consider GATool as a potential candidate cutoff values generator. The rationale for such a consideration is that it frequently succeeds in finding a good upper bound for the global minimum even for high-dimensional and complex problems in a reasonable time (see sections 4.2, 4.1, 2.3) and that it scales well with dimension. We start our examination by applying GATool with the default set of parameters (see Figure B.1) to the set of test problems from Table 2.3 (this table shows the COSY-GO performance).

Since the population size in EA is what largely determines its ability to thoroughly explore a search space, we tested different population size scaling schemes. Here we present results for two different mechanisms: the population size is 100*dimension and the population size is 10*dimension. Results obtained from random runs using these two strategies are summarized in Tables 2.4 and 2.5 respectively. For each problem, three parameters are listed: V is the volume of the search space, t is the execution time in seconds and Q is the quality factor calculated as the difference between the best obtained upper bound and the value of the global minimum (smaller is better, 0



(a) Search space volume for different initial volumes



(b) Number of monomials for different expansion orders

Figure 2.19: *Growth of complexity factors of global optimization with COSY-GO with dimension*

means that the global minimum is found). Note that the Sphere function test problem is not listed for the large population size due to its simplicity.

We note that GATool is by design a stochastic algorithm. Hence, even if it provides good results on an occasional run, there is no guarantee for results to be consistently good each time. However, statistical studies demonstrate its robustness in providing a good upper estimate of the minimum. See, for example, Table 2.6 in this section and Table 4.1 in section 4.1.

Albeit an increase in the population size usually increases the quality of the obtained estimate (by quality we mean its proximity to the global minimum), population size is also one of the main factors that influences the computation time of GATool. It is directly connected to the required number of function evaluations, it increases the execution time for the most fitness scaling algorithms, increases selection time and number of times crossover and mutation must be performed to generate the next population. Finally, it increses the size of the memory footprint, which increases the computation time due to a high price of the memory operations. The important property of the heuristic cutoff search algorithm is not only its ability to find a good result but also its ability to perform this search in a reasonable amount of time. GATool has several features that allow it to perform well even with relatively small population sizes (which also greatly reduces the computation time).

Now we investigate the tradeoff between the quality of the result and execution time in more detail. Between Tables 2.4 and 2.5, there is the difference of almost two orders of magnitude of the execution time can be easily seen, yet for some problems the quality of the result remains the same. For some problems, the quality decrease is lower than the increase in the time of the execution. Comparing the time of the execution of GATool from Table 2.5 with that of COSY-GO from Table 2.3, we see

Table 2.4: *GATool performance on the test problems from Appendix C with default settings (see Figure B.1) and population size = dim*100. V is the volume of the search space, t is the execution time in seconds and Q is the quality factor calculated as the difference between the best obtained upper bound and the value of the global minimum (smaller is better,* 0 *means that the global minimum is found)*

Problem	Parameters	Dimension							
		2	3	4	5	6	7	8	9
An	V	2.50e-1	1.25e-1	6.25e-2	3.13e-2	1.56e-2	7.81e-2	3.91e-3	1.95e-3
	t	11.14	27.01	129.37	239.82	309.71	454.95	712.54	822.86
	Q	1.11e-16	6.87e-5	3.32e-5	9.47e-5	3.52e-4	1.11e-3	8.58e-4	1.85e-3
CosExp	V	6.40e+1	5.12e+2	4.09e+3	3.27e+4	2.62e+5	2.09e+6	1.67e+7	1.34e+8
	t	29.15	78.72	134.63	302.23	391.39	571.32	1065.54	2123.57
	Q	3.99e-15	1.92e-10	9.15e-1	9.54e-1	9.75e-1	9.86e-1	9.92e-1	9.96e-1
Paviani	V	6.30e+1	5.11e+2	4.09e+3	3.27e+4	2.61e+5	2.09e+6	1.67e+7	1.33e+8
	t	34.25	114.64	180.08	366.69	450.42	750.87	998.48	1301.53
	Q	3.69e-6	1.89e-6	6.44e-6	4.04e-5	1.74e-5	8.37e-5	6.28e-4	1.32e-3
Rosenbrock	V	2.25e+2	3.37e+3	5.06e+4	7.59e+5	1.13e+7	1.70e+8	2.56e+9	3.84e+10
	t	11.72	84.13	149.12	238.19	355.25	515.39	714.45	962.20
	Q	4.03e-13	1.01e-2	1.96e-1	3.09e-1	3.47e-1	3.13	4.80	5.89
SinSin	V	4.00	8.00	1.60e+1	3.20e+1	6.40e+1	1.28e+2	2.56e+2	5.12e+2
	t	16.31	12.86	94.91	135.68	237.68	385.06	489.03	685.15
	Q	0.00	4.66e-7	2.71e-7	3.32e-7	2.96e-6	1.91e-6	4.54e-6	2.16e-6
Griewank	V	1.44e+4	1.72e+7	2.07e+10	2.48e+15	2.98e+18	3.58e+21	4.29e+24	5.16e+27
	t	11.08	20.95	55.24	72.78	113.12	183.95	282.63	354.37
	Q	7.39e-3	9.85e-3	1.23e-2	1.47e-2	1.47e-2	2.21e-2	5.42e-2	2.48e-2
Rastrigin	V	1.04e+2	1.07e+3	1.09e+4	1.12e+5	1.15e+6	1.18e+7	1.20e+8	1.23e+9
	t	5.03	18.50	41.94	67.71	87.23	190.56	251.17	328.35
	Q	2.16e-9	6.63e-8	2.80e-7	8.75e-7	2.87e-6	2.29e-5	1.31e-5	8.54e-6

Table 2.5: *GATool performance on the test problems from Appendix C, with default settings (see Figure B.1) and population size = dim*10. V is the volume of the search space, t is the execution time in seconds and Q is the quality factor calculated as the difference between the best obtained upper bound and the value of the global minimum (smaller is better,* 0 *means that the global minimum is found)*

Problem	Parameters	Dimension							
		2	3	4	5	6	7	8	9
An	V	2.50e-1	1.25e-1	6.25e-2	3.13e-2	1.56e-2	7.81e-2	3.91e-3	1.95e-3
	t	0.49	0.83	4.02	10.44	28.16	21.25	51.71	44.62
	Q	1.16e-3	1.27e-2	2.17e-4	1.79e-3	2.23e-3	9.25e-4	1.18e-3	2.50-4
CosExp	V	6.40e+1	5.12e+2	4.09e+3	3.27e+4	2.62e+5	2.09e+6	1.67e+7	1.34e+8
	t	0.86	1.52	4.03	4.44	5.36	12.13	19.40	26.12
	Q	7.13e-1	8.50e-1	9.15e-1	9.54e-1	9.76e-1	9.86e-1	9.92e-1	9.96e-1
Paviani	V	6.30e+1	5.11e+2	4.09e+3	3.27e+4	2.61e+5	2.09e+6	1.67e+7	1.33e+8
	t	0.89	2.33	4.67	8.48	8.65	16.10	21.40	27.24
	Q	1.16e-2	2.45e-2	9.33e-4	1.68e-2	4.29e-2	1.78e-1	7.17e-2	3.02e-2
Rosenbrock	V	2.25e+2	3.37e+3	5.06e+4	7.59e+5	1.13e+7	1.70e+8	2.56e+9	3.84e+10
	t	0.21	0.91	2.25	8.51	5.46	31.57	57.21	113.28
	Q	1.46	0.47	1.00	2.14	4.39	4.70	4.34	5.96
SinSin	V	4.00	8.00	1.60e+1	3.20e+1	6.40e+1	1.28e+2	2.56e+2	5.12e+2
	t	0.27	1.38	2.39	4.40	9.42	15.36	15.63	33.83
	Q	3.62e-2	9.13e-3	1.33e-2	5.12e-3	7.59-3	9.91-4	2.50e-3	2.82-4
Griewank	V	1.44e+4	1.72e+7	2.07e+10	2.48e+15	2.98e+18	3.58e+21	4.29e+24	5.16e+27
	t	0.58	2.41	3.65	6.30	10.51	14.25	17.83	24.09
	Q	1.18e-1	4.27e-2	1.13e-1	1.33e-1	3.02e-2	1.66e-1	1.45e-1	1.20e-1
Rastrigin	V	1.04e+2	1.07e+3	1.09e+4	1.12e+5	1.15e+6	1.18e+7	1.20e+8	1.23e+9
	t	0.57	1.39	4.32	5.99	9.30	12.54	16.64	20.52
	Q	1.00e0	1.08e-1	3.39e-2	4.75e-4	6.53e-2	2.53e-2	7.50e-2	2.04e-2
Sphere	V	1.04e+2	1.07e+3	1.09e+4	1.12e+5	1.15e+6	1.18e+7	1.20e+8	1.23e+9
	t	0.91	0.73	1.51	3.07	8.21	12.36	15.36	19.21
	Q	7.88e-10	2.37e-3	2.81e-5	1.41e-2	1.24e-4	7.85e-5	2.20e-5	6.43e-6

that for poorly scaling, high-dimensional problems (SinSin, Paviani and also complex multi-dimensional normal form defect function optimization discussed in section 4.2, Tables 4.2, 4.3), the time difference between the two methods becomes significant enough for GATool to be a useful heuristic.

It is important to note a relatively low quality of the several estimates provided by GATool. However, the interaction algorithm interaction of COSY-GO is such that the search domain for GATool is being reduced with time by the COSY-GO box elimination process. Domain reduction greatly improves GATool's accuracy, as is demonstrated later in this section. Also note that there might be a different optimal strategy for population size scaling with dimension. This choice is largely heuristic and depends on the problem under consideration. In Figures 2.20 and 2.21, we demonstrate how the execution time and the result quality result change with dimension for different multipliers m from the

$$\text{PopSize} = m \cdot \text{Dimension}$$

scaling strategy. Results are presented for the random run on the Rastringin's function test problem (see section C.2) with the default set of parameters from Figure B.1 and may vary from run to run. However, qualitatively the relation between the quality and the population size remains unchanged; larger population statistically provides better search results at the cost of increased execution time.

The statistical demonstration of the consistency of results is summarized in Figure 2.22. Data is gathered from 1000 runs of GATool on the 5-dimensional Rastrigin function test problem with the default set of parameters (see Figure B.1), population size = 10*dimension. The average run time is 5.22 seconds. Note that while the quality of the result from Table 2.5 is only achieved in 5.8% of the runs, almost 50% of the times, the result is consistent with the general trend demonstrated in Figures

2.20 and 2.21. Also note that since we are studying the quality of the obtained cutoff value, we measure the proximity of the results to the global minimum in the range space of the function. Hence, for some functions, even if the values of the function at some points are in the same sufficiently small neighborhood of the global minimum value, the points themselves may be far from each other.

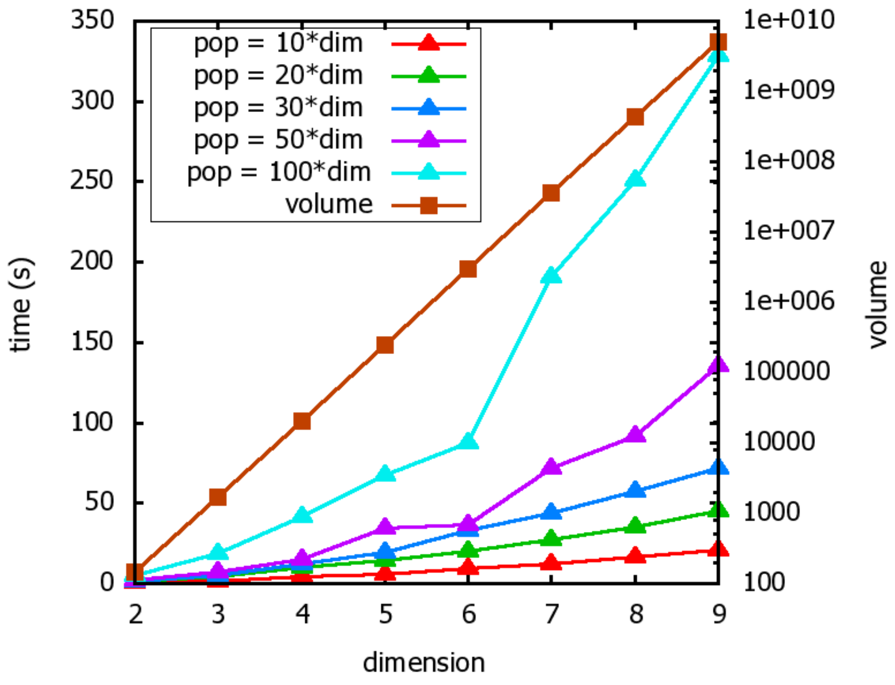


Figure 2.20: *Example of the execution time scaling for different scaling strategies for Rastrigin's function test problem (see section C.2) minimization with GATool. The volume of the search space (logarithmic scale) is shown to better demonstrate scaling issues.*

The interaction mechanism between COSY-GO and GATool could be turned from exploitation into true symbiosis. Suppose both methods start working on the same problem at the same time. In section 2.3.2 it was mentioned that GATool starts its search in a box and ends up with the minimum value of the function in this box that it was able to obtain. By that time COSY-GO, working in parallel on the same box, is likely to have already partitioned it into a set of smaller boxes and eliminated some of them. Using the cutoff value provided by GATool, COSY-GO likely could eliminate

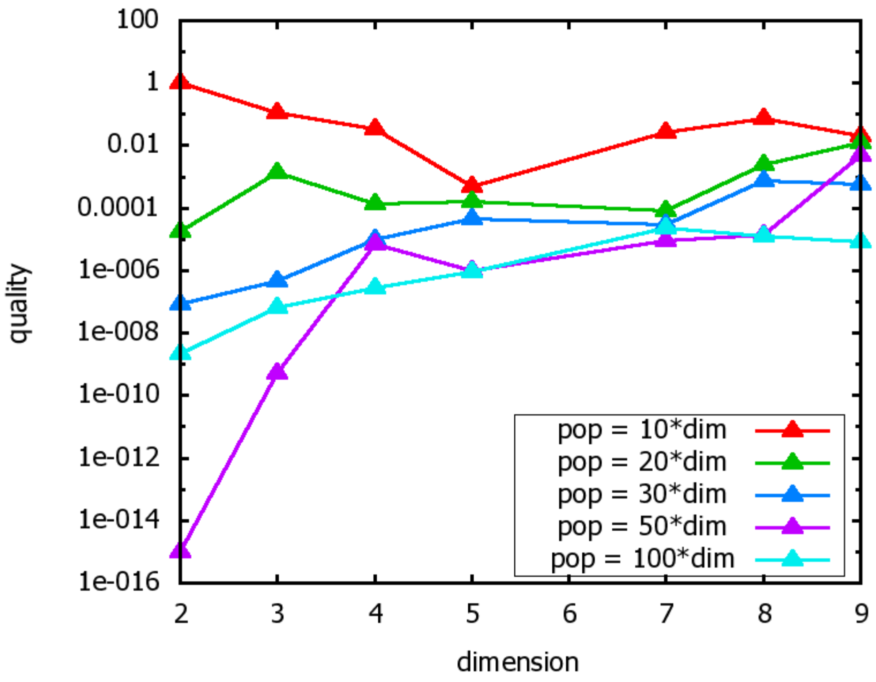


Figure 2.21: *Example of the result quality scaling for different scaling strategies for Rastrigin's function test (see section C.2) problem minimization with GATool.*

ε	% sol.
5.0	100.0
1.0	77.7
0.5	71.7
0.1	56.7
0.05	48.6
0.01	27.4
0.001	5.8

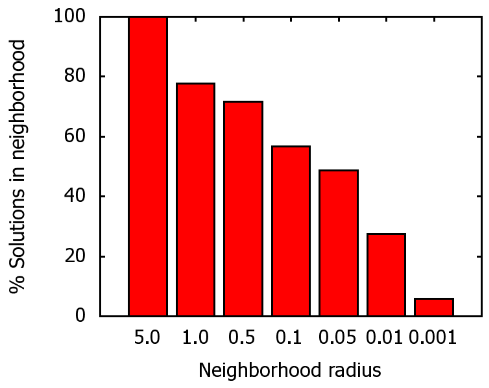


Figure 2.22: *Distribution of the results of 1000 runs of the GATool on the 5-dimensional Rastrigin function test problem (see section C.2) with the default set of parameters (see Figure B.1), population size = 10*dimension, by the ε neighborhoods of the global minimum. Average run time is 5.22 seconds.*

more boxes and send GATool the update of the current search space in the form of its current boxes list. COSY-GO rigorously guarantees that the global minimum resides in one of the boxes from this list. Then GATool runs again using this set of boxes as an initial search space and, in most cases, since its volume is smaller, obtains a better cutoff value. It is then returned to COSY-GO to let it eliminate more boxes. The process continues until the global minimum is bounded by COSY-GO with desired accuracy.

Since the whole method is heuristic, there might be several different strategies of using the information about the search space obtained from COSY-GO. For example, if we have some information about more suspicious and less suspicious boxes, we can run GATool on a set of the more suspicious boxes with the hope that a smaller volume results in a better cutoff. Alternatively, we can run GATool in each of the boxes for a very small number of generations with a very small population size to obtain a value of the minimum for each of the boxes in the collection in a time comparable to that of the midpoint test. Then the cutoff is selected as the best value among the ones obtained. Some other strategies could also be invented possibly using additional information about the problem. Regardless of the choice of the strategy, we claim that the smaller volume of the search space that encloses the sought minimum generally leads to a better upper bound on the minimum obtained by GATool. Since COSY-GO often provides smaller enclosures of the minimum on each step, this would lead to an increased quality of the cutoff provided by GATool, thus these two methods performances form a synergy. The exact strategy formulation and implementation is a direction for future work.

In order to support our hypothesis, we selected the Rosenbrock's function test problem in 10 dimensions (see section C.4) as the worst case scenario for which

GATool alone was unable to provide a reasonable estimate with a small population size (see Table 2.5) and/or high dimensionality. Since it is also one of the COSY-GO performance test problems, we first run it with COSY-GO to obtain the list of boxes it generates during execution (see the example of the 2-dimensional Rosenbrock's function minimization process in Figure 2.23). Then we used several boxes similar to the generated ones with the decreasing volume as initial search domains for GATool and performed the statistical study of the quality of the estimates obtained for each of these initial boxes in 100 runs. The results are summarized in Table 2.6 and in Figure 2.24.

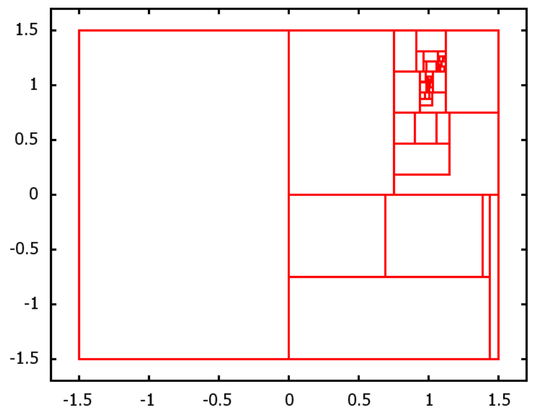
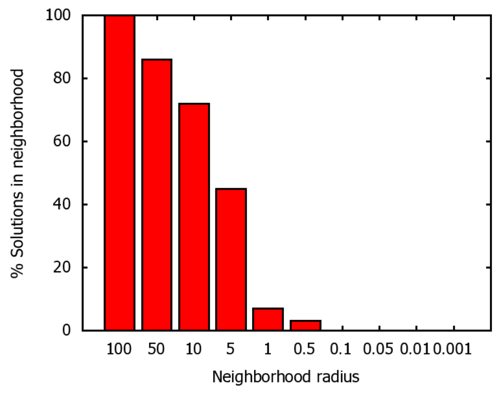
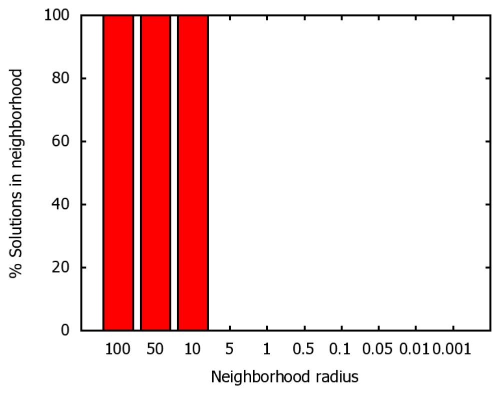


Figure 2.23: *Boxes generated during COSY-GO minimization of the 2-dimensional Rosenbrock's function (see section C.4)*

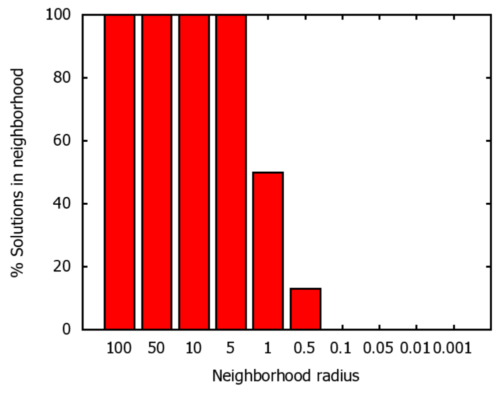
Note that while generally the distribution of results by neighborhood improves as the volume of the search space decreases, the volume itself is not the only factor determining performance. Comparing the results for the search domains of $[-5, 10]^{10}$ and $[-1.5, 1.5]^{10}$, we see that, while for the second box 100% of the results in the second domain lie within the radius 10 around the global minimum, there are no solutions that lie within the radii of 5 or 1. However for the first box, having a volume 7 orders of magnitude larger, 45% and 7% of the results lie within these



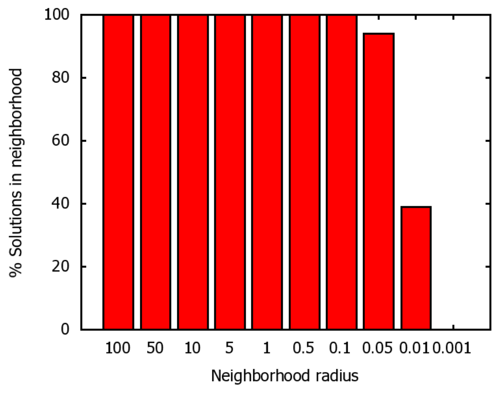
(a) $[-5, 10]^{10}$, $V = 5.67 \cdot 10^{11}$



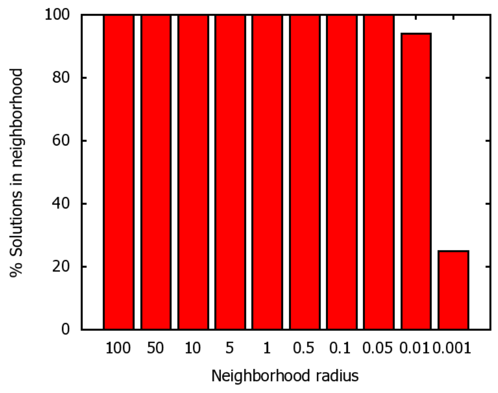
(b) $[-1.5, 1.5]^{10}$, $V = 5.9 \cdot 10^4$



(c) $[0, 1.5]^{10}$, $V = 5.76 \cdot 10^1$



(d) $[0.5, 1.5]^{10}$, $V = 1.0 \cdot 10^0$



(e) $[0.7, 1.3]^{10}$, $V = 0.6 \cdot 10^{-2}$

Figure 2.24: *Distribution of the results of 100 runs of the GATool on the 10-dimensional Rosenbrock's function test problem (see section C.4), with the default set of parameters (see Figure B.1) by the ε neighborhoods of the global minimum for different initial search domains with decreasing volumes. Average run time is ≈ 35 seconds independent of the domain size.*

Table 2.6: *Distribution of the results of 100 runs of the GATool on the 10-dimensional Rosenbrock's function test problem (see section C.4), with the default set of parameters (see Figure B.1) by the ε neighborhoods of the global minimum for different initial search domains with decreasing volumes. Average run time is around ≈ 35 seconds independent of the domain size.*

radius	domain				
	$[-5, 10]^{10}$ V = 5.76e+11	$[-1.5, 1.5]^{10}$ V = 5.9e+4	$[0, 1.5]^{10}$ V = 5.76e+1	$[0.5, 1.5]^{10}$ V = 1.00e+0	$[0.7, 1.3]^{10}$ V = 0.60e-02
100	100	100	100	100	100
50	86	100	100	100	100
10	72	100	100	100	100
5	45	0	100	100	100
1	7	0	50	100	100
0.5	3	0	13	100	100
0.1	0	0	0	100	100
0.05	0	0	0	94	100
0.01	0	0	0	39	94
0.001	0	0	0	0	25

closer neighborhoods, correspondingly. It was observed that for the $[-1.5, 1.5]^{10}$ initial search domain GATool converges on the function values around 7 and is not progressing towards the global minimum.

The reason behind such behaviour is that the different choices of the search domain can reveal or hide different properties of the studied function thereby changing the performance of the algorithm trying to exploit them. Therefore some particular choices of the search domain might work better than the other even if the volume size-based logic suggests otherwise. The problem of the optimal initial box selection is similar to the problem of finding a good initial point for the local optimization methods. Determination of the optimal initial search domain for an arbitrary function, when we generally have no information about its behaviour and the location of the minimum, is in this case a matter of trial and error. However, COSY-GO exploring the search space and eliminating regions that are guaranteed not to contain the sought minimum, provides GATool this additional information about the search domain, thereby greatly increasing the quality of search, as is demonstrated in Table

2.6 and in Figure 2.24. It is also worth noting that in this case the average run time for all domain sizes is around 35 seconds and it is not observed to depend on the volume of the search space.

As an additional note, we need to mention that during the final steps of the COSY-GO execution, the function and parameter values at which the algorithm operates can get very small. In such a case, the precision of the calculations might suffer from floating point arithmetic rounding errors. Hence it is also important for the cutoff search method to be numerically stable. As mentioned in section 2.3.2, GATool with the Rank selection method does not perform any numerically unstable operations. Hence it does not add numeric instability to the problem.

2.4 Conclusions

We reviewed the commonly used methods of unconstrained optimization and described the implemented GATool Evolutionary Algorithm for unconstrained continuous optimization in detail. We assessed its performance in terms of computation time and the quality of the obtained result, studied the tradeoff between the computational resources needed and the resulting quality GATool provides. We discussed GATool's performance in the presence of static and dynamic noise, suggested useful strategies of performance tuning for the EA-hard problems and demonstrated their usefulness on examples. We justified the choice of GATool as a heuristic method to generate cut-off values for COSY-GO rigorous optimization package, outlined the scheme of their interaction, and presented sample runs and statistics that support these choices. We demonstrated that the quality of the result increases as the information about the search domain is refined, which is an essential feature for integration with the box elimination scheme of COSY-GO. Full implementation of the combined GATool and COSY-GO algorithm is a matter of technical details of integration and is a promising direction of future research.

CHAPTER 3

Constrained Optimization

3.1 Challenges in Constrained Optimization with Evolutionary Algorithms

As noted in section 2.1, EAs are successfully applied to problems for which conventional methods are not applicable or fail, and they have proven themselves useful for many of such real-life problems. The issue here is that Evolutionary Algorithms were not originally designed to handle constraints. Even though unconstrained EAs had already demonstrated themselves to be very efficient general-purpose optimizers, the ability to handle constraints would significantly increase their range of applications and help in solving many important optimization problems.

This motivation drove the development of a large number of different approaches for constraints handling for EA and their successful usage in a number of different constrained optimization problems. In this section, we give just a short review of the most commonly used constrained optimization methods for EAs. Necessary modifications are made to the way they were originally presented to make the review unified.

For more methods, detailed descriptions, critique and comprehensive bibliography on the topic, we refer to [129], [42], and [128].

There are certain challenges in adapting EAs for constrained minimization. For a general EA optimizer, the only operation that connects an algorithm with a problem is the evaluation of an objective function. This operation alone then serves as a basis for the fitness evaluation of the population members (see section 2.3) and constraints are ignored. Suppose now that we are implementing an EA optimizer for constrained problems and have started from producing the initial generation demonstrated in Figure 3.1. Here the dots represent members of the population, the cross represents the sought feasible minimum, and S and F denote the search space and feasible set correspondingly.

Before we proceed further, we need to design the method to handle constraints and this poses several important problems. As can be seen, the population contains both feasible and unfeasible members positioned at different distances from the solution. Since we are interested in a feasible minimum at the end of the search, there is an obvious strategy to completely eliminate unfeasible members from further consideration. However, this strategy is too naïve since keeping unfeasible members in a population might be beneficial for the whole process as demonstrated in Figure 3.1. Here the arithmetic crossover (see section 2.3) between unfeasible points a and c or between feasible point d and unfeasible point b would likely produce a new member that is much closer to the optimum than the one produced by a crossover between feasible points f and d. Hence, keeping unfeasible members might increase the probability of success and speed of convergence.

If unfeasible members are allowed to remain in a population, the questions of comparison between feasible and unfeasible, and between two unfeasible members, arise during the fitness calculation. Repairing unfeasible members in order to make them feasible also seems like a worthwhile approach. Increased number of factors to deal with and a variety of possible modifications to the algorithm that can be made to handle constraints greatly increase the diversity of various approaches to constrained EAs.

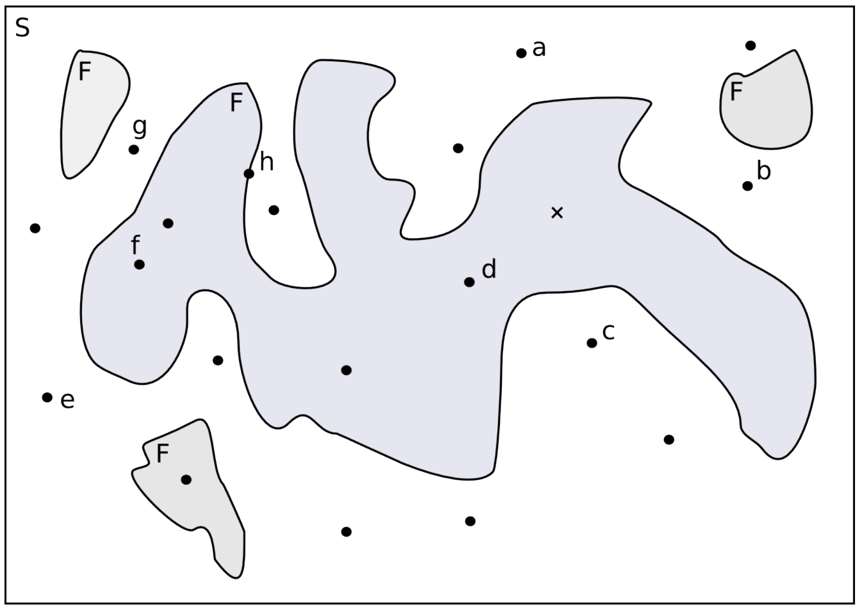


Figure 3.1: *Example of the generation produced by EA for a constrained optimization problem. Here points represent members of the population, the cross represents the sought feasible minimum, S is a search space, and F is a feasible set*

In the general Evolutionary Algorithm from Figure 2.6 we can pick out several operations to modify for constrained optimization: fitness evaluation, selection, genetic operators of recombination and mutation and reproduction process. Combinations of approaches and various heuristics are also possible and, in fact, are widely adopted. Also popular are co-evolutionary techniques where several populations are evolved using different fitness evaluation methods and/or different genetic operators. Such an approach can be viewed as a higher order meta-method that combines several EAs.

3.2 Overview of the Methods

In this section we describe various approaches and their variations in arbitrary order trying to cover them from the simplest, most widely adopted and general methods to the novel, more sophisticated, and more problem-specific algorithms. Note that EAs are very heuristic optimization methods and many of them are combinations of several techniques which makes them hard to classify. So for the reason of unification and for the sake of simplicity of comparison we mostly cover only the main ideas introduced in these methods. Their implementation details, testing and applications can be found in references given along with descriptions. As a last note before starting the review, we have to add that examples of successful applications of constrained EA to real-world problems from various fields are numerous and could be found, to list a few, in [36, 37, 45, 55, 59, 80, 99, 112, 149].

3.2.1 Killing

Perhaps, the most obvious and frequently used approach to constraint handling in EAs is to eliminate unfeasible members of the population. Usually elimination is performed after genetic operators are applied but before fitness evaluation. Replacements are then generated using the selected new members generation method (usually uniformly distributed random points from S, see section 2.3). To allow some unfeasible members in population for the reasons described earlier, members are often eliminated non-deterministically, with certain probability that increases with the amount of constraint violation (usually estimated by some combined penalty function, see section 3.2.2). To increase the number of the feasible members, the elimination-regeneration process can be repeated several times or until a certain percent of the population members becomes feasible. Note, that in its simplest form, i.e. when all the unfeasible members are eliminated independently of the amount of the constraint violation, this method uses minimal amount of information about the problem and thus is expected to be inefficient.

From the description it is logical to suggest that the killing method performs reasonably well only in the cases where the ρ factor is large [42]. Its practical usage demonstrated that this is, indeed, true and that its performance is far from acceptable (see section 3.3.6) only the cases of the small ρ, i.e. when the feasible set is significantly smaller that the search space [133], which frequently happens when constraints are hard to satisfy. This method is particularly inefficient for problems where the global minimum is attained at the point where some constraints are active, i.e. hold as equalities, because they are hard to satisfy. This observation supports the claim that limiting the EA search to only a feasible space may reduce its performance since in this case EA is omitting the information about a search space provided by already generated unfeasible individuals.

3.2.2 Penalty Functions

The penalty functions paradigm was not invented specifically for EAs. Rather it was suggested as a general numerical method applicable to constrained optimization problems. Its basic idea is to transform the original constrained minimization problem (1.3.1), (1.3.3) into an equivalent unconstrained minimization problem. Here equivalence means that the feasible minimum of the original constrained problem is a minimum of the resulting unconstrained problem or at least is acceptably close to it.

This transformation is performed via a set of so-called penalty functions

$P_j(h_j(\mathbf{x})),\ j = 1, \ldots, n$ corresponding to a set of constraints. Here penalty function P_j calculates the amount of penalty assigned to a vector $\mathbf{x}$ for violating j-th constraint. Utilizing those functions the problem of constrained minimization (1.3.7), (1.3.8) could be transformed into an unconstrained multi-objective minimization problem

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in S} \boldsymbol{\Phi}(\mathbf{x}), \tag{3.2.1}$$

where $\boldsymbol{\Phi}(\mathbf{x}) = \big(P_1(h_1(\mathbf{x})), P_2(h_2(\mathbf{x})), \ldots, P_n(h_n(\mathbf{x})), f(\mathbf{x})\big)^{\mathrm{T}}$ that could be solved by multi-objective optimization techniques. It could also be converted even further to an unconstrained single-objective minimization problem

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in S} \varphi(\mathbf{x}), \tag{3.2.2}$$

where $\varphi = \varphi\big(\boldsymbol{\Phi}(\mathbf{x})\big)$ is the function that combines the original objective function and penalty functions into a single objective function. Usually penalty functions are chosen such that $\|\varphi(\mathbf{x}) - f(\mathbf{x})\| \longrightarrow 0$ as $\mathbf{x} \to F$. Function φ also has to be balanced to guide the search process to a feasible set F and hold it there, but not to interfere with the search of the minimum inside F. Care must be taken to achieve this balance in terms of the influence of the original objective function and penalties on a combined function φ. In case penalties are dominating a value of φ, the pressure to produce feasible points might prevent the algorithm from finding an optimum. In the opposite situation, i.e. if the original objective function dominates in calculating the value of φ, the optimization result tends to be optimal but unfeasible and thus useless.

A variety of methods to define penalty functions for Φ, combine them and the original objective function into a function $\varphi(\mathbf{x})$, produced a large number of different constrained minimization methods. Nevertheless, since different problems have different properties of the constraint functions sets, there seem to be no universally optimal penalty function definition strategy, similarly to a nonexistence of the universally best general optimizer as discussed in section 2.1.4. The optimal approach should instead be selected and fine-tuned based on knowledge about the posed problem. It should also be noted that multi-objective optimization problems are generally harder to solve (see section 2.1), so it is often more desirable to convert a constrained problem into a single-objective unconstrained problem (3.2.2) by choosing appropriate $P_1, P_2, \dots, P_n, \varphi$.

The most frequently used method to define the combining function φ is a linear combination of the individual penalties:

$$\varphi(P_1, P_2, \dots, P_n; f) = \sum_{k=1}^{n} w_j P_j + w_{n+1} f, \tag{3.2.3}$$

where w_j are freely chosen weight constants. Under this choice of φ the constrained optimization problem (1.3.7), (1.3.8) is transformed into an unconstrained optimization problem (3.2.2). Since w_{n+1} is a weight coefficient of the objective function in the original constrained problem, usually chosen to be unity for simplicity. The objective function then assumes the form

$$\varphi(\mathbf{x}) = f(\mathbf{x}) + \sum_{j=1}^{n} w_j P_j(h_j(\mathbf{x})). \tag{3.2.4}$$

For a general-purpose optimizer in cases where no information about a problem is available, all weight coefficients for penalties are usually set to unity, at least initially. Since in practice constraints and thus penalty functions often have different ranges of values, weight coefficients w_j can be selected to either normalize penalty values to balance their influence on the combined objective function or to increase the relative impact of some constraints if they are known to be harder or more important to satisfy.

Historically, the penalty function method applied to nonlinear programming was proposed by Fiacco, McCormick and Zangwill [72,127]. Their proposal was to replace an original constrained optimization problem with a single-objective unconstrained optimization problem via φ from (3.2.4), where $w_j = r > 0$, $\forall j$, r is an arbitrary constant. Then by carefully choosing penalty functions it can be guaranteed that the search process applied to the unconstrained optimization problem (3.2.2) converges to a feasible minimum. Two most frequently used types of penalty functions that satisfy these requirements are inverse:

$$P_j(z) = -\frac{1}{h_j(\mathbf{x})} \tag{3.2.5}$$

and logarithmic:

$$P_j(z) = -\log(-h_j(\mathbf{x})), \tag{3.2.6}$$

$j = 1, \ldots, n$, so substituting them to (3.2.4) we get

$$\varphi(\mathbf{x}) = f(\mathbf{x}) - r\sum_{j=1}^{n} \frac{1}{h_j(\mathbf{x})} \tag{3.2.7}$$

and

$$\varphi(\mathbf{x}) = f(\mathbf{x}) - r\sum_{j=1}^{n} \log\big(-h_j(\mathbf{x})\big) \tag{3.2.8}$$

correspondingly.

Note that the penalty part of the objective functions (3.2.7) and (3.2.8) demonstrates fast and unbounded growth if one of the constraint functions $h_j(\mathbf{x})$ approaches zero. Therefore it creates an infinite barrier on the boundary of the feasible set in the objective function landscape and thus prevents iterative unconstrained minimization methods from stepping outside of the feasible region once they start their search inside it. Also note that the influence of the penalties falls off rapidly as we move from the boundary. So if we choose the parameter r so that the unconstrained minimum of the combined objective function (3.2.7) or (3.2.8) is reached at the feasible minimum of the original objective function, we can approach the original constrained optimization problem with most of the unconstrained minimization methods.

In practice, this method is applied by solving a sequence of the unconstrained optimization problems (3.2.7) or (3.2.8) with different values of the parameter $r = r_k$ for $k = 1, 2, \ldots$ such that $r_k \to 0$ as k increases. This approach is called *Sequential Unconstrained Minimization Technique* (SUMT) and is proven to converge under certain assumptions on the objective function, the constraint functions and the sequence $\{r_k\}$ [147]. Penalty functions of the barrier type are called *barrier functions*, so the method is also called *Barrier Functions Method.*

The disadvantages of this approach are its computational expensiveness and that in order to start the search it requires a feasible initial point which is frequently hard to find for all but the most trivial problems. Penalty functions of the barrier type are also called *interior penalty functions* since they prevent optimization methods from considering any unfeasible members during the execution, thereby pushing them to search in the interior of the feasible region. *Exterior penalty functions* allow unfeasible members to be considered during the search process but assign them a penalty that generally grows with their distance from the feasible set. Usually exterior penalty functions are such that $P_j = P_j(z) \geq 0$, $z \in R$, $j = 1, \ldots, n$ and defined in the following way

$$P(z) = \begin{cases} 0 & z \leq 0 \\ \text{penalty(z)} > 0 & \text{otherwise} \end{cases} . \qquad (3.2.9)$$

The most frequently used penalty functions of this type are from the power penalty family:

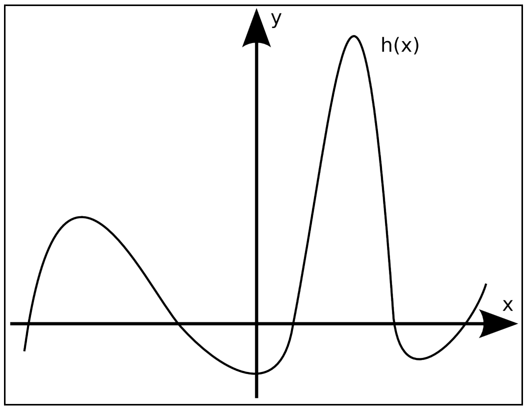
$$P^a(z) = \begin{cases} 0 & z \leq 0 \\ z^a & \text{otherwise} \end{cases} = (\max\{0, z\})^a, \qquad (3.2.10)$$

where $a = 0$, 1, and 2 are most frequently used.

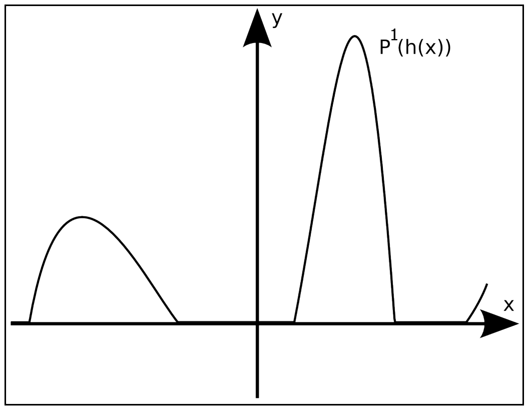
If we then substitute the value of the constraint function into a penalty function of the type (3.2.9)

$$P_j(h_j(\mathbf{x})),$$

we obtain a non-negative penalty assigned to a vector $\mathbf{x}$ for not satisfying the j-th constraint or zero if the j-th constraint is not violated (here index j of the penalty function is given because generally penalty functions could be selected separately for each constraint function). Power penalty functions (3.2.10) use a violated constraint function value at the unfeasible point raised to the a-th power as a penalty (see example of the power penalty for $a = 1$, one-dimensional problem in Figure 3.2).



(a) Inequality constraint function



(b) Power penalty for inequality constraint function

Figure 3.2: *Example of the one-dimensional inequality constraint function and corresponding power penalty function of the type* (3.2.10)*,* $a = 1$*.*

In order to demonstrate how penalty function values correlate with the distance of the unfeasible $\mathbf{x}$ from the feasible set F we use the simple constraint

$$h(\mathbf{x}) = \|\mathbf{x}\|_\infty - 1 \leq 0, \tag{3.2.11}$$

where

$$\|\mathbf{x}\|_\infty \overset{\text{def}}{=} \max_{i=1,\ldots,v} |x_i|$$

is the infinity norm of $\mathbf{x}$. This constraint defines F to be the v-dimensional cube centered at the origin of the coordinates, with the length of the side equal 1. Colormaps of the power penalties (3.2.10) with $a = 0, 1$, as well as the actual Euclidean distance

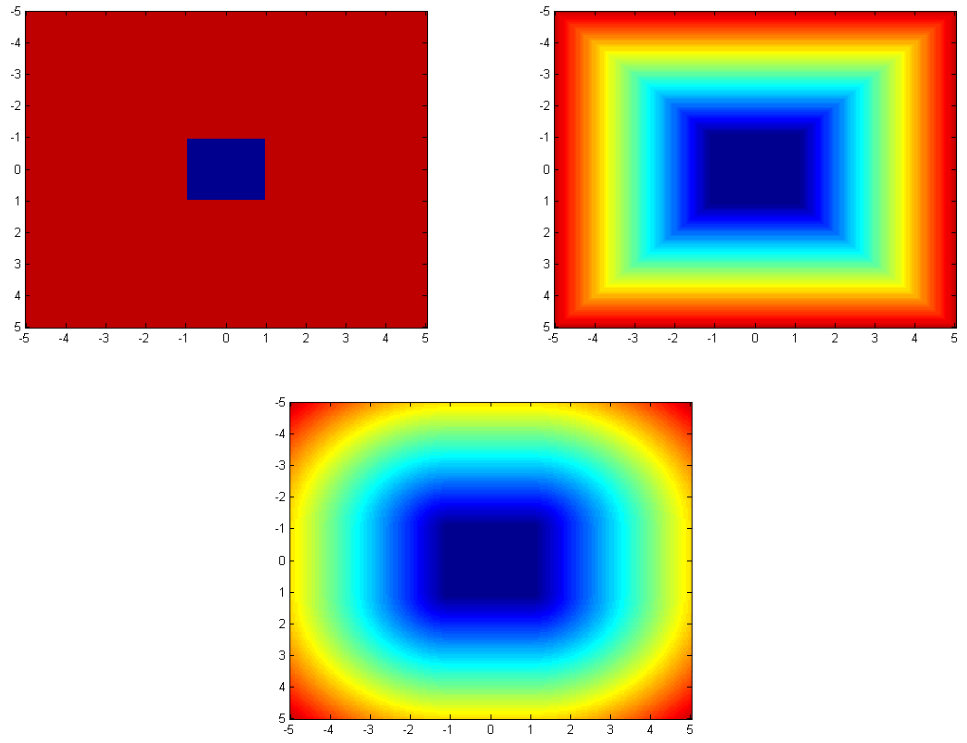


Figure 3.3: *Left to right, top to bottom: colormap plots (scales are different, i.e. the same color on different plots may correspond to different function values) of* $P^0(h(\mathbf{x}))$, $P^1(h(\mathbf{x}))$, *and the actual Euclidian distance from* $\mathbf{x}$ *to* F *where* h *is given by the formula* (3.2.11) *and* F *is a set of all* $\mathbf{x} \in S = [-5,5] \times [-5,5]$ *such that the constraint* $h(\mathbf{x}) \leq 0$ *is satisfied*

from $\mathbf{x}$ to F

$$d(\mathbf{x}, F) \overset{\text{def}}{=} \min_{\mathbf{y} \in F} d(\mathbf{x}, \mathbf{y}),$$

for $v = 2$ are demonstrated on the Figure 3.3. It can be seen that that for $a \neq 0$ a power penalty function provides an approximation of the actual distance to the feasible set, although there can be cases where such approximation can be too crude. In this case special penalty functions based on the knowledge about a particular set of constraints are preferred.

Since changing the fitness evaluation procedure by using combined penalty functions (3.2.2) requires relatively small effort and is relatively easy to analyze due to moderate changes to EA, this approach is widely adopted for constrained EA optimization and has demonstrated its practical usefulness [129]. A common weakness of such an approach is that the choice of penalty functions, combining function φ and their parameters are strongly problem-dependent. Most of the time it requires extensive fine-tuning for the problem in order to achieve optimal performance. Since EAs are often applied to the problems with little knowledge available in advance, this turns out to be a non-trivial task. Solving it is a matter of experience, trial and error, and good heuristics. Little to no theory about choosing the right penalty functions for the problem and method have been developed. There is, however, a promising potential solution in the development of penalty function methods that are adapting and self-adapting (see later in this section).

A general observation about penalty functions coming from experience is that penalty functions of just the number of the violated constraints generally perform worse than the penalty functions using the information about the degree of the constraint violation [162]. This observation supports the claim of the "No Free Lunch Theorem for Optimization" (see section 2.1.4) stating the direct dependence of algorithm performance on the amount of information about the problem it utilizes. From the family of power penalty functions (3.2.10),

$$P^0(z) = \begin{cases} 0 & z \leq 0, \\ 1 & \text{otherwise} \end{cases} \tag{3.2.12}$$

is the only one that does not use information about the degree of constraint violation and thus should be avoided.

The penalty function methods described so far, can be used not only with EAs but with most other unconstrained optimization methods (including those described in section 2.1) and, in general, are frequently used for constrained optimization. Later in this section we review several EA-specific penalty function methods. All of these use transformation to a single-objective minimization problem (3.2.2) via some sort of combining function, φ. It is also worth noting that other methods of constrained EA optimization frequently use penalty functions for various auxiliary tasks.

Levels of Violation

This method was proposed by Homaifar, Lai and Qi [87]. For each constraint, a user defines several levels of constraint violation:

$$0 = z_{0j} < z_{1j} < \ldots < z_{lj} < z_{l+1,j} = +\infty, \; j = 1, \ldots, n, \tag{3.2.13}$$

where l is the chosen number of levels of violation and penalty level coefficients are $R_{1j}, R_{2j}, \ldots, R_{l+1,j}$ so that

$$R_j(z) = R_{kj}, \; z_{k-1,j} < z \leq z_{kj}. \tag{3.2.14}$$

Then the combined objective function is built using a linear combining function (3.2.4) and power penalties (3.2.10) with

$$P_j(z) = P^2(z), \tag{3.2.15}$$

$$w_j = w_j(z) = R_j(z), \tag{3.2.16}$$

for $j = 1, \ldots, n$. Hence

$$\varphi(\mathbf{x}) = f(\mathbf{x}) + \sum_{j=1}^{n} R_j\big(h_j(\mathbf{x})\big) P^2(h_j(\mathbf{x})). \tag{3.2.17}$$

The main idea here is to give a user the ability to precisely balance the contribution of each constraint to the combined objective function by making weight coefficients w_j dependent not only on the index of the constraint function but also on the violation level for this constraint. However, this method requires levels of violation and violation coefficients to be defined for each constraint, thus the total number of parameters of this method is $n(2l + 1)$. Therefore their determination for the problem is a non-trivial problem itself and can easily get quite taxing. At the same time, practical studies from [133] indicate that the quality of solutions obtained via this method heavily depends on the choice of these parameters.

Multiplicative

An interesting approach was proposed by Carlson [34]; she suggested constructing the combined objective function for unconstrained optimization by multiplying the original objective function by a penalty:

$$\varphi(\mathbf{x}) = f(\mathbf{x})P(\mathbf{x}), \tag{3.2.18}$$

where P is designed such that $P(\mathbf{x}) \geq 1,\ \forall \mathbf{x} \in S$. Note that this method also requires $f(\mathbf{x}) \geq 0$, $\forall \mathbf{x} \in S$. Studies on various multiplicative penalties demonstrate reasonably good performance [37].

Dynamic

Joines and Houck [98] propose using a dynamic penalty function, i.e. a function with parameters depend on the number of the current EA generation, similar to the parameter r_k in the SUMT method. In their method, the combined objective function is built using the linear combining function (3.2.4) and power penalties (3.2.10) with

$$P_j(z) = P^{\beta}(z), \tag{3.2.19}$$

$$w_j = (Ck)^{\alpha}, \tag{3.2.20}$$

for $j = 1, \ldots, n$, where k is the generation number and constants C, α and β are method's parameters that can be used to tune it to the problem. With this choice of penalties and the penalty combination method, the unconstrained objective function assumes the form

$$\varphi(\mathbf{x}) = f(\mathbf{x}) + (Ck)^{\alpha} \sum_{j=1}^{n} P^{\beta}(h_j(\mathbf{x})). \tag{3.2.21}$$

Parameters must be chosen such that the penalty multiplier $(Ck)^{\alpha}$ of the objective function increases with generations, thus increasing the pressure to produce feasible individuals after a given time to explore the whole search space S. This method, albeit reported to be efficient and having much smaller number of parameters to set, suffers from the problem of most penalty function ,ethods: the sensitivity of the quality of a result to the choice of method parameters. Even though the values $C = 0.5$ and $\beta = 1$ or 2 were found by authors to work reasonably well, there is an evidence that these values lead to premature convergence at either an unfeasible solution or at a feasible solution impractically far from the optimum (see examples in [133]).

Another idea is to change the penalties dynamically in similarly to the Simulated Annealing method described in section 2.1. The method proposed by Michalewicz and Attia [130] divides all constraints into four types: linear equalities, linear inequalities, nonlinear equalities and nonlinear inequalities. th the uses a random single point that satisfies all linear constraints as a seed and maintains linear constraints satisfied via a set of specially designed genetic operators, finally, it applies a linear combining function (3.2.4) and power penalties (3.2.10) with

$$P_j(z) = P^2(z), \tag{3.2.22}$$

$$w_j = \frac{1}{2\tau_k}, \tag{3.2.23}$$

for $j = 1, \ldots, n$ and $\{\tau_k\}$, a cooling schedule defined by the user. The summation of penalties in the resulting objective function is done over all nonlinear equality and inequality constraints. Linear constraints are considered taken care of, hence

$$\varphi(\mathbf{x}) = f(\mathbf{x}) + \frac{1}{2\tau_k} \sum_{j \in A} P^2(h_j(\mathbf{x})), \tag{3.2.24}$$

where A is a set of indices of the violated nonlinear constraints and τ_k is decreasing as the evolution progresses. The process is stopped upon reaching a final, predefined "freezing temperature", τ_f. This method provides not only a good performance on many test functions with $\tau_0 = 1$, $\tau_f = 0.000001$ and $\tau_{k+1} = 0.1\tau_k$ [130] but also a significant sensitivity to the choice of cooling schedule.

Another annealing objective function is proposed by Joines and Houck [98] and is based on their penalty combination function (3.2.21):

$$\varphi(\mathbf{x}) = f(\mathbf{x}) + e^{(Ck)^{\alpha} \sum_{j=1}^{n} P^{\beta}(h_j(\mathbf{x}))}. \tag{3.2.25}$$

However, this function requires very precise normalization of penalties in order to avoid floating point overflows during computations.

It is worth noting the results published by Hilton and Culver [37]. They tested the multiplicative penalty function method (3.2.18) and additive penalty method (3.2.4). They observed that the linear change of weights with generations resulted in faster convergence when they were kept constant and the multiplicative penalty method (MPM) was more robust than the additive penalty method (APM) for their problem.

Adaptive

The main idea behind adaptive penalty function methods is to introduce a feedback about the performance of the search process into the penalized objective function. An intention behind this idea is to increase the penalty if the algorithm is experiencing insufficient pressure to generate feasible members or decrease it in the other order to redirect effort towards the search for the minimum of the objective function. Weights of the individual penalties could be redistributed based on the statistics about the number of members of the population satisfying particular constraints over generations (see, for example [57]). However, since the adaptive penalty approach itself requires some parameters that are set by the user, this method still suffers from the same parameter-dependence performance problem as all penalty function methods. Research in this direction looks very promising and rewarding, but it is out of the scope of this work. For a detailed description of the family of the adaptive and self-adaptive penalties we refer to [42] and for examples of the adaptive penalty functions and their applications we refer to [45, 58].

3.2.3 Special Genetic Operators

Special genetic operators could be applied to preserve the feasibility of the population. They are particularly important for problems where finding even one feasible solution is problematic. These operators are heavily utilized in the GENOCOP framework (GEnetic algorithm for Numerical Optimization for COnstrained Problems) invented and developed by Michalewicz. They are used to handle constraints in cases where the feasible space F is convex [131]. The first versions were capable of processing linear constraints only; later versions used co-evolutionary techniques and a repairing method to extend the method to process non-convex spaces as well [132,134]. We note that the task of designing such operators is very problem dependent and might not be solvable for the general feasible set F defined by generally nonlinear inequalities (1.3.14).

3.2.4 Selection

Since changes in an objective function produced by penalty functions methods directly affect the fitness of the individuals, they also implicitly affect the selection process. For the penalty function methods, we refer to section 3.2.2 and here we describe only methods that modify the selection process explicitly. Methods of selection modification avoid the weakness of penalty function methods, which is a sensitivity of their performance on the choice of the penalty functions and various parameters of the method. An example of the most straightforward approach of defining a selection for constrained problems can be found in the method of Powell and Skolnick [157]. They suggest using the heuristic selection rule: "every feasible solution is better than any unfeasible." However, for problems with small ρ factor, the algorithm is often trapped in an unfeasible solution [133] and its behaviour is similar to the behaviour of the killing method described in section 3.2.1.

Coello [43] suggests splitting the population into a number of sub-populations equal to the number of constraints plus one and then performing the selection as follows: in each of the sub-populations, selection is based on the corresponding constraint violation; in the last one it is based on the objective function value. This approach is reported to produce good results on several test problems albeit with a somewhat slow convergence rate.

An interesting algorithm to compare individuals was proposed by Jiménes and Verdegay [94]. Selection in their method is performed via a series of deterministic binary tournaments with the winner determined based on the following set of rules:

1. If both members are feasible, selection is performed on the objective function value.

2. If one member is feasible and another one is unfeasible, select the feasible member.

3. If both members $\mathbf{x}$ and $\mathbf{y}$ are unfeasible, select the one with the smaller maximum constraint violation:

$$\xi = \arg\min \left\{ \max_{j=1,\dots,n} h_j(\mathbf{x}), \max_{j=1,\dots,n} h_j(\mathbf{y}) \right\}.$$

This method strongly favors feasible points over unfeasible ones, thus might be omitting useful information about the problem provided by unfeasible members. Hence it might get trapped in a wrong part of the feasible space, i.e. far from the feasible minimum. However, the method may be useful if the feasible space is heavily constrained and/or relatively small.

An extension of this technique, inspired by the concept of the non-dominance from Game Theory and multi-objective optimization and based on so called Niched-Pareto Genetic Algorithm [88] for multi-objective unconstrained optimization, was suggested by Coello and Mezura [41]. Suppose we are given a multi-objective optimization problem: to find $\mathbf{x}^* \in S$ such that

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in S} \mathbf{F}(\mathbf{x}), \tag{3.2.26}$$

where $\mathbf{F}(\mathbf{x}) = \big(f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_N(\mathbf{x})\big)^{\mathrm{T}}$.

The definition of the minimum for the unconstrained multi-objective optimization problem is not trivial since, generally, objective functions $f_i(\mathbf{x})$ attain minimal values at different points. Therefore the problem is not finding a vector that minimizes all objective functions simultaneously but rather finding a set of all Pareto optimal points in a search space [151]. To define Pareto optimality for the problem (3.2.26) we first define the concept of Pareto dominance. Vector $\mathbf{x}$ is said to *dominate* vector $\mathbf{y}$ in the *Pareto* sense if it is not worse than $\mathbf{y}$ in all objective functions values

$$F_l(\mathbf{x}) \le F_l(\mathbf{y}),\ l = 1, \ldots, N$$

and is strictly better in at least one objective function value, i.e.:

$$\exists \hat{l} : F_{\hat{l}}(\mathbf{x}) < F_{\hat{l}}(\mathbf{y}).$$

Basically, this definition says that any of the objective function's values at $\mathbf{x}$ could not be improved by any $\mathbf{y}$ from the search space without worsening another objective function value. Then the point $\mathbf{x}^*$ is called *Pareto optimal* (or non-dominated) for the problem (3.2.26) if it is not dominated by any other point $\mathbf{x} \in S$, i.e. if there does not exist a better compromise for the $\mathbf{x}^*$ in the search space.

Considering penalty functions of constraints as additional objectives to minimize and thus transforming a single-objective constrained minimization problem into a multi-objective unconstrained problem (3.2.1), they propose to use this definition to modify the selection rules as follows:

1. If both members are feasible, selection is performed on the objective function value.

2. If one member is feasible and another one is unfeasible, select the feasible member.

3. If both are unfeasible, one is dominated, and another one is non-dominated, select the non-dominated member.

4. If both are unfeasible and dominated or both are unfeasible and non-dominated, select the one with the minimal constraint violation.

As could be seen, the first two rules remained intact and the last rule is changed using the definition of non-dominance. This approach is demonstrated to be useful and computationally efficient on various test problems. Based on this method and the concept of non-dominance, Oyama, Shimoyama and Fujii developed an extension also applicable to multi-objective constrained optimization problems [149].

3.2.5 Repairing

Repair algorithms are based on the idea of "repairing" the unfeasible members of the population to make them feasible and then either using the repaired version to evaluate the fitness of the original member or to replace it altogether. Particularly popular in combinatorial optimization problems, they are not widely used in non-linear numerical optimization problems. The GENOCOP algorithm, mentioned in section 3.2.3, employs repair algorithms [132]. During the search process it maintains two separate populations. The first one is kept feasible against linear constraints via specialized genetic operators and is used for the search of an optimum. The second population is used for constraint satisfaction and consists only of fully feasible members. These fully feasible members are used to repair linearly feasible members of the first population before fitness calculation.

In principle, any optimization algorithm (see section 2.1) can be used for repairing. Since these methods put high pressure on keeping population feasible, they might be particularly useful for heavily constrained problems, problems with small feasible space and problems where constraint satisfaction is critical like engineering problems or verified optimization (see section 2.2). The disadvantages of these methods are that they increase computational cost and might actually harm the search process, if repair algorithm is not tuned for a problem by turning potentially useful unfeasible members into useless feasible members (see examples of useful unfeasible members on Figure 3.1, section 1.3.3).

3.2.6 Other Methods

Handling constraints in some pre-defined order was suggested by Schoenauer and Xanthakis [166]. Their method starts with a randomly generated population and evolves it to minimize the violation of j-th constraint on j-th step, i.e. its objective function on j-th step is

$$\varphi_j(\mathbf{x}) = P_j(h_j(\mathbf{x})). \tag{3.2.27}$$

The population from step $(j-1)$ is used as an initial population for step j. Points that do not satisfy the first $(j-1)$ constraints are eliminated during the j-th step. The search process stops when a certain threshold on a number of members of the population that satisfy the current constraint is reached. On the last step, the method optimizes the objective function itself. This algorithm was primarily designed for engineering problems where the search space is small and sparse and shows reasonable performance in these cases. However, it poses a problem of selecting a particular order of constraints. It is reported that different orders produce different results in terms of the run time and precision [133].

Koziel and Michalewicz [108] proposed and reported successful application of the interesting method based on establishing a homomorphous mapping between a generally nonlinear, non-convex feasible search space and a v-dimensional cube that is convex, linear and is much easier to optimize with EA. Using this method, an original constrained problem is transformed into a topologically equivalent but simpler unconstrained problem. This method, apart from being elegant and efficient, suffers from the complexity of the transformation in general case and the additional computational expense it requires.

The co-evolutionary model [8] constitutes another popular direction of constrained optimization with EA. As was mentioned earlier, some algorithms are maintaining several populations in which evolution is performed based on various criteria. Sometimes different flavours of EA are used. GENOCOP (see section 3.2.3), for example, maintains two populations, one with members satisfying linear constraints and another one with members satisfying all constraints.

Other approaches include using Immune System Emulation [44, 84], Cultural Algorithms [40], Ant Colony Optimization [28], and many others. However, there is no universally superior constrained optimization method found yet; most of the described methods are applicable to different problems with reasonable average performance yet they perform particularly well on some specific problem or a class of the problems. There is a variety of other methods that are unique or combine several approaches and thus fall off our classification. For thees methods we refer to the references in the beginning of the section. Spreading interest in the field and active contributions from many researchers, have increased their number rapidly. We present our own approach to constrained optimization with EA in section 3.3.

3.3 The REPA Constrained Optimization Method

3.3.1 Introduction

As already noted in section 1.3.3, constrained optimization problems (1.3.8) form an important class of all optimization problems (1.3.4), which is the reason for the existence of a large number of constrained optimization methods for Evolutionary Algorithms (see section 3.2). These methods are usually divided into several subclasses by the main approach employed to handle constraints in originally unconstrained EA: killing, penalty functions, special genetic operators, selection modification, repairing unfeasible individuals and others. Each of these approaches has advantages and examples of successful applications as well as disadvantages demonstrated in test cases. It was also noted that existence of the universally best constrained optimization method for EA is not very probable due to the "No Free Lunch Theorem for Search and Optimization" (see section 2.1.4) [135].

However, non-existence of the best method does not eliminate the possibility to design better general methods. Since most of the time optimization methods are applied to a particular problem, there is also a definite need for new methods that perform better on this problem or class of the problems even if they fail on some other problems. Also, since the repair approach to constraints handling was successfully applied to a combinatorial optimization with EA but is not very popular in numeric mathematical programming, it was worth exploring. Our main motivations were to create a method which would be useful for constrained verified global optimization (see section 2.2 for global optimization and section 2.3.6 for the description of the scenario of integration of our EA with *COSY-GO* global optimization package) and to use for the problems of constrained optimization in accelerator physics where constraints are often imposed by physical limitations and thus must not be violated. For these purposes, the main requirement is not to produce the optimal value but rather to produce good value to serve as an excellent cutoff update as fast as possible. For constrained verified global optimization the result must be feasible, otherwise it is useless.

Also note that while the found value is feasible but not optimal, it could still be a good cutoff value. Most of the methods described in section 3.2 are constructing feasible members by generating the initial population and then performing stochastic mutation and crossover. Most of the methods, while shown to be efficient given enough time and good at preserving diversity in the population, might not put enough pressure on constraints satisfaction to work in conjunction with *COSY-GO*. The repair type of the constrained satisfaction methods seems the most promising for our purpose. Finally, this method might also get useful if the evaluation of the objective function is much more expensive than evaluation of constraints, which, nevertheless, have to be satisfied since it does not require objective function evaluations in order to perform repairing. The approach suggested in this work is of this repair type and is called REPair Algorithm (REPA).

3.3.2 REFIND: REpair by Feasible INDividual

The REPair Algorithm (REPA) consists of two repairing techniques working together in order to transform an unfeasible member of the population into a feasible one and then replacing it in the population with the result if repair was successful. These two techniques are called REpair by Feasible INDividual (REFIND) and REpair by PRojecting through OPTimization (REPROPT). The first one was originally suggested by Michalewicz, then implemented in his GENOCOP package [132] and then adapted and extended by us. The idea of the method is to use already feasible members of the population to repair unfeasible ones by searching for a replacement along the line connecting an unfeasible member $\mathbf{x}_u \in S \setminus F$ with the feasible member $\mathbf{x}_f \in F$. In most cases there is at least some neighborhood of $\mathbf{x}_f$ that belongs to F or the line might cross F in several places. Hence, such searches have high chances of producing a replacement feasible member that is not $\mathbf{x}_f$, thus not trading off the diversity of the population. The algorithm for the REFIND is presented on the Figure 3.4. Here P is a penalty function of the type (3.2.9) that also includes penalties for leaving the search space S. The search for a feasible point along the line is performed via one of the *COSY Infinity* built-in optimization algorithms (see section 2.1) starting from the pre-defined initial parameter value λ_0. The method has the following parameters that are to be set by user:

- The feasible member search algorithm: currently the search is sequential and is is performed until the maximum allowed number of feasible members (5 in this work) is found or the last member of the population is checked. At this point the one that is closest in a sense of the Euclidean distance to the repaired member is selected.

- The algorithm to search for the repair candidate: the search is performed along the line (thus only one parameter is fitted) that includes at least one feasible vector $\mathbf{x}_f$. Hence, the probability of finding a repair in this case is high (we are not considering the case of $\lambda = 1$ since it would produce a duplicate of the $\mathbf{x}_f$ and thus reduce diversity). In order to keep some randomness of the result we currently employ the ANNEALING optimization algorithm (see section 2.1). The initial value λ_0 is common for all members in all generations and quite possibly influences the results as well (see notes on the dependence of the iterative, single-point search methods on the initial point in section 2.1). We suggest avoiding $\lambda_0 = 0$ since it corresponds to $\mathbf{x}_u$ and we know that it is unfeasible in advance. On the other hand, values of λ_0 close to 1 make the initial point for the search closer to the $\mathbf{x}_f$, which is known to be feasible, so the search might stop prematurely and the result might be too close to the $\mathbf{x}_f$ which also unnecessarily decrease the diversity of the population. Thus, by choosing $\lambda_0 \in (0, 0.5)$ we increase the probability of finding a feasible member faster. In this work, $\lambda_0 = 0.1$ is used. Tolerance to the final value and maximum number of steps allowed influence the quality of a result and execution time. We used a tolerance of 0, since the expectation of finding a feasible member by this method is high and the maximum number of steps equal to 15 since we are dealing with a 1-dimensional optimization problem so the search should not take a lot of steps. We also do not want to set up the maximum allowed number of steps to a higher value to keep the repair method reasonably computationally expensive.

- The penalty function method: defines penalty function to minimize in order to find a feasible vector. For the description of the penalty function methods see section 3.2.2. In this work, we employed quadratic penalty functions of the power family (3.2.10) and combining function (3.2.4) with $w_j = 1$, $j = 1, \ldots, n$, $w_{n+1} = 0$ since we are minimizing penalty only. It must be noted, however, that the search algorithm and penalty function method should generally be selected together. For example, some algorithms might be able to solve multi-objective penalized problems (3.2.1) better than single-objective combined problems.

```
Find feasible individuals from the current population
    $R = \{\mathbf{x}_{f,1}, \mathbf{x}_{f,2}, \ldots, \mathbf{x}_{f,N}\}$
If at least one feasible individual is found
    Find $\mathbf{x}_f \in R$ such that $d(\mathbf{x}_f, \mathbf{x}_u) = \min_{\mathbf{x} \in R} d(\mathbf{x}, \mathbf{x}_u)$
    Search for a feasible point along the line connecting $\mathbf{x}_u$ and $\mathbf{x}_f$ by
    solving the optimization problem $\lambda^* = \arg\min_\lambda P(\mathbf{x}_u(1-\lambda) + \lambda\mathbf{x}_f)$,
    where $P$ is a penalty function for constraints violation
    If the resulting penalty is within tolerance
        Repair succeeded, return $\mathbf{x} = \mathbf{x}_u(1-\lambda^*) + \lambda^*\mathbf{x}_f$
    Else
        Repair failed
    End if
Else
   Repair failed
End if
```

Figure 3.4: *REFIND: REpair by Feasible INDividual algorithm*

3.3.3 REPROPT: REpair by PRojecting through OPTimization

The second method used to repair individuals is REPROPT. Its main idea is to perform a projection of the unfeasible member to the feasible set by optimizing the penalty function via some single-point iterative method (see section 2.1). The unfeasible point serves as an initial value for the optimizer. Note that the projection here means a possibly existing element in the feasible set F that could be found via the optimization process; hence it depends on the method and method parameters. Moreover, if the method is stochastic (for example, Simulated Annealing), the results of the projection are not unique.

This method follows the same logic as REFIND in Figure 3.4. However, for REPROPT, there is no feasible member in the population thus there is no parametrization of the line between $\mathbf{x}_f$ and $\mathbf{x}_u$ as in REFIND, hence instead of minimizing the penalty function by changing one parameter λ (1-dimensional problem), we perform minimization by changing all coordinates (or a subset) of the repair candidate (multi-dimensional problem). This, of course, increases the complexity of the problem (see also section 2.3.6 for discussion of the increased complexity of the multi-objective optimization), since the direction towards a feasible set is generally not known. Thus for high-dimensional problems, this method might be inefficient and might have to be replaced with some other strategy, for example, killing. Some other strategy to find at least one feasible member for REFIND can also be employed. Another possibility is to use quasi-projection, i.e. to project using a relatively large penalty tolerance. Then the points successfully projected by REFIND, would reside in some neighborhood of F but not necessarily inside F itself.

Parameters of this algorithm are the choice of the penalty functions method, optimizer for projection, penalty satisfaction tolerance, and a maximum number of steps allowed. Parameters for REFIND and REPROPT can be tuned separately. In this work we used the LMDIF optimization algorithm (see section 2.1) and transformation of a single objective constrained problem into a multi-objective problem (3.2.1) via power penalties (3.2.10) (see section 3.3.5 for the justification of the choice) with penalty tolerance 1^{-5} and the maximum number of steps allowed, 50. Note, that since LMDIF is capable of handling multi-objective optimization problems, we transform our single-objective constrained problems into a multi-objective unconstrained problem (3.2.1).

3.3.4 REPA: REPair Algorithm

A simplified demonstration of the possible results of the application of those two algorithms to the example initial population from Figure 3.1 is presented in Figure 3.5. Here, points e and g are repaired by the REFIND algorithm, and a is repaired by the REPROPT. Dotted lines represent optimization paths and primed points represent possible repair candidates.

For the unfeasible point e, the closest feasible point is f so the successful repair candidates e' and e'' are located on the line connecting e and f, e' corresponds to $\lambda^* < 1$ and e'' corresponds to $\lambda^* > 1$. For the unfeasible point g, the closest feasible point is h. Even though h lies on the boundary of F, the repair candidates g', g'', g''', and g'''' lie inside F. Note that the line connecting g and h leaves and enters F. This demonstrates that repair candidates obtained via some feasible point might not necessarily be close to it, thus the diversity is preserved. Here g' corresponds to $\lambda^* < 0$, g'' corresponds to $0 < \lambda < 1$, and g''' and g'''' correspond to $\lambda > 1$.

Even though there are feasible members in the population, point a is shown repaired by the REPROPT method as a demonstration. It can be seen that the repair candidate a' obtained via some iterative optimization method is not the closest feasible point from F that we can obtain knowing the structure of F. However, since the structure of F is complex and it is given as a set of nonlinear equalities and inequal-

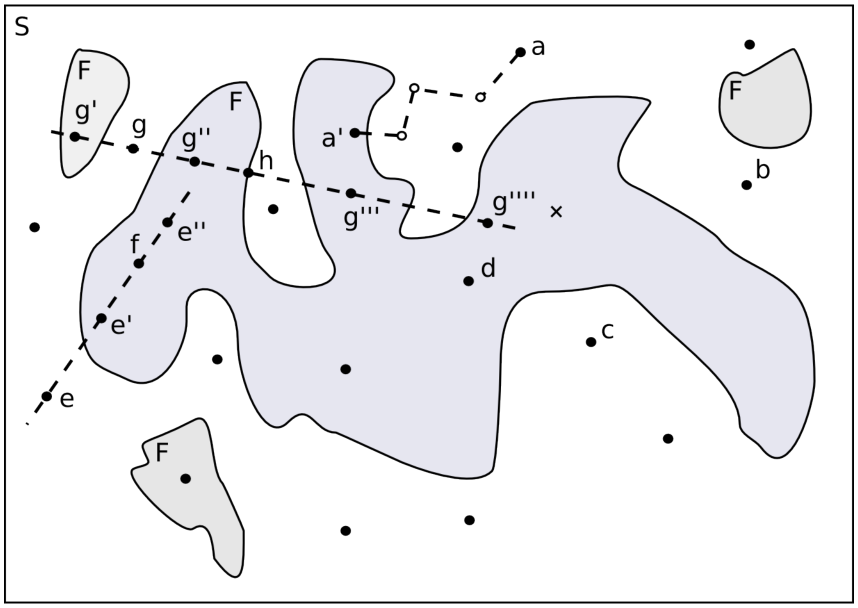


Figure 3.5: *An example of the repairs performed by REFIND and REPROPT repair methods (F is large compared to S)*

ities, such projection might not be feasible to perform. Also note that the feasible repair candidate a' is in some sense "worse" than the unfeasible point a itself since it is farther from the feasible minimum. To avoid this, we can include the original objective to optimization done by REPROPT but generally this should be decided on a per-problem basis, since, as we noted, the objective function itself might be expensive to calculate. Also it is worth noting that here we demonstrate only successful repairs and the volume of the example F compared to the volume of S is relatively large. If we start from the population from Figure 3.5 but reduce the feasible space as shown in Figure 3.6, we notice that there are no feasible members in the population to use for REFIND and that it is much more difficult for REPROPT to produce a feasible repair candidate. Thus, only d, which was already close to F, is repaired to d' while h', a', and c' are considered repair failures.

The REPA algorithm uses both of these methods to perform repairs of an unfeasible member as shown in Figure 3.7 (see (2.3.9) for the definition of rand). The user

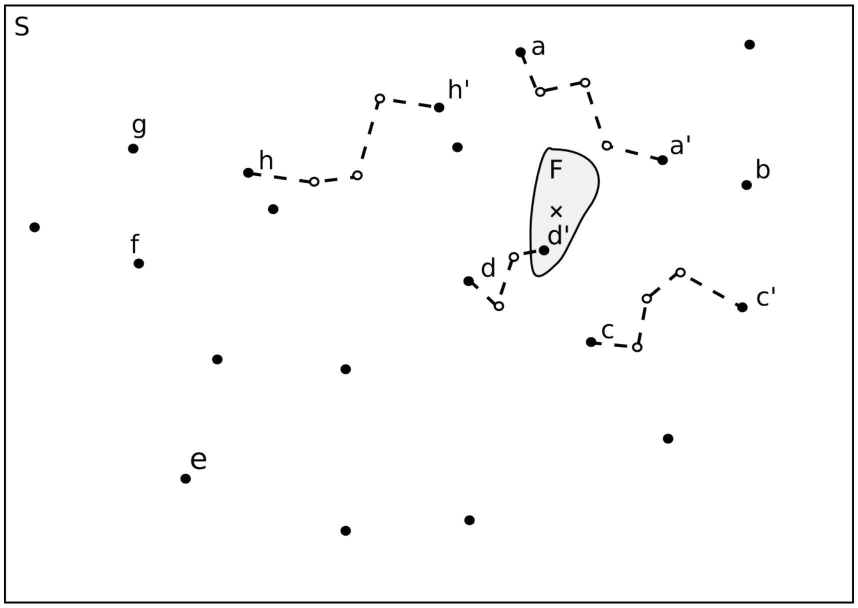


Figure 3.6: *An example of the repairs performed by the REFIND and REPROPT repair methods (F is small compared to S)*

can control the percentage of the population repaired and the required penalty tolerance. Note that this REPA does not change the objective function nor does it change the selection process, thus additional modifications to the EA might be needed. In this work we used the EA from section 2.3 and the penalty functions method (3.2.2) with a linear combining function (3.2.4) ($w_j = 1, \forall j$) and quadratic power penalties (3.2.10).

An Additional benefit of the REPA method is that it is only applied if the members of the population are unfeasible. As long as the population remains feasible, it produces no effect on the optimization.

```
If combined penalty > penalty tolerance
    If rand[0,1] < percent repaired
        If succeeded $\mathbf{x}$ = REFIND($\mathbf{x}_u$)
            Repair succeeded, replace $\mathbf{x}_u$ in population with $\mathbf{x}$
        Else
            If succeeded $\mathbf{x}$ = REPROPT($\mathbf{x}_u$)
                Repair succeeded, replace $\mathbf{x}_u$ in population with $\mathbf{x}$
            Else
                Repair failed
            End if
        End if
    Else
        Repair skipped
    End if
Else
    Repair not needed
End if
```

Figure 3.7: *REPA algorithm*

3.3.5 Studies on Constraint Projection by Standard COSY Infinity Optimizers

Introduction

Both the REFIND and REPROPT methods have certain parameters to be set by the user. Both use one of the built-in *COSY Infinity* optimizers (SIMPLEX, LMDIF, ANNEALING, all described in section 2.1). The choice of ANNEALING as the optimization method for REFIND was justified by two reasons: first, the task is one-dimensional, thus relatively simple, and is solvable with high probability, as explained in section 3.3.2 so all search methods work efficiently, and, second we wanted to add randomness to the process so that there is no pattern in the results of the repair such as a repair candidate being close to the boundary of F or close to the feasible point used for repair. Therefore we selected ANNEALING as the "most random" of all three methods. The REPROPT method, however, deals with much harder v-dimensional optimization problems, hence the question of the best optimization algorithm it is not solvable as easily. Although generally this selection should be made for the problem (see section 2.1.4), we investigated this question on the set of test problems commonly used in the field of constrained EA optimization [135] (see Appendix D).

The parameters of REPROPT include the penalty functions method, the algorithm used for projection, penalty satisfaction tolerance and the maximum number of steps allowed. To determine good default values of these parameters, we studied the performance of this method on a standard set of test problems for constrained optimization with Evolutionary Algorithms [128, 135] (see section 2.1). Built-in *COSY Infinity* unconstrained optimizers (see section 2.1) and their combinations were used for this purpose. The choice of the optimization methods is based on their long-term reputation of being versatile, robust and efficient tools (see section 2.1). They are frequently used by many nonlinear optimization packages and are readily available in the *COSY Infinity* system where GATool (see section 2.3) is implemented. Different methods to construct penalty functions and formulate the unconstrained optimization problem from the constrained problem by their means (see section 3.2.2) were explored. The transformations and conventions applied to all the test problems and the general setup of the tests are described in the next section.

Methodology

All the equality constraints of the type (1.3.2) were converted into the equivalent inequality constraints (1.3.3) using the transformation (1.3.12) so that the feasible set is given by (1.3.14).

All constraints in the test set were known to be satisfiable. Since we were not interested in the global minima of the constraint functions but rather in the simultaneous satisfaction of all constraints, the set of constraint functions was converted to a set of penalties using power penalties (3.2.10) with $a = 0$, 1, and 2. Using the property (3.2.9), which the power penalty functions satisfy, the problem of projecting point $\mathbf{x}_0$ onto F via a chosen optimizer can be formulated as follows: using $\mathbf{x}_0$ as a starting value, find $\mathbf{x}_f$ such that

$$P_i\big(h_i(\mathbf{x}_f)\big) = \min_{\mathbf{x}\in S} P_i\big(h_i(\mathbf{x})\big) = 0,\ i = 1, \ldots, n. \tag{3.3.1}$$

Such $\mathbf{x}_f$ is then automatically feasible. Note that this method is equivalent to the approach (3.2.1) that allows a conversion of a single-objective constrained optimization problem to a multi-objective unconstrained problem via penalty functions. The difference from it is that in our case we do not have an objective function to minimize. Note also, that in (3.3.1) we can be satisfied with non-zero penalty values if they are within the desired tolerance from zero (e.g. because of the practical considerations). This is particularly applicable to the converted equality constraints because they might be non-zero simply due to the limited precision of the floating-point arithmetic.

Three types of the objective functions were tested:

- *all combined:* a multi-objective problem (3.3.1) was converted into a single-objective problem (3.2.4) with all $w_i = 1$.

- *equality combined + inequality combined:* a multi-objective problem (3.3.1) was converted into a two-objective optimization problem with inequality constraints and equality constraints (transformed to inequality constraints using (1.3.12) but still more difficult to satisfy than the true inequalities) converted into two separate objective functions using the method from *all combined* approach.

This distinction was made because the equality constraints are usually harder to satisfy thus they might require more severe penalties in order to be satisfied by optimizer.

- *separate:* a multi-objective optimization problem (3.3.1) was treated as is. It must be noted, however, that for ANNEALING and SIMPLEX methods, the problem was internally converted into a single-objective optimization problem of optimizing the sum of the squares of the objective functions, i.e. this formulation is equivalent to the *all combined* method for $a = 2$. LMDIF has the ability to solve multidimensional problems directly.

The following abbreviations for the search methods are used: S — SIMPLEX, L — LMDIF, A — ANNEALING optimization methods. Combined methods were implemented by making several steps using one method and then making several steps using another method with the idea to combine the strengths of both methods and compensate for their weaknesses. Combinations of methods and their respective abbreviations are: S+A — SIMPLEX + ANNEALING, S+L — SIMPLEX + LMDIF, L+A — LMDIF + ANNEALING.

Each combination of the penalty function ($a = 0, 1, 2$, selected separately for equality and inequality constraints) and the optimization problem formulation (all combined, equality combined + inequality combined, separate) was tested on each of the simple (S, L, A) and combined (S+A, S+L, L+A) methods. For the problems without equality constraints, optimization problem formulations *all combined* and *equality combined + inequality combined* are equivalent, hence only *all combined* was tested. For problems with only one constraint all formulations of optimization problems are equivalent.

Special abbreviation for each variant of the problem formulation and optimization strategy is employed. The description starts with the abbreviation of the optimization method (S, L, A, S+A, S+L, L+A) followed by the type of the penalty function used for the constraints, enclosed by parentheses. For problems with equality and inequality constraints, both types are separated by a comma and the first type corresponds to equality constraints. The following types are used: 1 for power 0, z for power 1, and z^2 for the power 2. For problems with inequality or equality constraints only, one type denotes the type of the penalty used for the corresponding constraints. For optimization problems of the *all combined* type ":c" is added after the method abbreviation before parenthesis. For problems with both equality and inequality constraints type *equality combined + inequality combined* is also marked with ":c", types of the penalties are separated by "+" instead of a comma. Examples: S+L:c(z^2) denotes SIMPLEX+LMDIF combined method, problem with inequality constraints only, *all combined* objective function, and a penalty power is 2. L(z) denotes LMDIF method, *separate* objective functions, power of the penalty equals 1. L+A:c($z + z^2$) denotes combined LMDIF+ANNEALING optimization method, *equality combined + inequality combined* optimization problem with penalty power 1 for equality constraints and 2 for inequality constraints.

Test problems were constructed by taking constraints from the standard constrained optimization test bench for EAs [128,135] (see Appendix D). Since it mostly consists of the inequality-only constrained problems, a simple 2-dimensional problem (3.3.2) with one equality and four inequality constraints was suggested by Dr. Martin

Berz [27].

$$
\begin{aligned}
g_1(\mathbf{x}) &= x_1^2 + x_2^2 - 1.1^2 = 0 \\
h_1(\mathbf{x}) &= x_1 - 1 \leq 0 \\
h_2(\mathbf{x}) &= -x_1 - 1 \leq 0 \\
h_3(\mathbf{x}) &= x_2 - 1 \leq 0 \\
h_4(\mathbf{x}) &= -x_2 - 1 \leq 0
\end{aligned}
\tag{3.3.2}
$$

Initial points were generated randomly from the uniform distribution over $S = [-100, 100]^v$ and $S = [-1000, 1000]^v$. The total number of the different points tested for each combination was 1000.

For all these methods the maximum number of steps was 1000 and the precision was 10^{-5}. For the combined methods, the maximum number of steps with the first and second methods in one step of the combined algorithm was 10, the total number of steps was counted by summing steps made by both methods, and the maximum was set to 1000. The projection was considered successful if all the objective functions were within the tolerance from the global minimum of zero. The projection was considered failed if the desired tolerance was not reached and the method either converged or reached the maximum allowed number of steps.

Results and Conclusions

Using all those conventions a series of tests was performed. Outcome of those tests is summarized in the tables, where for every combination of the method, the penalty functions and the objective function construction method, the percentage of the successful runs and the average number of steps including the failed runs are listed. For each problem a set of the tables similar to the Tables 3.1, 3.2, 3.3, and 3.4 for the problem (3.3.2) is constructed. The best methods in terms of the number of the successful runs are listed in these tables in **boldface**, the number of the steps of these best methods is highlighted using the same style. Headers of the columns show powers of the penalty functions as described in methodology.

For each method three rows contain the results for the *all combined*, *equality combined + inequality combined* and *separate* objective function construction methods. In the cases where there are no equality constraints or no inequality constraints, the *equality combined + inequality combined* method is equivalent to the *all combined* hence it was not tested. Therefore the number of the rows for each method in this case is two. The problems G03 and G11 have one constraint each, hence the number of rows in these cases is one.

For the sake of space we do not list all tables for all the cases. Results of all test can be found in [156]. Here we list the results for the constraints set (3.3.2) since this problem has both an equality constraint and several inequality constraints (see Tables 3.1, 3.2, 3.3, and 3.4). We also list the results for the problem G03 in Figure D.3) because it has only one constraint (see Tables 3.5, 3.6, 3.7, and 3.8) and for the problem G07 in Figure D.7 because it has the largest number of constraints (see Tables 3.9, 3.10, 3.11, and 3.12).

Results of all tests are summarized in two performance tables: Table 3.13 and Table 3.14 [156]. For every problem the three best approaches to constraint satisfaction are listed. Different tables correspond to the different initial point sampling ranges. The comparison is based on the percentage of the successful runs and the average number of the steps made during the search process (including failed ones).

From those tables it could be clearly seen that the optimal approach to constraint satisfaction on the selected set of problems is:

Table 3.1: *Success rate of the constraints projection for the Problem G00* (3.3.2) *on 1000 random points from* $[-100, 100]^v$*. Here* $v = 2$*, problem has one nonlinear equality constraint and four linear inequality constraints. Three rows for each method correspond to the* all combined*,* equality combined + inequality combined *and* separate *optimization problem formulation methods. Best methods in terms of the percentage of the results are listed in boldface*

Method	% success								
	$1,1$	$1,z$	$1,z^2$	$z,1$	z,z	z,z^2	$z^2,1$	z^2,z	z^2,z^2
SIMPLEX	0.00	0.00	0.00	60.20	57.10	31.70	78.50	72.50	74.50
	0.00	0.00	0.00	29.10	29.10	29.70	37.00	37.00	37.70
	0.00	0.00	0.00	29.10	29.10	29.70	37.00	37.00	37.70
LMDIF	0.00	0.00	0.00	50.60	60.90	81.90	64.70	92.70	94.10
	0.00	0.00	0.00	66.80	91.80	**95.50**	80.10	85.90	89.80
	0.00	0.00	0.00	66.80	**98.10**	**99.60**	80.00	**98.10**	94.00
ANNEALING	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.10	0.20
	0.00	0.00	0.00	0.00	0.00	0.00	0.30	0.00	0.20
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.20
SMP+LMD	0.00	0.00	0.00	91.00	64.90	85.50	**96.10**	**100.0**	**99.40**
	0.00	0.00	0.00	84.50	**100.0**	**100.0**	**98.80**	**99.90**	**100.0**
	0.00	0.00	0.00	84.50	**100.0**	**100.0**	**98.80**	**100.0**	**100.0**
SMP+ANN	0.00	0.00	0.00	0.10	0.30	0.100	77.30	76.30	71.60
	0.00	0.00	0.00	0.20	0.20	0.200	76.60	73.30	74.10
	0.00	0.00	0.00	0.20	0.20	0.00	77.70	75.90	75.80
LMD+ANN	0.00	0.00	0.00	**99.60**	91.20	73.10	**98.20**	93.00	**95.50**
	0.00	0.00	0.00	**99.70**	**100.0**	**100.0**	**98.50**	75.00	**97.30**
	0.00	0.00	0.00	**99.70**	**100.0**	**100.0**	**98.50**	**100.0**	**96.60**

Table 3.2: *Average number of steps of the constraints projection for the Problem G00* (3.3.2) *on 1000 random points from* $[-100, 100]^v$

Method	avg # of steps								
	$1,1$	$1,z$	$1,z^2$	$z,1$	z,z	z,z^2	$z^2,1$	z^2,z	z^2,z^2
SIMPLEX	4	37	40	338	409	422	262	338	329
	4	4	4	452	452	452	359	359	359
	4	4	4	452	452	452	359	359	359
LMDIF	6	26	34	113	94	72	134	128	166
	6	23	73	56	79	**91**	112	203	176
	6	10	62	56	**50**	**65**	112	**92**	145
ANNEALING	1000	1000	1000	1000	1000	1000	1000	1000	1000
	1000	1000	1000	1000	1000	1000	1000	1000	1000
	1000	1000	1000	1000	1000	1000	1000	1000	1000
SMP+LMD	1000	1000	1000	236	405	206	**169**	**118**	**125**
	1000	1000	1000	248	**140**	**271**	**129**	**233**	**179**
	1000	1000	1000	248	**70**	**201**	**129**	**113**	**126**
SMP+ANN	1000	1000	1000	1000	1000	1000	428	435	476
	1000	1000	1000	1000	1000	1000	431	458	476
	1000	1000	1000	1000	1000	1000	410	439	468
LMD+ANN	1000	1000	1000	**118**	240	383	**225**	279	**273**
	1000	1000	1000	**104**	**170**	**232**	**217**	741	**382**
	1000	1000	1000	**101**	**82**	**147**	**220**	**156**	**226**

Table 3.3: *Success rate of the constraints projection for the Problem G00* (3.3.2) *on 1000 random points from* $[-1000, 1000]^v$

Method	% success								
	$1,1$	$1,z$	$1,z^2$	$z,1$	z,z	z,z^2	$z^2,1$	z^2,z	z^2,z^2
SIMPLEX	0.00	0.00	0.00	56.40	58.30	31.80	70.40	73.90	74.80
	0.00	0.00	0.00	29.00	29.00	29.30	40.60	40.60	41.10
	0.00	0.00	0.00	29.00	29.00	29.30	40.60	40.60	41.10
LMDIF	0.00	0.00	0.00	45.80	56.10	81.30	64.70	86.90	92.00
	0.00	0.00	0.00	64.90	92.80	**97.70**	81.20	80.60	88.80
	0.00	0.00	0.00	65.10	**99.60**	**99.10**	81.10	**99.60**	92.80
ANNEALING	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SMP+LMD	0.00	0.00	0.00	90.80	63.20	88.50	**98.20**	**100.0**	**99.70**
	0.00	0.00	0.00	84.50	**100.0**	**100.0**	**99.60**	**100.0**	**100.0**
	0.00	0.00	0.00	84.50	**100.0**	**100.0**	**99.60**	**100.0**	**100.0**
SMP+ANN	0.00	0.00	0.00	0.20	0.20	0.10	76.60	73.90	70.20
	0.00	0.00	0.00	0.30	0.20	0.40	76.20	71.80	69.10
	0.00	0.00	0.00	0.20	0.10	0.10	76.20	74.10	71.60
LMD+ANN	0.00	0.00	0.00	**99.40**	93.60	70.60	**98.30**	92.70	91.50
	0.00	0.00	0.00	**99.50**	**100.0**	**100.0**	**97.70**	0.700	51.00
	0.00	0.00	0.00	**99.60**	**100.0**	**100.0**	**97.70**	**100.0**	87.40

Table 3.4: *Average number of steps of the constraints projection for the Problem G00* (3.3.2) *on 1000 random points from* $[-1000, 1000]^v$

Method	avg # of steps								
	$1,1$	$1,z$	$1,z^2$	$z,1$	z,z	z,z^2	$z^2,1$	z^2,z	z^2,z^2
SIMPLEX	4	51	53	386	395	455	351	343	344
	4	4	4	447	447	447	366	366	366
	4	4	4	447	447	447	366	366	366
LMDIF	6	33	48	187	168	121	210	207	209
	6	30	87	131	92	**114**	191	271	227
	6	11	72	131	**45**	**79**	192	**101**	167
ANNEALING	1000	1000	1000	1000	1000	1000	1000	1000	1000
	1000	1000	1000	1000	1000	1000	1000	1000	1000
	1000	1000	1000	1000	1000	1000	1000	1000	1000
SMP+LMD	1000	1000	1000	261	442	201	189	161	163
	1000	1000	1000	274	192	332	164	342	281
	1000	1000	1000	274	90	233	164	154	171
SMP+ANN	1000	1000	1000	1000	1000	1000	512	531	573
	1000	1000	1000	1000	1000	1000	508	541	599
	1000	1000	1000	1000	1000	1000	513	537	573
LMD+ANN	1000	1000	1000	**151**	223	437	**297**	341	366
	1000	1000	1000	**135**	**271**	**332**	**299**	1000	788
	1000	1000	1000	**133**	**106**	**171**	**292**	**218**	318

Table 3.5: *Success rate of the constraints projection for the Problem G03 (Figure D.3) on 1000 random points from* $[-100, 100]^v$*. Here* $v = 10$*, problem has one nonlinear equality constraint. One row for each method correspond to the* separate *optimization problem formulation methods. Best methods in terms of the percentage of the results are listed in boldface*

Method	% success		
	1	z	z^2
SIMPLEX	0.00	**100.0**	**99.90**
LMDIF	0.00	83.30	77.70
ANNEALING	0.00	0.00	0.00
SIMPLEX+LMDIF	0.00	**100.0**	**100.0**
SIMPLEX+ANNEALING	0.00	46.80	55.20
LMDIF+ANNEALING	0.00	**100.0**	**100.0**

Table 3.6: *Average number of steps of the constraints projection for the Problem G03 (Figure D.3) on 1000 random points from* $[-100, 100]^v$

Method	avg # of steps		
	1	z	z^2
SIMPLEX	7	**258**	**257**
LMDIF	9	338	430
ANNEALING	1000	1000	1000
SIMPLEX+LMDIF	1000	**333**	**488**
SIMPLEX+ANNEALING	1000	924	901
LMDIF+ANNEALING	1000	**270**	**361**

Table 3.7: *Success rate of the constraints projection for the Problem G03 (Figure D.3) on 1000 random points from* $[-1000, 1000]^v$

Method	% success		
	1	z	z^2
SIMPLEX	0.00	**99.50**	**99.50**
LMDIF	0.00	69.40	68.80
ANNEALING	0.00	0.00	0.00
SIMPLEX+LMDIF	0.00	**99.90**	**100.0**
SIMPLEX+ANNEALING	0.00	0.00	0.00
LMDIF+ANNEALING	0.00	**100.0**	**100.0**

Table 3.8: *Average number of steps of the constraints projection for the Problem G03 (Figure D.3) on 1000 random points from* $[-1000, 1000]^v$

Method	avg # of steps		
	1	z	z^2
SIMPLEX	7	**343**	**343**
LMDIF	9	475	526
ANNEALING	1000	1000	1000
SIMPLEX+LMDIF	1000	**466**	**691**
SIMPLEX+ANNEALING	1000	1000	1000
LMDIF+ANNEALING	1000	**419**	**614**

Table 3.9: *Success rate of the constraints projection for the Problem G07 (Figure D.7) on 1000 random points from* $[-100, 100]^v$*. Here* $v = 10$*, problem has 8 inequality constraints (3 linear, 5 nonlinear). Two rows for each method correspond to the* All combined *and* separate *optimization problem formulation methods. Best methods in terms of the percentage of the results are listed in boldface*

Method	% success		
	1	z	z^2
SIMPLEX	0.00	48.20	64.70
	0.00	0.00	0.00
LMDIF	0.00	**100.0**	**90.10**
	0.00	**100.0**	**99.30**
ANNEALING	0.00	0.00	0.00
	0.00	0.00	0.00
SIMPLEX+LMDIF	0.00	0.00	0.00
	0.00	0.00	0.00
SIMPLEX+ANNEALING	0.00	0.10	0.00
	0.00	0.00	0.00
LMDIF+ANNEALING	0.00	0.00	0.00
	0.00	0.00	0.00

Table 3.10: *Average number of steps of the constraints projection for the Problem G07 (Figure D.7) on 1000 random points from* $[-100, 100]^v$

Method	avg # of steps		
	1	z	z^2
SIMPLEX	12	782	741
	12	44	44
LMDIF	14	**130**	**441**
	14	**122**	**342**
ANNEALING	1000	1000	1000
	1000	1000	1000
SIMPLEX+LMDIF	1000	1000	1000
	1000	1000	1000
SIMPLEX+ANNEALING	1000	1000	1000
	1000	1000	1000
LMDIF+ANNEALING	1000	1000	1000
	1000	1000	1000

Table 3.11: *Success rate of the constraints projection for the Problem G07 (Figure D.7) on 1000 random points from* $[-1000, 1000]^v$

Method	% success		
	1	z	z^2
SIMPLEX	0.00 0.00	25.20 0.00	40.40 0.00
LMDIF	0.00 0.00	**99.80** **100.0**	**92.90** **97.20**
ANNEALING	0.00 0.00	0.00 0.00	0.00 0.00
SIMPLEX+LMDIF	0.00 0.00	0.00 0.00	0.00 0.00
SIMPLEX+ANNEALING	0.00 0.00	0.00 0.00	0.00 0.00
LMDIF+ANNEALING	0.00 0.00	0.00 0.00	0.00 0.00

Table 3.12: *Average number of steps of the constraints projection for the Problem G07 (Figure D.7) on 1000 random points from* $[-1000, 1000]^v$

Method	avg # of steps		
	1	z	z^2
SIMPLEX	12 12	908 50	869 50
LMDIF	14 14	**139** **129**	**499** **514**
ANNEALING	1000 1000	1000 1000	1000 1000
SIMPLEX+LMDIF	1000 1000	1000 1000	1000 1000
SIMPLEX+ANNEALING	1000 1000	1000 1000	1000 1000
LMDIF+ANNEALING	1000 1000	1000 1000	1000 1000

Table 3.13: *Best constraint satisfaction approaches according to the tests performed at 1000 random points from $[-100, 100]^v$. Here v is defined by a problem (see Appendix D). Success rates for the problem G05 are too low, hence II and III best methods for it are not listed*

problem	I			II			III		
	method	% succ	steps	method	% succ	steps	method	% succ	steps
G00	L+A(z, z)	100.0	82	L(z, z)	98.1	50	S+L(z, z)	100.0	70
G01	L(z)	100.0	45	L:c(z)	98.67	94	L(z^2)	100.0	234
G02	L(z)	97.8	65	S+A:c(z^2)	97.6	93	S+A:c(z)	97.0	94
G03	S(z)	100.0	258	L+A(z)	100.0	270	S+L(z)	100.0	333
G04	L(z)	100.0	19	L+A(z)	100.0	53	L(z^2)	100.0	80
G05	L$(z^2 + z)$	10.1	938	-	-	-	-	-	-
G06	L(z)	99.9	83	L(z^2)	99.6	121	S+L(z)	100.0	191
G07	L(z)	100.0	122	L(z^2)	99.3	342	-	-	-
G08	L+A(z)	100.0	66	S+L(z)	100.0	67	L(z)	99.5	56
G09	S:c(z^2)	96.1	327	L+A(z^2)	97.5	513	L+A(z)	89.6	373
G10	L(z^2)	81.9	386	S+L(z^2)	76.0	501	L+A(z)	74.1	379
G11	L(z)	100.0	20	S+L(z)	100.0	50	L+A(z)	100.0	56
G12	S+L(z)	100.0	125	L+A(z)	100.0	132	S+L(z^2)	100.0	210
G13	L+A(z)	99.9	361	S+L(z)	98.3	327	L(z)	75.6	342
vess	S+L:c(z^2)	98.3	242	L+A(z)	91.6	141	L(z)	89.4	90
tens	L(z^2)	22.8	202	L(z)	20.7	329	S+A:c(z^2)	25.1	902

Table 3.14: *Best constraint satisfaction approaches according to the tests performed at 1000 random points from $[-1000, 1000]^v$. Here v is defined by a problem (see Appendix D). Success rates for the problem G05 are too low, hence I, II and III best methods for it are not listed*

problem	I			II			III		
	method	% succ	steps	method	% succ	steps	method	% succ	steps
G00	L+A(z, z)	100.0	106	L(z, z)	99.6	45	S+L(z, z)	100.0	90
G01	L(z)	100.0	45	L:c(z)	98.5	97	L(z^2)	100.0	278
G02	L(z)	94.0	103	S+A:c(z)	78.8	302	S+A:c(z^2)	78.6	301
G03	S(z)	99.5	343	S+L(z)	99.9	466	L+A(z)	100.0	419
G04	L(z)	99.9	41	L+A(z)	100.0	130	L(z^2)	100.0	125
G05	-	-	-	-	-	-	-	-	-
G06	L(z)	99.5	116	L(z^2)	98.4	183	S+L(z)	100.0	209
G07	L(z)	100.0	129	L(z^2)	97.2	514	-	-	-
G08	L+A(z)	100.0	92	S+L(z)	100.0	91	L(z)	98.1	80
G09	S:c(z^2)	59.3	715	L+A(z^2)	47.4	572	L+A(z)	25.4	913
G10	L(z^2)	77.8	445	S+L(z^2)	74.3	540	L+A(z)	66.6	453
G11	L(z)	99.9	25	S+L(z)	99.9	69	L+A(z)	100.0	79
G12	S+L(z)	100.0	171	L+A(z)	100.0	187	S+L(z^2)	100.0	295
G13	L+A(z)	98.3	472	S+L(z)	98.1	502	L(z)	66.6	542
vess	S+L:c(z^2)	93.5	268	S:c(z^2)	93.3	123	S:c(z)	92.1	121
tens	L(z^2)	4.6	196	L(z)	2.5	168	S+A:c(z^2)	15.3	984

- optimizer: LMDIF

- objective function type: separate, i.e. penalties for individual constraints are treated as separate objectives in a multi-objective optimization problem (3.2.1)

- power for the penalty function: $a = 1$ for both equality and inequality constraints

This approach is the first best for problems G01, G02, G04, G06, G07 and G11, second best for G00 and "tens" and the third best for G08, G13 and "vess" (see Appendix D). The combined LMDIF+ANNEALING search method used with the same penalty function and the objective function type is the second best approach with a slightly larger number of steps. However, for some problems (G03, G13), it demonstrated significantly better performance; and, for most of them it does not perform significantly worse than the leader. We believe that this is caused by the fact that the random and very heuristic ANNEALING method helps the deterministic and analytic LMDIF method to avoid getting stuck on difficult landscapes in the search space of the complicated problems. We also believe that a good performance of the next best SIMPLEX+LMDIF combined method is also due to the LMDIF while the heuristic SIMPLEX method helps LMDIF to not get stuck. Therefore we consider LMDIF (possibly paired with heuristic "helper method") as the best selection for the constraint satisfaction on the presented set of problems. ANNEALING method alone demonstrated the worse results and SIMPLEX showed generally average performance.

Considering "No Free Lunch Theorems for Search and Optimization" (see section 2.1.4) such a superior performance of one optimization method over others can be explained since it uses the largest amount of the information about the problem to guide the search process. While SIMPLEX and ANNEALING are purely heuristic methods and do not use any information about a problem apart from the function values, LMDIF uses both the first derivative and approximation of the second derivative [77] to determine the direction to the minimum. As one can see in Appendix D, most of the constraints in the presented set of the problems are given in a form of nice, twice continuously differentiable functions. Hence it is possible to use this available information in order to run the specialized method. We speculate that for general constraint functions LMDIF would not work so well do not, in terms of the best constraint satisfaction method might be quite different.

The data in the summary tables 3.13, 3.14 can also be used to help in selecting an optimal number of steps for guaranteed constraint satisfaction. However, we are interested in the computational cost as well as the performance. Hence, a different set of the tests might be needed in order to determine the minimum maximum number of steps allowed in order to reach a desired rate of successful runs to all runs. Here we can only conclude that this level would depend on the maximum allowed number of steps. Setting it to the values less than the average from the tables would most likely lead to a degradation in performance.

We also note that problems with equality constraints (G03, G05, G13) and high-dimensional problems (G03, G07, G09, G10) have indeed demonstrated themselves as harder to solve. However, high-dimensional problem G02 and problem G11 with equality constraint only, did not obey this empirical rule. Hence we suggest the estimation of the difficulty of the problem based on this rule to be taken with care and always verified by simulations.

We see that for these problems the power penalty functions (3.2.10) with $a = 1$ are the best choice, while $a = 2$ is a significantly inferior variant. However, this result is not only a problem-dependent but also an optimizer-dependent hence we can not conclude that these functions are the best choice for any combination of the problem and the optimizer. We believe that since the step penalty functions ($a = 0$) demonstrated near zero successful runs in our test (see tables in section "Results" in [156]), they should generally be avoided as they do not provide any information about the direction in which the penalty is changing. Since they only indicate if the point is feasible or not, the search landscape for such penalties is flat, which is causing most optimization methods to fail because of their inability to make a move to a point that is better than the initial. This conclusion is in accordance with the study on penalty functions in [162].

Wherever it applies (problems G00, G05), our studies do not demonstrate a significant difference in performance between the *all combined* and the *equality combined + inequality combined* optimization problem formulation methods except for the G05 tested on 1000 random points from $[-100, 100]^4$ where it demonstrated 2.5 times better performance than any other method. However, these results were not verified by the test performed on the search domain from the problem formulation (see Figure D.5) [156]. Both these objective function types were outperformed by the *separate* method and thus are not recommended.

Poor results for the problem G05 in both test ranges are observed to be obtained due to a difference in 3 orders of magnitude between search domains for $x_1, x_2 \in [0, 1200]$ and $x_3, x_4 \in [-0.55, 0.55]$ from the problem formulation (see Figure D.5) that are inconsistent with the common search domains of $[-100, 100]^4$ and $[-1000, 1000]^4$ used in testing. Additional testing on the suggested search domain supported all the observations about the best optimization and the objective function construction methods presented earlier. It must be noted, however, that the best results were obtained when the quadratic power penalties were used for the equality constraints, i.e. when penalties for violating the equality constraints were steeper than the ones for violating the inequality constraints.

Based on our tests we conclude that the transformation of the constraint satisfaction problem into the multi-objective unconstrained optimization problem (3.2.1) via the power penalty functions (3.2.10) with $a = 1$ and successive treatment of the resulting optimization problem with the LMDIF *COSY Infinity* optimizer is a reasonable choice for the default parameters of REPROPT. However, we should note that the considered problem set is not very large and it does not cover all the possible cases, hence the results might not be general enough and thus might not be universally applicable. In case of the poor performance of the REPROPT method we suggest tuning the parameters based on the information about the problem, possibly after studies similar to the ones performed for this work.

3.3.6 Performance

In this section we assess the performance of the REPA constraint handling method for Evolutionary Algorithm. The REPA repair algorithm does not work as a standalone method, but rather is designed to work with any Evolutionary Algorithm for the continuous optimization on the real domain. Here we study its performance together with GATool (see section 2.3). The population size for GATool was chosen using the dimension*20 formula, to compensate for additional difficulty that constrained problems have compared to unconstrained ones, all parameters were set to their default values (see Figure B.1). As the test problems we used the standard constrained optimization test bench for EAs [128, 135] (see appendix D for problem formulations and properties).

Three methods are chosen for comparison. The first one is the most straightforward and expectedly least efficient, killing method (see section 3.2.1). For each member of the population the sum of the quadratic penalties (3.2.10) is calculated. Members of the population that have this combined penalty higher than the allowed tolerance (10^{-5} in our simulations) are eliminated and then regenerated using the selected method of the new members generation (randomly uniform by default). This process is repeated several times up to the maximum allowed number (50 in our cases) or until all the penalties are zeros, i.e. until the resulting member is feasible. The objective function is not modified and the elimination process is the only connection of the algorithm with the problem's constraints.

Another one is based on the modification of the objective function via the penalty function method (see section 3.2.2). Due to its efficiency despite the simplicity we employed the annealing dynamic penalty (3.2.24) where $\tau_0 = 100$, $\tau_{k+1} = 0.9\tau_k$ and τ_f is not needed since the number of generations we used for testing is relatively small and the termination upon reaching the final temperature is not required. Despite the existence of the more sophisticated techniques, the penalty function methods are still efficient and quite popular.

Another strategy that we tested is a killing method combined with the objective function modification method. This algorithm has only 5 kill/regenerate cycles but the objective function was calculated using the formula (3.2.4) where all $w_j = 1$ and P_j are quadratic penalty functions from the power penalty family (3.2.10).

For the REPair Algorithm (REPA) following values for the parameters were used: the percentage of the repaired unfeasible members was set to 95% tolerance was set to 10^{-5}. The search space in our tests was given in a form of a v-dimensional box. This box was represented as a set of linear inequality constraints and added to the definition of the feasible set (1.3.14). This was done since the repairing process technically can produce repair candidates out of the search space hence we need to penalize the repair algorithm for leaving the search space.

For the repair methods of REPA the following parameters were used:

- REPair by Feasible INDividuals (REFIND): the feasible member to repair with was chosen among 5 feasible members as the closest to the unfeasible one being repaired; the objective function for constraint satisfaction is built by using the multi-objective optimization approach (3.2.1) with the quadratic power penalties (3.2.10); the optimization algorithm was ANNEALING, the maximum steps allowed was set to 15 and the tolerance was set to 0.

- REpair by PRojecting through OPTimization (REPROPT): constraint satisfaction optimization problem formulation and solution methods were selected based on studies presented in section 3.3.5: single-objective constrained problem was converted into a multi-objective optimization problem via linear power penalties (3.2.10), then solved via LMDIF optimization algorithm with the maximum number of steps allowed set to 50 (compromise between speed and efficiency) and the tolerance set to 10^{-5}.

For each combination of the problem and the method 100 runs were performed. Percentage of the successful runs is given in Table 3.15. Results on the obtained values are summarized in Tables 3.16, 3.17, 3.18, 3.19 for the killing, annealing penalty, killing+penalty and the REPA methods, respectively. In those tables first six columns describe the problem (for full description see Appendix D) with the sixth column, "Optimum", listing the value of the global feasible optimum or the best known value of the optimum. Next four columns describe the results of the 100 runs with the selected optimizer: the best, median, mean and the worst values that were found.

To select the resulting value from the final population produced by the chosen method, the member with the minimum function value among those whose penalties were smaller than the allowed tolerance. If the emphasize is on the constraints satisfaction rather than on the minimum function value (as in the rigorous constrained optimization), the selection of the result can be based on the minimum penalty violation (among those within tolerance, of course) instead. If no population member with its constraint violation penalty within required tolerance was found, the run was considered failed. Mean and median results were calculated over the successful runs only.

It is also worth mentioning that due to the fact that one of the stopping criteria was the maximum number of generations allowed and some of those problems are significantly difficult (particularly high-dimensional), most constrained optimization methods were frequently not convergent. Rather they exhausted the allowed number of generations and stopped, which is particularly true to the least efficient naive killing strategy. Results obtained using this method, even though they can be worse than the one obtained with no such restriction, are particularly important for the problem of cutoff value update generation in rigorous optimization (see section 2.3.6) and allows to keep the execution time under control. In this study we tried to show that at a reasonable computational expense REPA method is capable of generating good solutions. Another study might be required in order to find out if it is able to find optimal solutions to all the problems. Another note that has to be made is that here all the constrained optimization methods were used in conjunction with GATool (see section 2.3) but in principle are not tied to it. For different pairing unconstrained continuous EA-based optimizers results can be different.

By studying the rate of success in Table 3.15 one can conclude that the REPA

Table 3.15: *Percentage of the successful runs for different methods (from* 100 *runs total). HEre* v *is the problem dimension,* n *is the number of constraints, "Diff." column lists difficulties of the problems according to [128]. Here E,A,D,VD mean EASY, AVERAGE, DIFFICULT and VERY DIFFICULT, correspondingly*

Problem	Diff.	v	n	Success Rate (%)			
				Killing	Killing+Penalty	Anneal. Penalty	REPA
G01	D	13	9	2	3	100	9
G02	D	20	2	100	100	100	100
G03	D	10	1	3	2	100	100
G04	A	5	6	100	100	99	100
G05	VD	4	5	0	0	0	100
G06	A	2	2	23	2	54	99
G07	A	10	8	0	0	100	100
G08	E	2	2	100	100	100	100
G09	A	7	4	100	100	100	100
G10	D	8	6	10	0	0	11
G11	E	2	1	12	1	100	99
G12	E	3	1	100	95	100	100
G13	VD	5	3	0	0	76	100
tens	E	3	4	96	44	89	100
vess	A	4	4	100	100	100	100

method is the best one for the considered problem set. Apart from its poor performance on the problems G01 and G10 (see Figure D.10) it was able to produce a feasible result in almost all runs with the default settings. The second best was the annealing penalty method, the third and the fourth places are shared between the killing and the killing+penalty methods.

It is worthwhile to note that for the problem G06, G10, G11 the killing method outperforms the killing+penalty method. If we compare the numerical values of the results produces by those two methods (Tables 3.16, 3.18), we can notice that none of those two methods is superior to another. They both failed on the problems G05, G07, G10 and G13 and demonstrated similar quality of the result on the problems G04, G09 and G12. On the problems G01, G02, G03, G11 the killing+penalty has demonstrated better performance, while on the problems G06, G08, "tens" and "vess" the naive killing performed better. On almost all the problems marked as EASY those

Table 3.16: *Summary of the performance of the killing method, after 150 generations. The first six columns describe the problem (for full description see Appendix D) with the sixth column, "Optimum", listing the value of the global feasible optimum or the best known value of the optimum. The next four columns describe the results of the* 100 *runs with the killing method: the best, median, mean and the worst values that were found. "Failed" is listed if the method failed to produce feasible members in all runs*

Problem	Diff.	v	n	$\approx \rho$	Optimum	Best	Median	Mean	Worst
G01	D	13	9	0.0003	-15	-1.23269847	-0.94973546	-0.94973546	-0.66677245
G02	D	20	2	99.9973	-0.803619	-0.72033161	-0.61513444	-0.61886576	-0.52580262
G03	D	10	1	0.0026	-1	-0.04064644	-0.02961901	-0.02017910	-0.00129928
G04	A	5	6	27.0079	-30665.539	-30335.31418895	-29698.09294104	-29657.03283007	-28900.53863476
G05	VD	4	5	0.000	5126.4981	failed	failed	failed	failed
G06	A	2	2	0.0057	-6961.81388	-6739.778492261	-3987.04501437	-3980.57213715	-1896.96943370
G07	A	10	8	0.0000	24.3062091	failed	failed	failed	failed
G08	E	2	2	0.8581	-0.095825	-0.09581097	-0.06787154	-0.06463032	-0.00760358
G09	A	7	4	0.5199	680.6300573	740.99585723	969.30248340	996.71183836	1470.61981509
G10	D	8	6	0.0020	7049.3307	failed	failed	failed	failed
G11	E	2	1	0.0973	0.75	0.75136398	0.78380867	0.81664430	0.97531872
G12	E	3	1	4.7697	-1	**-0.99998686**	-0.99037649	-0.98523602	-0.89373689
G13	VD	5	3	0.0000	0.0539498	failed	failed	failed	failed
tens	E	3	4	0.7537	0.012681	0.01482986	0.03940990	0.04944165	0.25312028
vess	A	4	4	39.6762	6059.946341	26097.98546078	145573.26246941	155590.61692148	468631.52633727

Table 3.17: *Summary of the performance of the annealing penalty method, after 150 generations. The first six columns describe the problem (for full description see Appendix D) with the sixth column, "Optimum", listing the value of the global feasible optimum or the best known value of the optimum. The next four columns describe the results of the* 100 *runs with the annealing penalty method: the best, median, mean and the worst values that were found. "Failed" is listed if the method failed to produce feasible members in all runs*

Problem	Diff.	v	n	$\approx \rho$	Optimum	Best	Median	Mean	Worst
G01	D	13	9	0.0003	-15	-13.97081785	-13.35167806	-13.34304247	-11.96858017
G02	D	20	2	99.9973	-0.803619	-0.76499256	-0.68851895	-0.68379083	-0.59756816
G03	D	10	1	0.0026	-1	-0.93203377	-0.61954710	-0.59692390	-0.078735953
G04	A	5	6	27.0079	-30665.539	-30606.91795041	-30495.02578847	-30469.84288151	-29731.00533651
G05	VD	4	5	0.000	5126.4981	failed	failed	failed	failed
G06	A	2	2	0.0057	-6961.81388	-6748.31770679	-4679.18773358	-4730.95464291	-2571.65801945
G07	A	10	8	0.0000	24.3062091	26.16089488	36.88350082	36.83329579	56.30857947
G08	E	2	2	0.8581	-0.095825	**-0.09582504**	-0.09570611	-0.07797430	-0.00077549
G09	A	7	4	0.5199	680.6300573	680.70355471	681.42228913	681.75869259	684.73570304
G10	D	8	6	0.0020	7049.3307	failed	failed	failed	failed
G11	E	2	1	0.0973	0.75	**0.75002111**	0.75954186	0.76373447	0.84479402
G12	E	3	1	4.7697	-1	**-0.99999999**	-0.99999995	-0.99999960	-0.99999314
G13	VD	5	3	0.0000	0.0539498	0.05419238	0.10865341	0.16825105	2.15765954
tens	E	3	4	0.7537	0.012681	0.01278792	0.02404540	0.03169533	0.14433929
vess	A	4	4	39.6762	6059.946341	9118.37509202	15403.52108843	17793.18153820	40492.24882717

Table 3.18: *Summary of the performance of the killing+penalty method, after 150 generations. The first six columns describe the problem (for full description see Appendix D) with the sixth column, "Optimum", listing the value of the global feasible optimum or the best known value of the optimum. The next four columns describe the results of the* 100 *runs with the killing+penalty method: the best, median, mean and the worst values that were found. "Failed" is listed if the method failed to produce feasible members in all runs*

Problem	Diff.	v	n	$\approx \rho$	Optimum	Best	Median	Mean	Worst
G01	D	13	9	0.0003	-15	-4.11191434	-3.67404969	-3.14838320	-2.09705022
G02	D	20	2	99.9973	-0.803619	-0.74293001	-0.61800972	-0.62074613	-0.53806335
G03	D	10	1	0.0026	-1	-0.10566401	-0.05324017	-0.05324017	-0.00081634
G04	A	5	6	27.0079	-30665.539	-30299.19404787	-29563.89836339	-29591.03015139	-28757.76853452
G05	VD	4	5	0.000	5126.4981	failed	failed	failed	failed
G06	A	2	2	0.0057	-6961.81388	-3178.72262616	-2439.82085398	-2439.82085398	-1700.91908180
G07	A	10	8	0.0000	24.3062091	failed	failed	failed	failed
G08	E	2	2	0.8581	-0.095825	-0.09213784	-0.02160343	-0.02769523	0.09567066
G09	A	7	4	0.5199	680.6300573	756.10799304	980.40810075	1002.93629674	1450.50357277
G10	D	8	6	0.0020	7049.3307	failed	failed	failed	failed
G11	E	2	1	0.0973	0.75	**0.75046025**	0.87479938	0.87301611	0.99627875
G12	E	3	1	4.7697	-1	**-0.99995272**	-0.96122908	-0.92904864	-0.66399677
G13	VD	5	3	0.0000	0.0539498	failed	failed	failed	failed
tens	E	3	4	0.7537	0.012681	0.01801072	0.06771680	0.07952207	0.26079158
vess	A	4	4	39.6762	6059.946341	31980.67920455	118496.38432568	129879.85346377	415508.99141038

Table 3.19: *Summary of the performance of REPA method, after 150 generations. The first six columns describe the problem (for full description see Appendix D) with the sixth column, "Optimum", listing the value of the global feasible optimum or the best known value of the optimum. The next four columns describe the results of the* 100 *runs with the REPA method: the best, median, mean and the worst values that were found. "Failed" is listed if the method failed to produce feasible members in all runs*

Problem	Diff.	v	n	$\approx \rho$	Optimum	Best	Median	Mean	Worst
G01	D	13	9	0.0003	-15	-14.40789064	-14.12021639	-13.27759026	-6.67395203
G02	D	20	2	99.9973	-0.803619	-0.78062233	-0.69885282	-0.69465392	-0.58371925
G03	D	10	1	0.0026	-1	-0.98759123	-0.95595592	-0.96619962	-0.37845116
G04	A	5	6	27.0079	-30665.539	-30663.67783472	-30625.17570146	-30619.88321270	-30511.31893171
G05	VD	4	5	0.000	5126.4981	**5126.49810959**	5126.51773089	5126.67221275	5130.97866237
G06	A	2	2	0.0057	-6961.81388	-6961.83025982	-6601.42894942	-6111.78553514	-3531.26283679
G07	A	10	8	0.0000	24.3062091	25.66434898	28.51201463	28.80447050	35.31442295
G08	E	2	2	0.8581	-0.095825	**-0.09582504**	-0.09582496	-0.09311891	-0.02914349
G09	A	7	4	0.5199	680.6300573	680.81263235	681.54728706	681.76838004	685.17250653
G10	D	8	6	0.0020	7049.3307	7097.35655952	8713.69524553	9080.98370862	11245.06195825
G11	E	2	1	0.0973	0.75	**0.75000037**	0.75078883	0.75515776	0.82928494
G12	E	3	1	4.7697	-1	**-0.99999999**	-0.99999999	-0.99999988	-0.99999652
G13	VD	5	3	0.0000	0.0539498	**0.05395041**	0.05398875	0.05409692	0.05900387
tens	E	3	4	0.7537	0.012681	**0.01268532**	0.01321193	0.01546248	0.10709291
vess	A	4	4	39.6762	6059.946341	8825.10657358	10004.41585468	11346.49591488	40395.19347539

methods were able to achieve the optimum or a value close to the optimal. Both the killing and the killing+penalty methods were clearly outperformed by the annealing penalty and REPA techniques (see additionaly Tables 3.17, 3.19) both in the success rate and the quality of the results.

It is worth noting that the killing method can be adequate for problems with a relatively large ρ factor. However, for the cases when it is relatively small or very small, random regeneration has a very small chance of generating a feasible member. Here the directional search for the feasible set that is done by REPA demonstrated itself to be more efficient. Specifically this is demonstrated on the problems with equality constraints, i.e. cases with the theoretical ρ equal to zero.

Comparing the annealing penalty and REPA methods (Tables 3.15, 3.17, 3.19) we notice that while the success rate for the problem G01 is higher for the annealing penalty method, the quality of the best and median results was better for REPA method. The annealing penalty method failed on the problems G05 and G10 (VERY DIFFICULT and DIFFICULT), demonstrated a similar performance on the best achieved results for the problems G08, G09, G11 and G12 (EASY and AVERAGE). It was able to find optimal values for the problems G08, G11, G12 (EASY). However the mean and the median results for all the problems except G09 were much better for REPA than for the annealing penalty method (for problem G09 the mean and the average values are of the same quality). Therefore we can conclude that REPA method provides more consistent results over the runs and thus is a more robust algorithm. REPA method found the optimal values for the problems G05, G08, G11, G12, G13 and "tens" and the results close to the optimum for the problems G03, G04, G06, G09. For two problems that were classified by Mezura [128] as VERY DIFFICULT (G05, G13) it was able to find an optimal value consistently (the mean and the median values are nearly optimal). REPA method outperforms the annealing penalty method on all the problems but G09, where the results are of the same quality.

Based on our tests we can conclude that the proposed REPA method was the best from the tested methods for the continuous constrained optimization. It also demonstrates itself as being robust which is important for consistent generation of good cutoff values for rigorous constrained optimization. Due to the repairing process performed by it, the REPA method is relatively computationally expensive but for rigorous optimization it is a valid tradeoff for the quality of the result. Another important case is the optimization of the physical device design, where the objective function can be calculated on the base of the expensive simulations and thus be very taxing to calculate. Constraints in this case are very important to satisfy.

If we compare the results obtained by REPA method with the results obtained with other constrained Evolutionary Algorithms by other authors, we can conclude that the proposed approach is, indeed, efficient and competitive. Those experiments include the studies performed by Michalewicz [133] (problems 1, 2, 3 and 5 in his test set are problems G01, G10, G09 and G07 in our test set, correspondingly); by Coello and Mezura [41] (examples 2, 3, 4 in his test set are the problems "vess", "tens" and G12 in our test set, correspondingly) and by Mezura [128] (see Chapter 5, test functions are the same). We also note that for the problems G05, G13 that are efficiently and robustly solved by our method, most of the other state of the art approaches to constrained optimization with EAs failed even to find a feasible member [128].

Trying to improve the results for the test problems marked DIFFICULT (G01, G02, G10) and those marked AVERAGE (G07, "vess") that were not completely solved because the feasible optimum was not reached with the default settings, we tested if the increase in the maximum allowed number of generations from 150 to 250 increases the quality of the results (this suggestion is more or less generic for GA-hard problems, see for example section 2.3.5). For G01 we did not obtain any statistically significant improvement. For G02 the increased number of generations resulted in the increase in the quality of the results (compare with values from Table 3.19):

```
best   = -0.7896210467822932
median = -0.746413744215766
mean   = -0.740383328183245
worst  = -0.6797819227868159
```

For G10 the increased number of generations produced an increase in the success rate (grows to 19%) with reduction in the quality of the results:

```
best   = 7407.208490599209
median = 9378.23832879927
mean   = 9870.55475611721
worst  = 19827.95976815720
```

For G07 the increased number of generations resulted in the increase in the quality of the results:

```
best   = 25.22498907620453
median = 28.1881923101962
mean   = 28.2941232844445
worst  = 35.71934554079836
```

For "vess" the increased number of generations resulted in the increase in the quality of the results:

```
best   = 8797.914097806060
median = 9724.07325335419
mean   = 10449.0859049756
worst  = 23374.47976868584
```

However, for G02, $\rho \approx 99.9973\%$, hence the main impact on the performance for this problem is an effective optimization inside the feasible region, which REPA method is not interfering with. Hence the performance is mainly affected by the performance of the underlying genetic algorithm and the problem should be tuned for the optimal performance by tuning GATool parameters. We suggest that the better performance of GATool in this case is a main reason for the better results of the GATool + REPA method. For other problems changing REPA settings might be more relevant.

Since the problem G01 is not only high-dimensional (13) but has the largest number of constraint functions (nine) defining the feasible set, we suspected that the increase of performance for this function can be reached by changing the settings for the optimizer used by REPROPT method to project unfeasible members to the feasible set. Experiments demonstrate that by decreasing the projection penalty tolerance to 1 and increasing the maximum allowed number of steps for projection to 70 we can improve the success rate from 9% up to the 100% and increase the quality of results to

```
best   = -14.95789279338429
median = -14.3710717847534
mean   = -14.3276103831189
worst  = -13.12539263469034
```

This case also demonstrates an important technique of repairing. By setting the desired constraint satisfaction penalty to 1 we allow the resulting repaired point to be in the close proximity to the feasible set (provided the constraint functions are continuous, see section 3.2.2), but not necessarily inside it or on the boundary. It turned out that this relaxation of the repairing conditions still can be useful for the optimization process. The determination of the optimal set of parameters for the given constrained optimization problem is not a trivial task and generally done by trial and error. Little to no theory exist on the topic.

If we consider optimizers constructed from GATool and REPA by changing their parameters as different optimizers rather than the same version of the one optimizer (this assumption is valid since most of them can influence the search process to a large extent) from “No Free Lunch Theorems for Search and Optimization” or NFL theorems (see section 2.1.4), we can conclude that there is no universally best set of those parameters, no matter how hard and extensively we try to find them. There will always exist problems for which the default parameters will not work in an optimal way, hence knowing the strategies to tune the method for the problem is also important. For the parameters of GATool and REPA see sections 2.3 and 3.3, correspondingly. Note that the number of parameters is large and some of them are dependent, which on one hand makes it hard to tune them, but on the other hand gives the user a great flexibility. Since we view our optimizer as a general-purpose tool applicable to a wide range of different problems, we consider this mostly as its advantage.

Few words of critique must be said about the standard problem set for EA-based constrained optimization that we used. First, we note that the difficulty of the problem according to NFL theorems is method-dependent, hence the classification of the difficulty made by Mezura [128] and we adopted in this work (see Appendix D) might not be adequate for all methods. Indeed, it turns out, that two of the problems: G05 and G13 marked as VERY DIFFICULT due to a presence of one or more nonlinear equality in the constraint functions set, did not pose a significant problem for our method (see Tables 3.15, 3.19) thus can not be considered VERY DIFFICULT with respect to it. The problems G01, G02, G03, G10 marked as DIFFICULT, due to a high dimensionality and small ρ (except for G02, which has the largest dimensionality) have demonstrated themselves as being moderately difficult since the values close to the optimal were found. However, the problems G07 and vess marked as AVERAGE posed for our method the same or even higher difficulty than the DIFFICULT problems. The problems G08, G11 and G12, marked as EASY, pose no difficulty for our method thus the classification here is adequate.

We claim that the good performance of our approach on the VERY DIFFICULT problems G05 and G13 is due to the fact that their extreme difficulty was suggested based on the small size of the feasible set ($\rho \approx 0$) and the presence of the relatively large number of the nonlinear and linear equalities in the set of constraint functions. However, all those constraint functions are analytic therefore REPROPT repair method having the LMDIF as a default optimizer and thus using the extra information coming from differentiability (see the description of LMDIF in section 2.1), is capable of efficiently repairing unfeasible members (see section 3.3.5). For functions that do not possess those desirable properties, results can be different.

We claim that the difficulty of the DIFFICULT problems is mostly due to their high dimensionality which is known to increase the problem's complexity (see section 2.3.6). Additional studies with the increased maximum number of generations for GATool, increased maximum number of steps for repairing and the decreased constraint projection tolerance for REPA demonstrate an increase in the quality of the result.

We additionally note that the problem G12 originally suggested to be a test for the cases when where a feasible set is disjoint and thus the problem is difficult, demonstrated itself as an easy problem, successfully solved even by the most inefficient methods (see Tables 3.15, 3.17, 3.19, the studies in Chapter 5 of [128]). It seems like the disjoint pieces of the feasible set (3-dimensional balls, see Figure D.12) are large enough, placed close enough to each other and cover the search space dense enough ($\rho \approx 4.7697$) that they do not model the main difficulty they were supposed to model well enough. Moreover, for this problem the unconstrained and the constrained minimumizers coincide (which is easy to verify) hence the results of both unconstrained and constrained minimization should coincide as well. Therefore it does not test the specific difficulty of the constrained minimization. We conclude that other more adequate test problems for disjoint feasible sets are needed.

3.4 Conclusions

In this section we reviewed the challenges of the constrained optimization with the Evolutionary Algorithms. We presented a unified overview of the commonly used constrained optimization methods for EA. We described our proposed constrained optimization by the repairing method called REPA in the exquisite details including its two repairing methods: REFIND and REPROPT. For REPROPT we assessed its performance on the standard constrained optimization test problems for EA with a variety of configurations and we discussed the results and suggested the optimal default configuration. We also studied the performance of the REPA method with the default settings on the same set of test problems and compared its results with several of the commonly used constrained optimization methods for EA described in the review section. We outlined and demonstrated with examples several strategies for REPA performance tuning for the hard problems. We also presented some critique on the de-facto standard test problems set used in this work.

More extensive testing of REPair Algorithm (REPA) on other synthetic problems (see, for example, Chapter 8 in [128]) and real-life problems is needed and therefore is a direction for the future research. The problems that pose difficulties for our method should be identified and their distinctive properties should be studied and described. Here it must be noted that the optimization methods used for the constraint projection by REPA can be changed. Hence the optimization methods that are more appropriate for the considered constrained problem can be used to increase the REPA efficiency.

For the practical purposed REPA repair method should be used together with penalty function methods that augment the objective function with penalties for constraint violation. This is done in order to direct the evolutionary search performed by GATool towards a feasible optimum, not just an optimum of the objective function itself. Otherwise all additional computations required to repair unfeasible members and bring them to the feasible region can become worthless. For example, if the unconstrained optimum is outside of the feasible set and is strongly pronounced, the unconstrained search based on the original unmodified objective function tends to converge to this unconstrained optimum thus driving the constrained optimum search in the wrong direction.

To conclude the section we present some ideas that can be used to further enhance REPA method. The concept of elitism is important for the convergence proofs (see, for example, Chapter 9 in [128])) and in practice it improves convergence. However, in the case of the GATool+REPA optimization the result of the elitism operator application in GATool are frequently destroyed by REPA since every unfeasible member can be repaired and thus has its function value changed. The concept of the feasible elitism might be introduced to perform a similar task, i.e. the best feasible members can be preserved (note, however, that finding even one feasible member is often the main difficulty of the constrained optimization). Preservation of all feasible members found during the optimization might also be a valuable enhancement, especially for the problems with a small ρ, i.e. a feasibility set that is a much smaller than the search space. First, it guarantees the success in terms of finding a feasible result even if it is not optimal. Second, it allows REPA to use the much less expensive and the more efficient REpair by Feasible INDividuals (REFIND) repair method instead of the more general-purpose REpair by PRojecting through OPTimization (REPROPT) (see sections 3.3.2, 3.3.3).

Based on the results described in this Chapter, particularly on the outstanding performance of REPA on the test problems that presented significant difficulty for other constrained optimization EAs, we conclude that the method is useful and competitive. Important consequence of this development is that REPA constrained optimization method can be used to extend the GATool integration with COSY-GO (see sections 2.3.6 and 2.4). Since COSY-GO is capable of the constrained rigorous global optimization as well as of the unconstrained one, GATool, given the ability to handle constrained optimization problems, can be used as is suggested in the scheme of GATool & COSY-GO integration for the constrained rigorous global optimization as well as for the unconstrained one.

CHAPTER 4

Optimization Problems in Accelerator Design

Evolutionary Algorithms (EAs), as was mentioned in section 2.1, have many properties that make them attractive for various applications in Accelerator Physics. They can find globally optimal or nearly optimal solutions even for very high-dimensional functions (when the objective function depends on many control parameters) with moderate computational expenses (see section 2.3.4. They can handle noisy problems which often arise in physical systems, have no requirements on the objective function other than the ability to calculate its value for all points from the search space, and, finally, are relatively easy to implement (see Chapter 2). These properties make EAs a great tool to explore and solve problems that were previously considered unsolvable, or find newer, better, and even unpredicted solutions to well-known problems. EAs are capable of the both unconstrained and constrained optimization (see Chapter 3). This is particularly important for design problems that usually include a large number of constraints imposed by physical limitations, cost considerations, and available technology. EA's ability to generate new solutions mimics the nature's ability to produce new species. It has proved itself useful in the field of design in general and shows great potential for application to accelerator design in particular (see section 4.1 in this chapter). The results presented earlier in this work extend EA's application to the field of rigorous global optimization which is used, for example, for rigorous estimation of the stability of the particles in large accelerators that consist of thousands of elements (see section 4.2).

Despite these attractive features that have created growing interest in Evolutionary Algorithms in science and industry, their usage in the field of Accelerator Physics and Beam Theory is not very common. The number of publications that we were able to find on accelerator design with Evolutionary Algorithms is surprisingly small ([32, 64, 160]) especially in comparison with the number of papers on their application to various other design problems. In this chapter we present several different problems from the various stages of accelerator design that were treated with the GATool Evolutionary Algorithm (see section 2.3). These problems include a simple accelerator design problem, a complex real-life accelerator design problem, and the problem of optimizing the normal form defect function that is connected to the rigorous stability estimation of the particle motion in a circular accelerator (see section 1.1.2). The usefulness of the obtained results encourages us to suggest that the EAs application to these problems is indeed very promising and has great potential for future research.

4.1 Quadrupole Stigmatic Imaging Triplet Design

Different types of electromagnetic elements are used to confine and manipulate charged particle beams in accelerators. The most common are dipoles, which usually generate a constant magnetic field to bend the beam, and quadrupoles which employ a linear field gradient to focus the beam in the transverse dimension. The questions of confinement and focusing are of the utmost importance for circular accelerators and storage rings. Here particles must remain in the central channel for millions of turns in order to be accelerated to the required energy, or focused, as is the case in a collider, to provide continuous collisions of beams circulating in the opposite direction. The three main types of focusing employed in most accelerators are: weak focusing, strong focusing, and solenoidal edge focusing.

Weak focusing was historically the first type of focusing developed for accelerators and is primarily achieved by shaping the poles of the dipole magnets such that the B_y component of the magnetic field decreases with orbit radius r

$$\frac{\partial B_y}{\partial r} < 0,$$

Such shaping of the poles generates a linear restoring force which forces particles to oscillate around a stable reference orbit as they travel through the channel. These oscillations are called betatron oscillations, since the first machines built employing this design principle were betatrons [47,170]. However, the principle of weak focusing has one serious drawback: when the circumference of the accelerator is large, the amplitude of the betatron oscillations is large as well, which leads to either increased particle loss or large magnet apertures to avoid it. Large apertures, in turn, lead to an increase in the magnetic lattice cost and increased complexity in the manufacturing process.

The main principle of strong focusing is in alternating focusing and defocussing magnetic lenses in order to achieve an overall focusing effect. Quadrupole lens are commonly used in accelerators to achieve focusing. However, they focus only in one direction and defocus in the other direction. Hence, in order to to achieve simultaneous focusing in both planes one needs to combine at least two of the quadrupole lenses.

The smallest accelerator element that is focusing in both x and y directions using alternating focusing is called a FODO cell. Here F denotes the lens focusing in one direction (and defocussing in the other direction), D denotes the lens defocussing in the same direction, and O denotes a drift. FODO cells are optically stable if the distance between lenses of equal strength in a sequence is less than their focal length in the focusing plane.

The edge field of the solenoids can be used to focus in both directions simultaneously. However it introduces non-linear components to the particles motion, couples the motions in the (x-a) and (y-b) planes and thus makes the dynamics significantly more complex [119]. Hence this type of focusing is typically limited to the cases where the beam has a large phase space size and a large momentum distribution [144] so the quadrupole focusing cannot be used.

Most modern high-energy accelerators utilize strong focusing by quadrupole lens combinations. It is worth noting that the combination of the magnetic quadrupole lenses with alternating north-south pole orientation (which makes them alternately focusing in (x-a) and (y-b) planes) has a stronger net focusing effect than a series of solenoid lenses of the same field strength. Thus quadrupole lattices are generally favored over solenoidal ones because of a reduced size, power consumption, and decoupled particle motion in the transverse planes. These are the key reasons why alternating gradient (strong) focusing with the quadrupole lens is the main technology employed in many large-scale high-energy modern accelerators.

The map methods described in section 1.1.1 are typically employed to study transverse motion of the particles around the reference trajectory. The combined effect of accelerator lattice elements on the particle coordinates can be calculated by applying series of the maps of individual elements to the particles' initial phase space coordinates. In the linear case these maps can be obtained rather easily. They are simple matrices and their combined effect is calculated via a matrix multiplication. In linear optics (both light and magnetic), studies on the maps' properties provide a simple and elegant derivation of well-known physical laws of optics [31].

For nonlinear cases transfer maps can be described as polynomials for which the particle coordinates are variables (see (1.1.20)). As the required order of calculations goes up, their computation becomes more and more costly. In weakly nonlinear systems (which most accelerators are by design) the Differential Algebra framework can be successfully applied to easily obtain maps up to an arbitrary order [18].

If we consider a rotationally symmetric optical system and set x to be the position of the ray and m its slope, then the transfer matrix for its optical element can be conveniently denoted as

$$M = \begin{pmatrix} (x,x) & (x,m) \\ (m,x) & (m,m) \end{pmatrix}. \tag{4.1.1}$$

If the initial coordinates of the ray are given as $(x,m)^{\mathrm{T}}$ then its coordinates after this element are computed via

$$\begin{pmatrix} x_f \\ m_f \end{pmatrix} = \begin{pmatrix} (x,x) & (x,m) \\ (m,x) & (m,m) \end{pmatrix} \begin{pmatrix} x_i \\ m_i \end{pmatrix}. \tag{4.1.2}$$

Applying the maps of the elements in the system along the ray trajectory, we obtain the final coordinates. Imaging systems, i.e. optical systems in which the final position of a ray is independent of its initial angle and depends only on the initial position, must have a transfer map satisfying the following condition:

$$(x,m) = 0.$$

Now consider the linear uncoupled motion of the particles in the accelerator in the (x-a) and (y-b) planes. Suppose it is not radially symmetric. Applying a similar principle we see that in order for such system to be simultaneously imaging in both planes, its transfer matrix must possess the following property:

$$(x, a) = (y, b) = 0. \tag{4.1.3}$$

In light optics the smallest imaging system consists of drift, rotationally symmetrical lens and one more drift. As was mentioned earlier, in accelerators the quadrupole lens focuses in one plane and defocuses in other plane, hence the system that consists of the drift, quadrupole, and one more drift cannot achieve simultaneous imaging in both planes. Optical systems that are imaging in both planes are called *stigmatically* imaging or *point-to-point* systems.

A system that consists of two quadrupoles with alternating focusing planes is called a quadrupole doublet. Its net effect is focusing in both planes, but the focusing effects in x and y directions are different. If we use the convention that the first quadrupole is focusing in the x direction and defocussing in the y direction, then the calculations show that the orbit displacement is larger in the x direction and the focusing action in this direction is stronger. Hence the focal points for x and y are different and simultaneous imaging cannot be achieved (see [170] for detailed derivation).

Stigmatic imaging can be achieved by a system of three quadrupole lenses called a triplet. In order to achieve this property two outside quadrupoles are set to the equal strength and the same focusing direction, and the quadrupole in the middle is set to focus the beam in the perpendicular plane with a different strength. Quadrupoles in the system are almost always separated by drifts. The parameters of such system could be tuned so that it performs stigmatic imaging.

An example of tuning quadrupole triplet parameters to achieve stigmatic imaging can be found in the standard *COSY Infinity* Beam Theory package [22] in one of the demonstrations from the **demo.fox** file. In this particular case the system is built from a quadrupole of length 0.1 meters and strength $-q_1$ Teslas (negative strength convention denotes that it is defocussing in x direction); a drift of length 0.06 meters, another quadrupole of the same length and strength of q_2 Tesla, followed by the drift, and a quadrupole with the same parameters as the first two (i.e. system is symmetric with respect to the center quadrupole). In *COSY Infinity* such system is conveniently defined by the following sequence of commands:

```
MQ .1 -q1 .025 ;
DL .06 ;
MQ .1 q2 .035 ;
DL .06 ;
MQ .1 -q1 .025 ;
```

If we now define the function

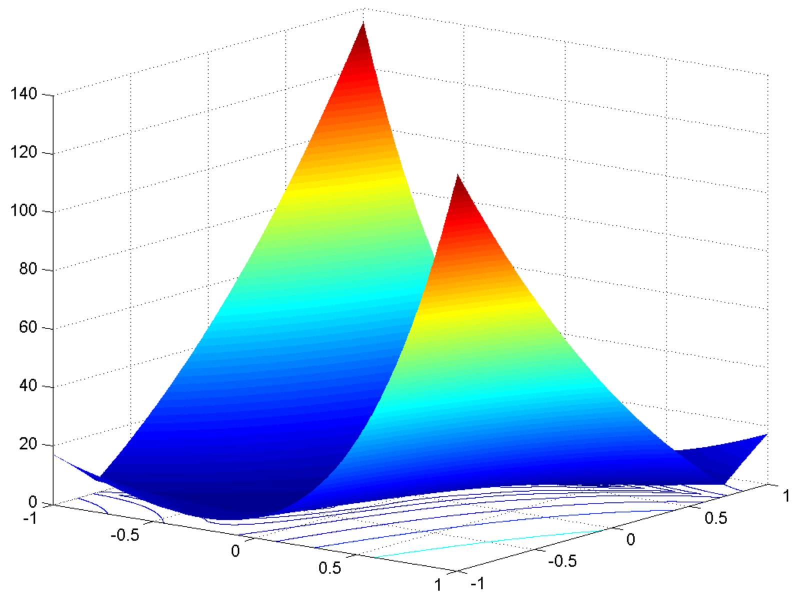
$$f(q_1, q_2) = |(x|a)| + |(y|b)| \geq 0, \ \forall q_1, q_2, \tag{4.1.4}$$

where $(x|a)$ and $(y|b)$ are the corresponding elements of the triplet transfer map, then for $\hat{q_1}$ and $\hat{q_2}$ such that

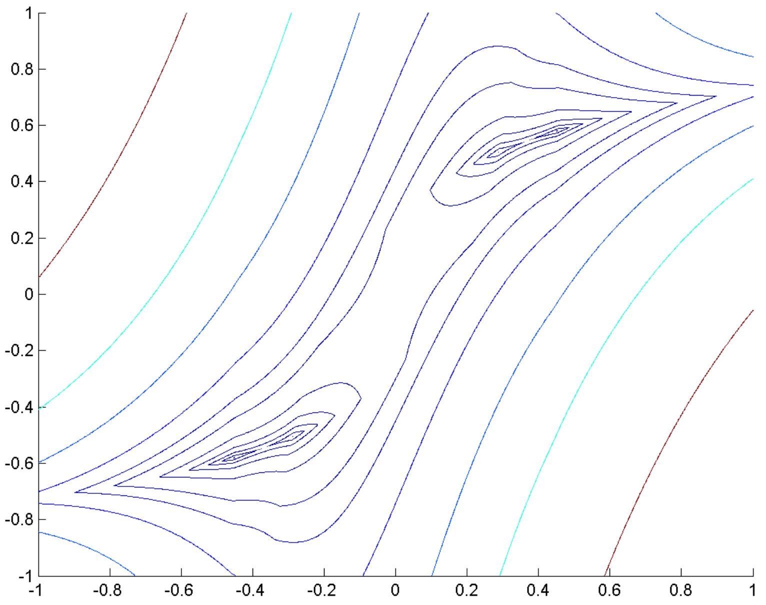
$$f(\hat{q_1}, \hat{q_2}) = 0,$$

also $(x|a) = 0$ and $(y, b) = 0$ hence the system is stigmatically imaging, according to condition (4.1.3). From the definition of this function it is clear that its minimum is at zero. The graph of the function (4.1.4) is shown in Figure 4.1.

By looking at the contour plot we can roughly guess that the objective function has four extrema. From the 3D plot we can deduce that all four of them are minima. Using the *COSY Infinity*'s built-in SIMPLEX method as it is demonstrated in **demo.fox** we can find all four of them. But the problem here is that it requires us to provide



(a) 3D plot



(b) Contour lines plot

Figure 4.1: *Objective function for triplet stigmatic imaging* $f(q_1, q_2)$, $q_i \in [-1, 1]$, $i = 1, 2$

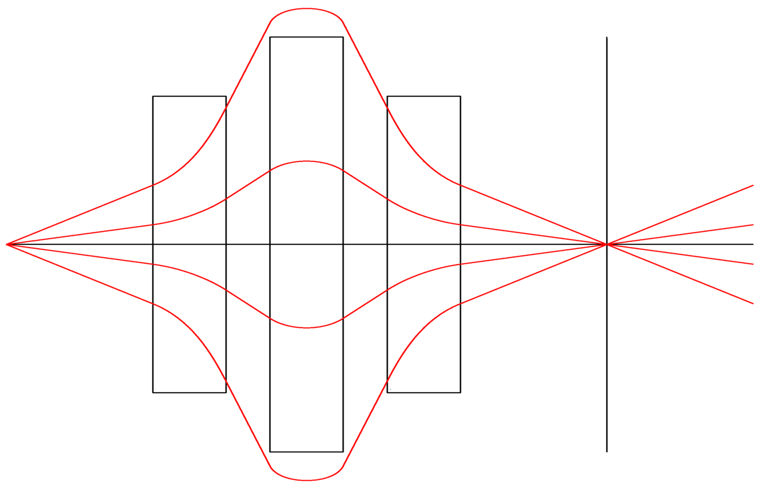
good initial guesses in the domain of attraction of each minimum (starting from the same initial guess the SIMPLEX method always converges to the same result, i.e. it is deterministic, see Figure 2.4). In a two-dimensional problem we can search for such points relatively easily, using visual tools such as a contour plot. However, for a high-dimensional problem this task is often far from trivial. Even if we perform very fine-grained sampling of the search space (which gets prohibitively expensive as the dimensionality goes up) we may not find good starting points since the search space volume is too large.

Once we know the locations of these four extrema, we can find them using one of the standard optimization methods. Their values with a precision up to 10^{-5} are:

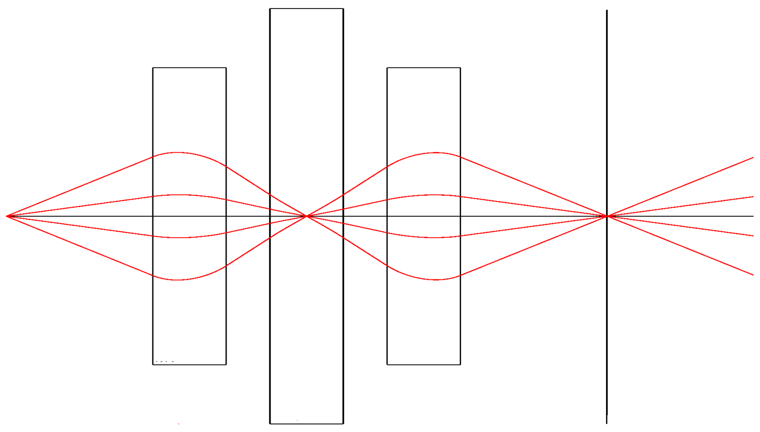
1. $q_1 \approx 0.452$, $q_2 \approx 0.58$,
2. $q_1 \approx 0.288$, $q_2 \approx 0.504$,
3. $q_1 \approx -0.288$, $q_2 \approx -0.504$,
4. $q_1 \approx -0.452$, $q_2 \approx -0.58$.

Points 1 and 4, and 2 and 3 are symmetric relative to the origin, since the quadrupole is reflection symmetric in both planes even though the focusing/defocussing pattern reverses (the change of the sign of the quadrupole strength just flips the direction of focusing). With these quadrupole strengths the system is imaging in both planes, hence it stays imaging. It is worth noting that the paths of the particles in the (x-a) and (y-b) planes, corresponding to symmetric solutions, also interchange between Figures 4.2, 4.3 and 4.4, 4.5), correspondingly.

We ran GATool on the same objective function using the default parameters (see Figure B.1, p.273), choosing the population size to be 100 times the dimension which

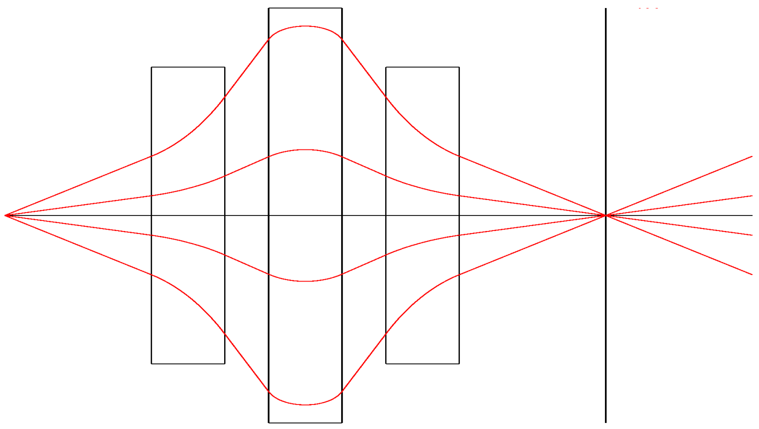


(a) (x-z) projection

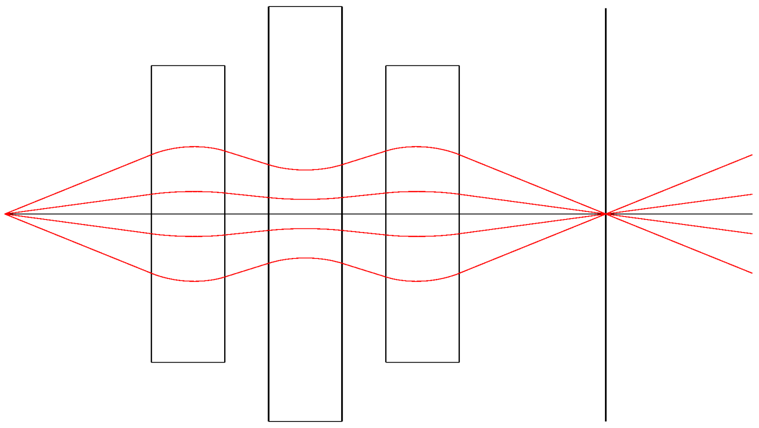


(b) (y-z) projection

Figure 4.2: *Ray tracing of the triplet, solution 1*

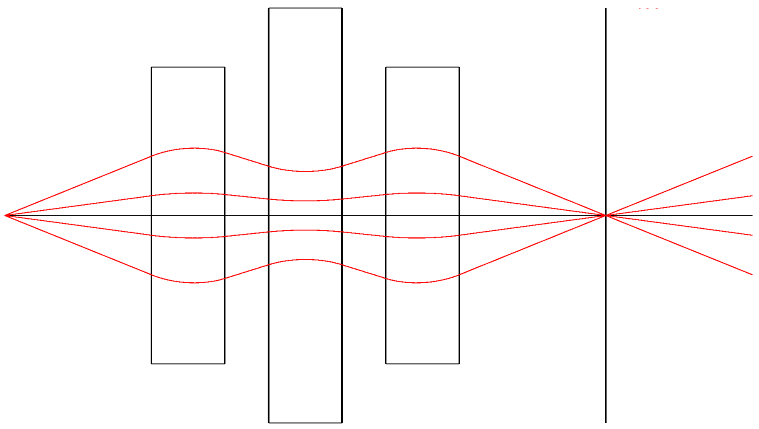


(a) (x-z) projection

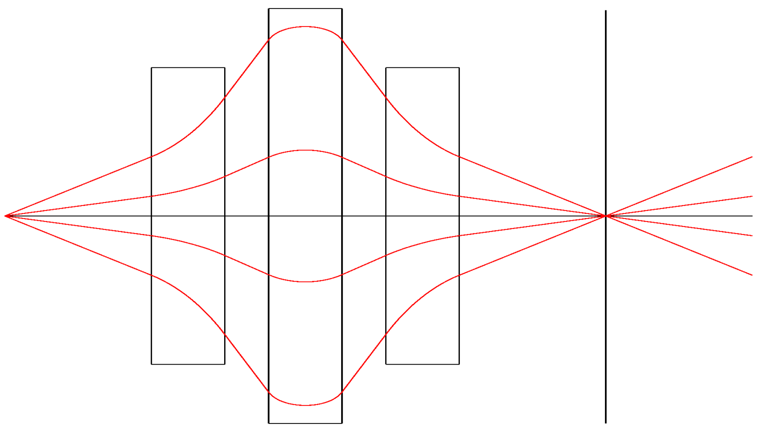


(b) (y-z) projection

Figure 4.3: *Ray tracing of the triplet, solution 2*

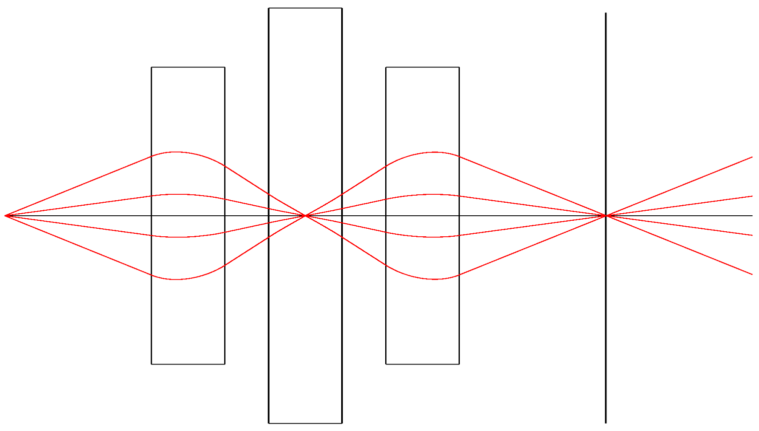


(a) (x-z) projection

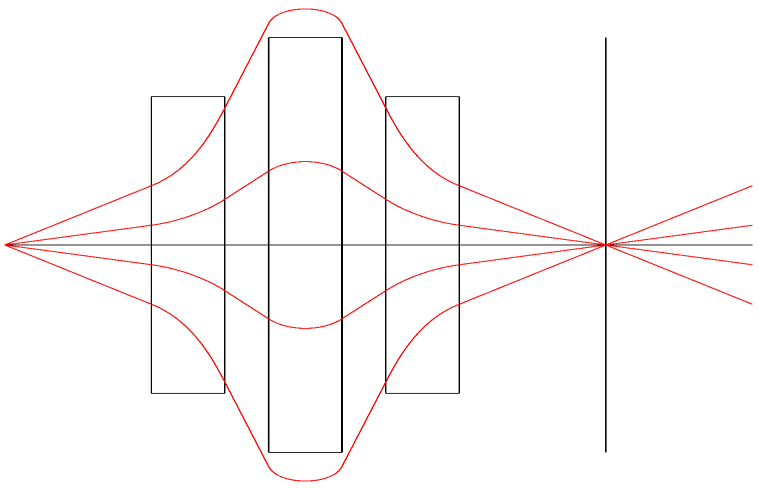


(b) (y-z) projection

Figure 4.4: *Ray tracing of the triplet, solution 3*



(a) (x-z) projection



(b) (y-z) projection

Figure 4.5: *Ray tracing of the triplet, solution 4*

Table 4.1: *Triplet stigmatic imaging design statistics*

# Runs	Solution found (%)			
	1	2	3	4
200	12.0	46.5	36.0	5.5
1000	9.0	46.9	37.0	7.1
3000	4.7	31.3	60.3	3.7
10000	8.18	47.27	38.19	6.36

is 2 in our case. The stopping criteria was set to the maximum number of stall generations (see section 2.3) and the search domain was $[-10.10] \times [-10, 10]$. Taking into account physical considerations about the strengths of non-superconducting quadrupoles, we should have searched for q_1, q_2 in a realistic interval $[-1, 1]$ but we wanted to demonstrate GATool's usefulness for exploration even if the information about a search domain is scarce and approximate. Constraints in general reduce the search space volume thus simplifying the task of the optimizer.

Each run of the GATool successfully finished the search on one of the solutions 1-4. Since the Genetic Algorithm is a stochastic optimization method, it does not guarantee the result will be the same on each run. Running the algorithm 200, 1000, 3000, and 10,000 times with the same set of parameters and search domain and then analyzing the resulting solutions we get Table 4.1. One important observation here is that GATool was able to find one of the global minima on every run without needing to specify an initial search point. Second observation is it was able to find all solutions to the problem in just 200 runs. These observations demonstrate that GATool is a valuable method to perform at least an initial study of a system with complicated behaviour, complex dependences on control parameters, a sophisticated structure, and a non-analytic objective function. Such conditions often arise during the accelerator design. Even though this method does not guarantee a global minimum, it frequently provides excellent insight into the behaviour of the system and is usually able to find a good upper bound for the minimum. It can then serve as a starting point or a cutoff value for another optimization method as it is described in section 2.3.6.

Some of the minima (1-st and 4-th solutions) are more difficult for the optimizer to find. We believe that non-symmetric percentages for symmetric solutions were introduced by the random number generator implementation details but this question requires further investigation which is out of the scope of this work. We suggest running GATool a sufficiently large number of times in order to get a better estimate of the minimum value (or values).

In the case of the objective function (4.1.4), insights into the fact that solutions 1 and 4 are harder to find can be obtained by studying the contour lines of the function (see Figure 4.1). The contour lines indicate that these minima are sharper than the other two and that their domains of attraction are much smaller. As we noted earlier, a graphical method of system investigation is not always available.

It is also worth noting that, in principle, the linear transfer map of the quadrupole can be calculated analytically using the expressions for the magnetic field

$$\begin{aligned} B_x &= 2ky \\ B_y &= 2kx, \end{aligned}$$

where k is a quadrupole strength, and x, y are transverse coordinates. Plugging them into the equations of motion (1.1.8)–(1.1.13), linearizing them and then solving, we can directly obtain the transfer matrix of the system. The linear map of the drift is the same as it is in light optics. Thus we can calculate the map describing the combined action of the triplet by multiplying known maps. Then, using strengths of the quadrupoles as variables, we can obtain an analytic expression for the objective function (4.1.4).

However, the algorithm described in this section and its implementation in *COSY Infinity* are applicable to any map element, including non-linear elements (which are much harder to obtain), and its combinations, including non-analytic. *COSY Infinity* efficiently calculates transfer maps to an arbitrary order, giving the user a powerful tool to build complex yet relatively computationally inexpensive objective functions that describe the desired properties of accelerating structures. Then, applying the method described in this section, the user can tune the control parameters to achieve the design goals.

It is worth noting that this particular example problem is relevant to the frequently encountered collider Interaction Region design problem. Here, strong Final Focus Telescope (FFT) quadrupoles are required to focus the beam in both planes to extreme sizes at the low-beta Interaction Point (IP) where the beams collide. All of the current approaches and techniques fail to find an adequate minima for the IP optics [97]. These accelerator codes start from the assumption of the optically small beta sizes at the IP and attempt to fit and match the strengths of FFT quadrupoles to optical regions outside of the IR. Even with this simplification, the sensitivities are such that the solutions are highly oscillatory, thus additional constraints are often imposed to find an acceptable solution. The optical designs of the advanced systems such as forefront colliders, the International Linear Collider, the Muon Collider, and the Linear Hadron Collider with most modern codes create a difficult and often intractable problem.

The considered example highlights the power of GATool in uncovering solutions that are difficult to find for the problems originating in advanced accelerator optics. Thus this work is of a particular importance to the advanced accelerator design field, especially since most modern optimization algorithms break down for parameter spaces of large dimension and volume. Based on this evidence we conclude that GATool is a significant addition to an advanced accelerator designer's tool set.

4.2 Normal Forms Defect Function Optimization

Normal form defect functions, described in section 1.1.2, are very useful for rigorous estimation of the stability of a circular accelerator. Deviations from the invariant circles they are measuring, directly influence the number of stable turns particles stay in an accelerator before being lost. The maximum of a deviation allows us to get a lower bound of the number of turns particles stays within the considered region.

The difficulty in this seemingly solved problem is that these functions are multidimensional polynomials of the order of up to 200 [125] (thus they are very oscillatory) with many high-order terms that cancel each other out during the function evaluation. Conventional optimization methods do not perform well on such functions and usually get stuck in one of the local extrema. See Figures 4.7, 4.11, particularly phase angle dependence plots for examples of the landscapes of these functions. Conventional methods of global optimization (such as various flavors of interval methods) [83, 101] suffer from the cancellation and the clustering effects [26]. Taylor model methods (see section 2.2.2), however, allow one to obtain tight rigorous estimates of the maximum, even under these unfavorable conditions. While in this case tight estimation is practically doable, such a daunting task still requires a tremendous amount of computation time. This time can usually be reduced when there is a method to provide good cutoff values to the box elimination algorithm. Here by a cutoff we mean a lower bound for the maximum. Knowing the lower bound we can safely cut off all the candidates (in our case boxes for maximum) that are below this bound from the future considerations, thus saving computing resources and speeding up the search process.

The box elimination algorithm is one of the main parts of the rigorous global optimization process of COSY-GO, hence its execution time heavily contributes to the execution time of the whole search. We claim that having a cutoff value generated by the GATool optimizer (see section 2.3.6) is advantageous for COSY-GO and it leads to a reduced computation time.

In this section we study the performance of GATool on normal form defect function optimization in order to:

- assess its performance on the complex high-dimensional multi-modal function
- compare the computation time and the quality of the results with the ones obtained by the Taylor model methods-based rigorous global optimizer developed by Youn-Kyung Kim [103] in order to study the potential for combining COSY-GO and GATool into one hybrid method

At first, a less complex (and thus potentially easier to optimize) synthetic defect function in 6 variables (three pairs of phase radii and angles) based on the generated polynomials available at the `http://bt.pa.msu.edu` was considered on the search domain from Figure 4.6.

```
[ 0.499999999E-001, 0.100000001 ]  [ -3.14159266,  3.14159266 ]
[ 0.499999999E-001, 0.100000001 ]  [ -3.14159266,  3.14159266 ]
[ 0.499999999E-001, 0.100000001 ]  [ -3.14159266,  3.14159266 ]
```

Figure 4.6: *Synthetic normal form defect function domain of interest*

Projection on the space of two phase angles is demonstrated in Figure 4.7 (fixed values of the radii are equal to 0.1). Note that the oscillatory behavior of the function

Table 4.2: *GATool's performance for different population sizes compared to the performance of the Taylor model methods-based global optimizer (TMMGO) and the Naive Sampling method, on the synthetic normal form defect function (see Figure 4.7). TMMGO was executed on 256 IBM SP POWER3 processors 375 MHz each, GATool and Naive Sampling were executed on 1 Intel Pentium IV 2 Mhz processor. *For TMMGO time is given as a number of processors × wall clock time of the run*

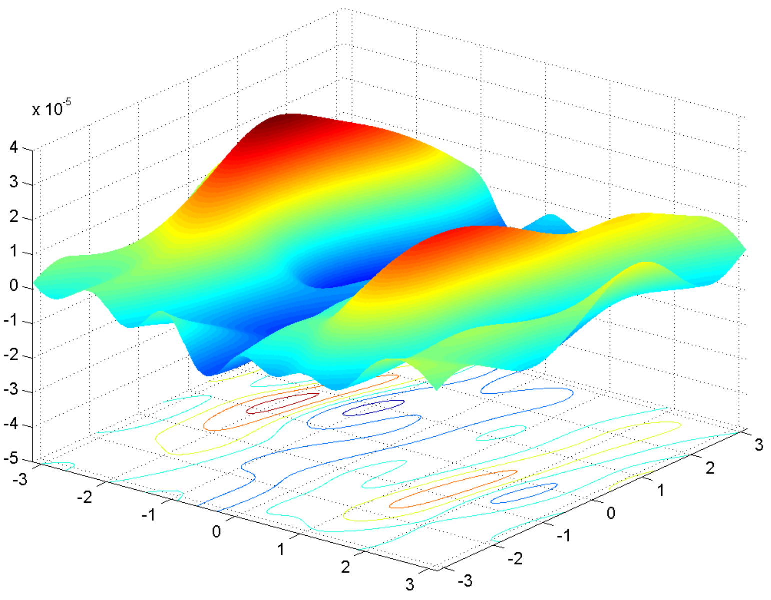
Method	Time (s)	Max Value	Difference with COSY-GO
TMMGO	256 x 3297*	-	[-, -]
Naive Sampling	109	0.209075292E-4	[1.28294580E-5, 1.28294626E-5]
GATool, pop=60	17	0.327416142E-4	[9.95373092E-7, 9.95377677E-7]
GATool, pop=180	83	0.319524687E-4	[1.78451855E-6, 1.78452314E-6]
GATool, pop=300	300	0.332044502E-4	[5.32537049E-7, 5.32541634E-7]
GATool, pop=600	378	0.331694477E-4	[5.67539577E-7, 5.67544162E-7]
GATool, pop=1000	553	0.332085478E-4	[5.28439469E-7, 5.28444054E-7]
GATool, pop=1200	613	0.336515785E-4	[8.54087318E-8, 8.54133164E-8]
GATool, pop=2000	3459	0.337010630E-4	[3.59242826E-8, 3.59288671E-8]

is very prominent.

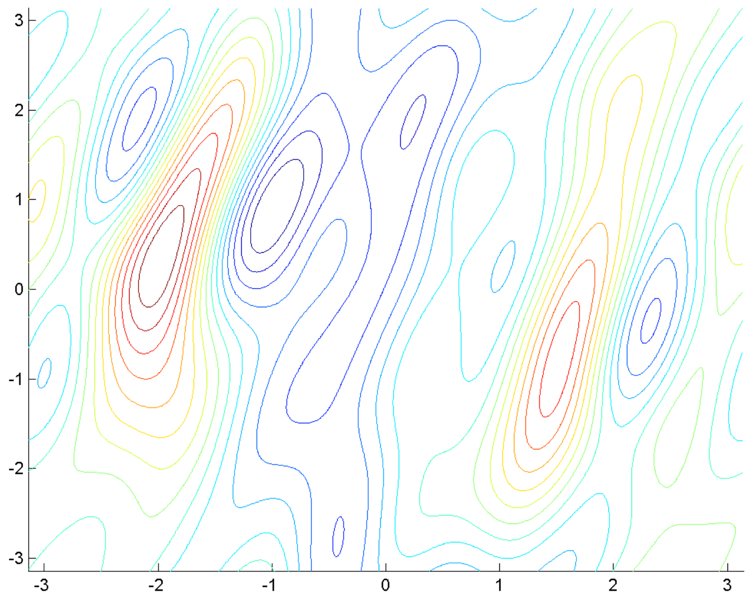
The results obtained from the Taylor model methods-based global optimizer used for the rigorous estimation of this function's maximum [26, 103] are summarized in Figure 4.8,

Results obtained from GATool with parameters from Figure 4.9 and initial box from Figure 4.6 are summarized in Table 4.2. Results obtained by naive sampling of the search space are presented for comparison. The number of samples is equal to the maximum number of function evaluations made by GATool during the search (over all population sizes). Timing for Taylor model methods-based global optimizer (TMMGO) is calculated as a product of the number of processors used and the total execution time measured by a wall clock. We should note, however, that parallel execution introduces unavoidable overhead for communications which should be subtracted from the total execution time, but for our problem it is relatively small hence this correction is neglected.

From the comparison table it is evident that even with a population size that is



(a) 3D plot



(b) Contour lines

Figure 4.7: *Synthetic normal form defect function plots. Function values vs two phase angles*

```
SAMPLE NORMAL FORM DEVIATION FUNCTION (RADII IN [0.05,0.1])
STOPPING CONDITION 1 HAS BEEN MET.
NUMBER OF PROCESSES: 256
NUMBER OF ITERATIONS: 2013
WALL CLOCK TIME: 0 hr 54 min   57.32927064225078 sec
WALL CLOCK TIME IN SECONDS:   3297.329270642251 sec
ORDER OF TAYLOR MODELS USED: 5
TAYLOR MODEL BOUNDING METHOD: REDB
MAXIMUM LIST SIZE: 503064
FINAL LIST SIZE: 16
NUMBER OF SMALL BOXES IN THE LIST: 0
INTERVAL ENCLOSURE FOR THE MAXIMUM:
[0.3373698730538533E-04, 0.3373699188996994E-04]
WIDTH:   .4584584624369067E-11
```

Figure 4.8: *COSY-GO output on synthetic normal form defect function maximization*

```
Reproduction:    number of elite = 10, mutation rate = 0.2
Mutation:        UNIFORM, gene mutation probability = 0.1
Crossover:       HEURISTIC, ratio = 0.8, randomization is on
Fitness scaling: RANK
Selection:       STOCHASTIC UNIFORM
Creation:        UNIFORM
                 killing is on
Stopping:        max generations = 1000,
                 stall generations = 25,
                 tolerance = 1E-9
```

Figure 4.9: *GATool's parameters used for synthetic normal form defect function maximization*

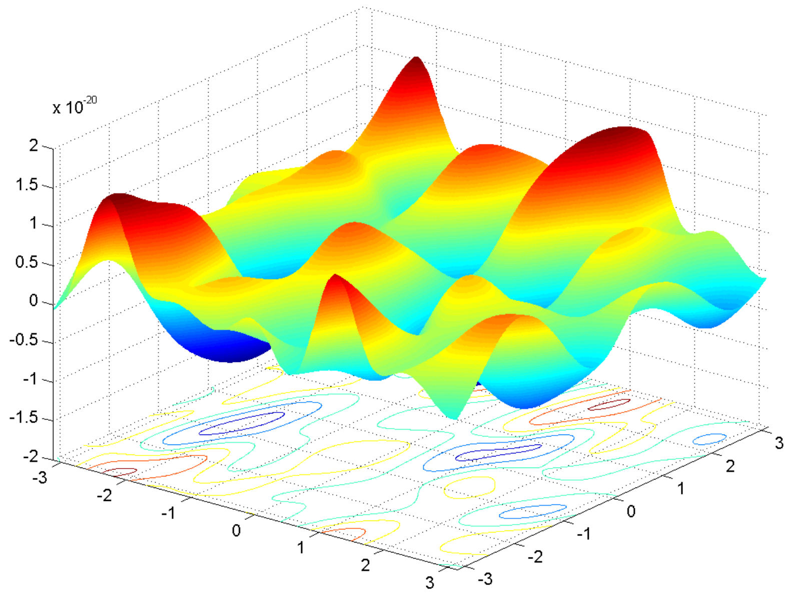
just 10 times the dimension of the problem (60), GATool is able to provide a good estimate (naive sampling estimates the range of the function when the values are in the search box as $0.5 \cdot 10^{-4}$) of the lower bound of the maximum. As can be seen from the table, an increase in the population size generally leads to an increase in the quality of the estimate. However, as could be seen for the population of the size 180, this is not always the case and even runs with larger populations can occasionally perform worse than runs with smaller populations. Performing a statistically sufficient number of runs for each population size, we can verify that for the problem under consideration the quality of the obtained estimate averaged across runs is getting better as the population size increases.

Another normal form defect function is computed for a real large circular accelerator and thus contains a lot of nonlinear elements which makes it generally harder to optimize. It is computed by P. Snopok for the Tevatron accelerator [159] located at Fermi National Accelerator Laboratory. Here we were interested in estimating a maximal defect on a particular circular region of the phase space defined in Figure 4.10.

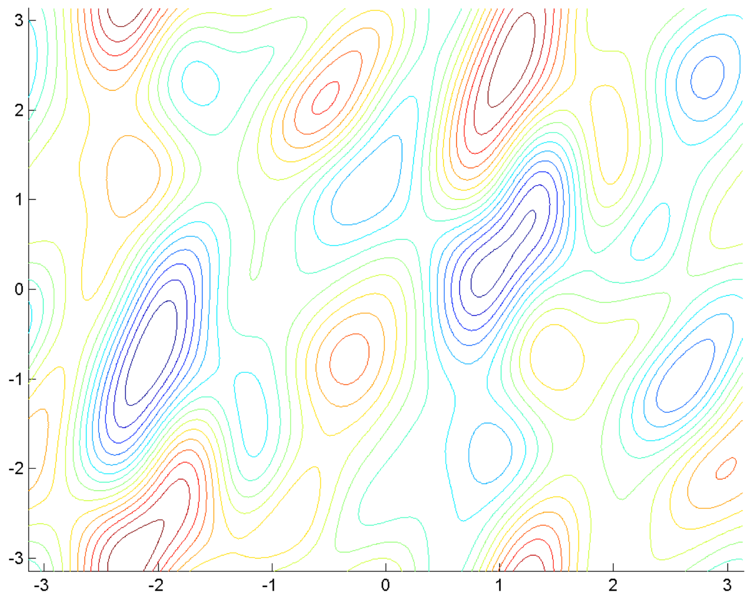
```
[ 0.199999999E-004, 0.400000001E-004 ]  [ -3.14159266,  3.14159266 ]
[ 0.199999999E-004, 0.400000001E-004 ]  [ -3.14159266,  3.14159266 ]
```

Figure 4.10: *Tevatron's normal form defect function domain of interest*

A projection on the space of the two phase angles is demonstrated in Figure 4.11 (fixed values of radii are equal to $0.4 \cdot 10^{-4}$). The oscillatory behavior similar to the synthetic function is clearly recognizable. The dynamics of the particles in Tevatron that resulted from applying the one-turn transfer map in conventional and normal form coordinates is demonstrated in Figure 4.12. This map was used for the normal form defect function computation.

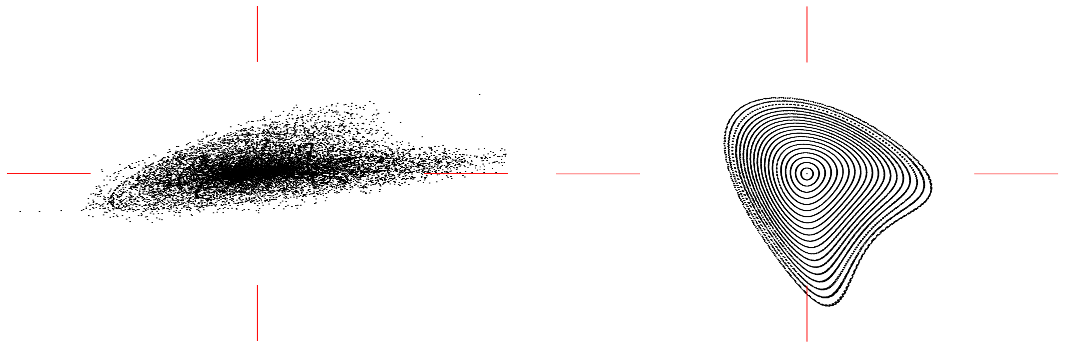


(a) 3D plot



(b) Contour lines

Figure 4.11: *The Tevatron normal form defect function. Function values vs two phase angles*



(a) Conventional coordinates (b) Normal form coordinates

Figure 4.12: *Particles dynamics in the Tevatron*

Results of applying COSY-GO to the rigorous estimation of this function's maximum [26,103] are summarized in Figure 4.13,

```
NORMAL FORM DEVIATION FUNCTION FOR TEVATRON
STOPPING CONDITION 0 HAS BEEN MET.
NUMBER OF PROCESSES: 1024
NUMBER OF ITERATIONS: 326
WALL CLOCK TIME: 0 hr 15 min   35.44252496212721 sec
WALL CLOCK TIME IN SECONDS:   935.4425249621272 sec
ORDER OF TAYLOR MODELS USED: 7
TAYLOR MODEL BOUNDING METHOD: REDB
MAXIMUM LIST SIZE: 999424
FINAL LIST SIZE: 998400
NUMBER OF SMALL BOXES IN THE LIST: 0
INTERVAL ENCLOSURE FOR THE MAXIMUM:
[.3846166509606185E-18,  .7114456197038863E-13]
WIDTH:   .7114417735373770E-13
```

Figure 4.13: *COSY-GO output on the Tevatron normal form defect function maximization*

Applying GATool with parameters from Figure 4.9 (only relative tolerance was changed to $1 \cdot 10^{-25}$ to reflect much smaller function values), and the initial box from Figure 4.10, we obtained results summarized in Table 4.3. As in Table 4.2, results obtained by naïve sampling are presented for comparison.

Table 4.3: *GATool's performance for different population sizes compared to the performance of the Taylor model methods-based global optimizer (TMMGO) and Naive Sampling methods on the Tevatron normal form defect function (see Figure 4.11). TMMGO was executed on 256 IBM SP POWER3 processors 375 MHz each, GATool and Naive Sampling were executed on 1 Intel Pentium IV 2 Mhz processor. *For TMMGO time is given as a number of processors × wall clock time of the run*

Method	Time (s)	Max Value	Difference with COSY-GO
COSY-GO	1024 x 935*	-	[-, -]
Naive Sampling	46	0.384215054E-18	[4.01596187E-22, 7.11441777E-14]
GATool, pop=40	5	0.380347985E-18	[4.26866555E-21, 7.11441816E-14]
GATool, pop=200	18	0.382665745E-18	[1.95090547E-21, 7.11441793E-14]
GATool, pop=400	75	0.384126132E-18	[4.90518103E-22, 7.11441778E-14]
GATool, pop=600	177	0.384406960E-18	[2.09690285E-22, 7.11441775E-14]
GATool, pop=800	117	0.384035970E-18	[5.80680790E-22, 7.11441779E-14]
GATool, pop=1000	230	0.384644775E-18	[-2.81241401E-23, 7.11441773E-14]

Notice that the last value of the maximum obtained by GATool for a population size of 1000 is inside the rigorous enclosure established by COSY-GO, hence we might conclude that GATool improved the lower bound for the maximum obtained by COSY-GO. However, this effect might also be attributed to floating point operation errors that is made significant by the values of the considered function being close to the machine precision. Such errors are treated by COSY-GO in a rigorous way via outward rounding for the interval calculations. GATool uses standard floating points operations and thus is susceptible to numerical inaccuracies. Therefore the numbers from the table can be used to demonstrate only the growth of the quality of the estimate obtained by GATool with the growth of the population size.

If a more rigorous result is needed, *COSY Infinity*'s object-oriented features allow the user to easily overload standard floating point arithmetic with high-precision arithmetic. Development of the high-precision arithmetic package for *COSY Infinity* is currently underway [27, 174]. It is worth noting that this effort is partially inspired by a normal form defect function rigorous bounding problem.

In this section we demonstrated that GATool can be satisfactory applied to the practically useful problem of estimating the extrema of a complex multidimensional function. We showed that the quality of the result together with the computation time support the usage of GATool as a fast generator of good cutoff values for COSY-GO. It should be noted, however, that a cutoff is a lower bound of the maximum. GATool itself can not be used to estimate the upper bound of the maximum of the normal form defect function. COSY-GO is needed to accomplish this task with GATool working in parallel to reduce the overall computation time. As is discussed in section 2.4, the integration of these methods is a topic for the future research. Normal form defect function bounding problem would server a good test to assess the performance of this combined tool.

4.3 Neutrino Factory Front End Design Optimization

4.3.1 Problem Description and Motivation

The Neutrino Factory, as is described in section 1.2, is an important facility for a future of the neutrino research program, and is currently in the active R&D stage [178]. Its designs are frequently changed and explored in search for the optimal solution and the cost/performance ratio. Such a solution would allow for the international collaboration (mostly members are from USA universities and laboratories) to realistically consider building this next-generation accelerator [10]. The front end section plays an important role in the overall performance of the factory. It conditions the high emittance beam coming from the production target for the subsequent acceleration (see section 1.2.3).

Effective delivery of the particles with optimal phase space formation (to match the transverse acceptance and the acceleration regime) and minimal particle losses are key performance characteristics of the Front End. Thus one quantity that requires optimization is the ratio of the muons matching the accelerating regime at the end of the channel to the number of the pions coming from the target, i.e. production efficiency. It is also one of the main factors in achieving the primary goal of the whole accelerator, namely producing high-intensity beams of neutrinos for various experiments.

From the description of a currently accepted baseline front end in section 1.2.3 it can be easily concluded that there are many variations in the lattice parameters that can potentially lead to different performance characteristics. Since the front end is just a subsection of the Neutrino Factory, it needs to fit into a general scheme, which means that its performance cannot be considered alone. Rather it should be tuned to fit optimally into the overall accelerator design. However, there are different suggested designs of the subsections that precede and follow the Front End, and different variants of the Front End itself, therefore different optimizations might be required in order to explore all possibilities to their full extent. Hence, it is important to establish a general scheme of exploration and optimization that can be applied to study any of these variants. Some of the factors that should be considered or explored for the Front Ends include: physical limits on the maximum gradients that can be obtained in RF cavities or a number of RF cavities with different frequencies; schemes that provide shorter or longer bunch trains; optimization of the production parameters (the number of muons captured into the accelerating regime); different central energies of exiting bunches; different allowed energy spread, and, of course, the cost considerations. Matching the beam into different accelerating/cooling structures following the considered lattice also has to be taken into account.

Optimization studies that address some of these issues [10, 71, 74, 95, 109, 145, 152, 154,155] are summarized in the yearly summary reports produced by the international collaboration working on the Neutrino Factory project [1, 62, 86, 150, 178]. However, given the fact that there was not general agreement on the design and there are still a lot of the variations that were to be considered, we were motivated to perform such a study ourselves. Another motivation lied in the fact that Evolutionary Algorithms had demonstrated themselves as an efficient tool for design exploration and optimization (see section 2.1). Thus the application of the GATool algorithm (see section 2.3) to the problem of a front end optimization was interesting both from the practical point of obtaining new optimal designs and as a test of the algorithm performance on a complicated real-life problem. Here the complexity of the problem lied in the objective function that was not defined analytically and included stochastic simulations. Moreover, successful application of the algorithm to this problem would have established a general scheme of front end optimization which could be used for the subsequent studies. Successful application of the EA to an exploration of one of the front end designs [32] served as another factor that lead us to believe in the success of the experiment.

4.3.2 Optimization of the Front End Production Parameters

As can be seen from the description of the Front End design, parameters that can be changed for different sections (see section 1.2.3) include:

1. *Capture and Decay:* the length of the section L_{D} and the focusing fields.

2. *Bunching:* the length of the section L_{B}, RF voltages V_{B}^{i}, $i = \overline{1, n_{\mathrm{rfs}}}$ or initial

and final voltage and the voltage increase formula (linear, quadratic, etc.). Final frequency is usually strictly specified by the cooling/accelerating subsections of the whole accelerator, but can be varied if it positively influences the overall Neutrino Factory performance.

3. *Phase Rotation:* the length $L_{\varphi\mathrm{R}}$, RF voltage $V_{\varphi\mathrm{R}}$ of the phase-energy rotation section, number N of RF field oscillation periods between chosen second central particle and the main central particle (with $n = 0$), and the vernier parameter δ. Also the kinetic energy T_c of the main central particle can be changed (usually T_c is taken as the peak of energy distribution of the particles of the beam).

4. *Ionization Cooling:* parameters of the RF cavities ($\nu_{\mathrm{rf,cool}}$, $V_{\mathrm{rf,cool}}$, $\varphi_{\mathrm{rf,cool}}$); position, width, material, and the position of absorbers, and focusing field.

For our study we explore the cooling section. We varied RF cavities parameters and the momentum of the central particles in the beam within the ranges obtained from the physical considerations. We also optimized the mathematical model of the structure in order to find a configuration which would provide the maximum particle production described earlier.

Most of the numerical studies of beam dynamics in the Front End are performed in ICOOL — the de-facto standard Muon Collaboration particle tracking code. It was originally developed in 1999 for ionization cooling simulations of muon beams [66] and has been actively developed over years to include new elements and models [69] (available at `http://pubweb.bnl.gov/users/fernow/www/icool/readme.html`).

ICOOL belongs to the family of the so-called ray tracing codes. It calculates particle dynamics employing the Runge-Kutta or the Boris numerical integration methods to integrate equations of motion. The dynamics is studied in Frenet-Serret coordinate system [29], which is a right-handed system where s is tangent to the reference orbit, y is vertical, and x is the third orthogonal coordinate. In a circular orbit x grows in the radial direction. The reference orbit is defined to be the path where the transverse coordinates x and y and the transverse momenta p_x and p_y remain zero. The shape of the reference orbit in a global Cartesian coordinate system is determined by the curvature parameter.

The electromagnetic field can be specified using built-in models that include most common accelerator elements and their approximations. It can be calculated from the field maps or Fourier coefficients, or read from external sources. ICOOL accurately models the decay processes and particle interactions with matter including energy loss, energy straggling and multiple Coulomb scattering [68, 70]. The beam can be generated from the uniform or Gaussian distributions or read from an input file.

Various tools are developed to analyze the results produced by ICOOL. The standard code for the emittance calculation is called ECALC9 [67]. It allows a user to compute the number of particles in the fixed phase space volume. The input is read from a file that contains the particle type, maximum, and minimum value for p_z in GeV/c, two different cuts for the transverse acceptance in m·rad (to obtain a number of particles that are left after two different acceptance cuts when all other cuts staying the same at once), a longitudinal acceptance cut in m·rad and an RF frequency to determine the RF bucket area for the longitudinal cut.

The tool chain for the optimization of the production parameters was assembled from the following pieces:

- *COSY Infinity:* provided the implementation of the GATool optimization method (see section 2.3 for the description of the algorithm and Appendix B for the technical details of the implementation).

- *ICOOL:* performed actual simulations of the beam dynamics in the Front End with the parameters values passed from COSY.

- *ECALC9:* performed analysis of the results of the ICOOL simulations, calculated the number of particles within the desired acceptance (and thus the production ratio), that served as the objective function value.

- *Perl:* used to control other programs in the tool chain and pass parameters and values between them. It was used to set up the Front End lattice for ICOOL based on the control parameters provided by COSY, run ICOOL and then ECALC9 to obtain the objective function value, and finally pass it back to COSY to complete one optimization step.

The initial distribution of particles coming from the target contains 8000 particles. It was generated by MARS simulation code for the 24 GeV proton beam on the Hg jet target [138]. The Front End lattice that was used for this study started from the target and included capture, decay, bunching, and phase rotation regions as well as a cooling section and a matching between phase rotation and cooling subsystems:

- Capture: 15.25 m of the vacuum channel in a solenoidal field that falls off starting from 20 T on the target to 2 T at the end of the channel. At the same time the radius of the channel increases from 0.075 m to 0.3 m.

- Decay: vacuum channel of a constant aperture of 0.3 m in a constant solenoidal field of 2 T.

- Bunching: vacuum channel of a constant aperture of 0.3 m and a total length of $L = 21$ m in a constant solenoidal field of 2 T. An array of RF cavities separated by drifts where the parameters were calculated according to logic described in section 1.2.3 so as to perform the adiabatic bunching (28 cells, each consists of the drift of 0.125 m, followed by RF cavity of 0.5 m and another drift of 0.125 m). Particles are bunched around the central momentum of 0.280 GeV/c. An integer number of wavelengths that separate two reference particles (n in (1.2.2)) is 7 (hence the momentum of the second central particle according to (1.2.5) is 0.154 GeV/c) and the initial RF gradient is set to 15 MV/m, the formula for the RF gradient dependence on the longitudinal coordinate z, measured from the start of the buncher, is

$$V_{\mathrm{rf}} = V_{0,\mathrm{rf}} \frac{z}{L} .$$

- Phase rotation: vacuum channel of a constant aperture of 0.3 m and a total length of $L = 24$ m in a constant solenoidal field of 2 T. An array of RF cavities separated by the drifts with the parameters calculated according to logic described in section 1.2.3 so as to perform the rotation of the beam in the longitudinal space by decelerating higher-energy bunches and accelerating lower-energy ones (32 cells, each consists of the drift of 0.125 m, followed by RF cavity of 0.5 m and another drift of 0.125 m). Vernier offset δ from (1.2.8) is 0.1, RF gradient is 15 MV/m for all cavities.

- Cooling: vacuum channel of a constant aperture of 0.3 m and a total length of $L = 93$ m in an alternating solenoidal field of the maximum strength ≈ 2.5 T. An array of the 124 cells (0.75 m each), with LiH absorbers to achieve total momentum loss and an RF fields to achieve longitudinal momentum regain, are combined together in order to cool the transverse emittance of the beam. The first four cells have the solenoidal field designed so as to match the transverse particle dynamics in phase rotation section to the one of the cooling section.

All RF cavities have a frequency of 201.25 MHz, field gradient is 18 MV/m and the RF phase is 30 degrees.

This particular design was shorter than the one of the baseline and it was developed to study the cost gain versus the performance losses resulted from shortening the Front End. It achieved this tradeoff by removing some of the elements of the baseline. Another goal was to study the potential applicability of this design for the Muon Collider project [46].

We used the described lattice as a reference design and explored its performance related to changes in the following control parameters:

- RF frequency in the cooling section (also influences the downstream accelerator section): $\nu_{\mathrm{rf,cool}} \in [200, 204]$ MHz.
- RF field gradient in the cooling section: $V_{\mathrm{rf,cool}} \in [12, 20]$ MV/m.
- RF field phase in the cooling section: $\varphi_{\mathrm{rf,cool}} \in [0, 360]$ degrees.
- Central momentum in the first four matching sections of the cooling channel: $p_{\mathrm{c,match\_cool}} \in [0.22, 0.24]$ GeV/c.

The values of the cuts for ECALC9 analysis were selected to estimate of the acceptance of the subsequent acceleration subsystem [143]:

- minimum and maximum p_z: 0.100 GeV/c and 0.300 GeV/c, correspondingly;
- transverse acceptance cut: 30E-3 m·rad;
- longitudinal acceptance cut: 0.25 m·rad;
- RF frequency for the bucket calculation set to a value used by RF cavities of the cooling section (on of the control parameters).

The number of particles within the specified acceptance (n_2) [67] at the end of the lattice was chosen as an objective value to be maximized. The initial number of particles was kept constant so the targeted quantity was the production ratio. GATool parameters were set to default values (see Figure B.1, p.273) and the population size for this 4-dimensional problem was set to 250 (dimension$\times$62.5). Such choice provided a good compromise between the total time of the search given the expensiveness of the objective function calculation (see below), and the quality of the GATool result (see studies in section 2.3.6).

Several of the best obtained results (elite in GATool terminology) from three runs (each of them took several months to complete on a single machine) were evaluated using the described scheme and the full initial number of particles of 8000 (2000 of which were used in optimization to reduce computation time). The control parameters and the objective function values for the discovered designs are listed along with the reference design provided by Neuffer [143] in Table 4.4. The range of the values of the objective function that was obtained during the optimization runs is 15 to 497. From the table it can be seen that the optimization of the current scheme with control parameters in the specified ranges was unable to achieve designs that have statistically (simulation includes stochastic processes) significantly better production efficiency. Although this cannot serve as a rigorous proof of the nonexistence of such designs, we can take into account generally good performance of the GATool on other problems (see section 2.3.4) and suggest that this gives a good reason to believe that the reference design is, indeed, optimal. Relatively small deviations of the optimal RF frequency (201.20–201.55) and RF gradient (17.67-18.88) among all the solutions (except for the first optimization run) support the assumption about the reference parameters being robust and located near the global optimum. This observation is particularly important, since the parameters of the devices that are calculated by numerical simulations eventually have to be realized in the physical devices operating with finite precision and subject to construction errors.

The best solution obtained from the first optimization run, demonstrated one of the best performances and, along with it, the smallest RF gradient among other solutions. Smaller-gradient RFs are generally easier/cheaper to manufacture, which can significantly reduce in the potential cost of the lattice. However, the final frequency from this solution is different from the frequency of the current baseline accelerating section. Hence additional studies on the combined performance are needed to reveal additional benefits and drawbacks of this solution. The best solution from the third optimization run provides similar performance on a frequency that is much closer to the reference 201.25 and thus might be preferable. Some of the other sets of parameters that provide similar production performances can also be useful since they might be easier or cheaper to obtain, or provide additional opportunities for the designers of the downstream sections of the Neutrino Factory. Hence the main benefits of the GATool for this problem are in the exploration of the space of the solutions that provide some interesting candidate solutions. It would otherwise only be possible through extensive trial and error, similar to the method used to obtain the reference solution over the years of the research. Also we verified the quality of this solution obtained through such a laborious process.

Yet another important result that was obtained is that we established a framework for the Neutrino Factory Front End lattice numerical optimization. It can be used for many optimization scenarios, including, for example, a simultaneous optimization of all control parameters of the most realistic Front End simulation on the large ranges of the parameter values.

Parameters	$\nu_{\mathrm{rf,cool}}$ [MHz]	$V_{\mathrm{rf,cool}}$ [MV/m]	$\varphi_{\mathrm{rf,cool}}$ [degrees]	$p_{\mathrm{c,match\_cool}}$ [GeV/c]	n_2 ($n_2/2000$) particles	n_2 ($n_2/8000$) particles
reference parameters [143]	201.25	18.00	30.000	0.220	498 (0.249)	1740 (0.218)
3rd opt. run, 6th best	201.46	17.77	11.320	0.229	480 (0.240)	1791 (0.224)
3rd opt. run, best	201.40	17.06	12.648	0.226	492 (0.246)	1782 (0.223)
1st opt. run, best	200.55	17.10	26.970	0.220	467 (0.234)	1780 (0.222)
3rd opt. run, 5th best	201.28	17.76	12.457	0.226	484 (0.242)	1773 (0.222)
3rd opt. run, 3rd best	201.47	17.67	13.470	0.228	485 (0.243)	1762 (0.220)
3rd opt. run, 2nd best	201.42	17.68	12.555	0.226	486 (0.243)	1750 (0.219)
3rd opt. run, 7th best	201.34	17.68	12.020	0.226	479 (0.240)	1746 (0.218)
2nd opt. run, 2nd best	201.24	18.91	20.520	0.228	471 (0.236)	1714 (0.214)
3rd opt. run, 4th best	201.48	17.75	11.860	0.227	485 (0.243)	1669 (0.209)
2nd opt. run, best	201.20	18.88	22.477	0.230	497 (0.249)	1643 (0.205)

Table 4.4: *Results of the Front End design optimization (in ascending order sorted on the production rate for the 8000 particles initial beam)*

An additional note must be made on performance. Even though 2000 particles were used for simulations during optimization instead of the 8000 used in the baseline simulations, one simulation run of the optimized front end lattice still took approximately 0.4 hours on a Pentium IV 2Mhz computer with 1Gb of memory. Therefore the calculations needed to perform one step of GATool optimization (which include the evaluation of the objective function for every population member, see section 2.3) took approximately 100 hours. Since a typical number of generations needed for GATool to explore a space of parameters and converge can easily get above 100, the computing time can get prohibitively long. For subsequent studies (possibly on more realistic and thus more computationally expensive lattices), a potentially beneficial strategy is to employ the parallelization of the objective function evaluation as discussed in section 2.3.4.

4.4 Conclusions

In this section we considered an application of the GATool evolutionary optimizer presented in this work to set of the cases covering a broad spectrum of the problems from the accelerator design field:

- an example of the quadrupole stigmatic imaging triplet design that is relatively simple on the surface yet demonstrates certain complexities under closer investigation (this problem is also directly connected to the real-life complicated problem of the collider interaction region design);
- an estimation of particle dynamics stability via a normal form defect function;
- optimization of the control parameters of the front end for the next generation accelerator. All of these optimization problems are formulated in such a way

that their objective functions are either very hard to treat for most standard optimizers or are even non-treatable due to their small domains of attraction for the extrema, highly oscillatory landscape that contains a large number of local extrema, or, as in the third problem, general unavailability of the function in the algebraic form (it is computed through numerical simulations) and its stochasticity (some devices in accelerators are simulated with stochastic effects).

We demonstrated how GATool is capable of solving these problems (or helping to solve by dramatically reducing the computational time as is the case for the normal form defect function optimization). It proved itself to possess attractive features such as: moderate requirements on the computational resources, no requirements on the objective function except for the trivial ability to compute its value for a given set of control parameters, and a surprisingly high (considering such modern requirements) quality of the obtained result.

GATool also demonstrated a very useful ability to find furtive or unpredicted solutions to accelerator design problems, thus enabling easy exploration of the space of the different optimal solutions. Exploration phase usually require a domain expert, a lot of the time spent in trial and error, with fair amount of intuition and even blind luck. With the help of GATool both the initial exploration of the design and the final fine tuning phases become much more efficient and/or rich since the number of the solutions that can be considered and their quality dramatically increase with almost no additional human effort. Summing up all the evidences we conclude that GATool demonstrated itself to be a valuable addition to a tool set of a modern accelerator scientist.

APPENDICES

APPENDIX A

COSY++ Macroprogramming Extension for COSY Infinity

A.1 COSYScript

A.1.1 Introduction

COSY Infinity [22, 23, 48] is a powerful software package for scientific computations. It was originally created by Dr. Martin Berz [14], who is currently maintaining and further developing the package with Dr. Kyoko Makino. Contributions, additions and enhancements, accumulated for over a decade, experience and feedback obtained from different users working on different scientific problems, along with careful design and the implementation have made this powerful tool even better.

It is built around the Fortran77 kernel which implements the Differential Algebra arithmetic [16]. Other packages implement graphical interface, optimization methods, and even own scripting language interpreter. This language is called *COSYScript* and has simple yet rich syntax closely resembling Pascal. Despite a relatively small number of built-in operators, this language gives a demanding user full access to not only real numbers, vectors (with optimizations for vectorizing supercomputers), complex numbers, logical type variables, and strings but also to such complex data types as Interval Numbers, Differential Algebra Vectors, Taylor Models [118, 121] and their complex arithmetic via a transparent set of operators and functions. The concept of polymorphism from Object-Oriented programming is carefully used in design to allow a user to easily switch between different data types and mix them in calculations, thus giving this package the ability to easily manage the complexity of computations, type and precision of the obtained results.

Another interesting feature of the language is an optimization built into the standard syntax. After each step of the optimization process with one of three built-in optimizers (each one has its own strengths and weaknesses) a user is given the current best value and execution control, and is from there free to make decisions about the subsequent execution flow. This feature allows one to build complex optimization scenarios combining automatic optimization by built-in optimizers with the user input. It is also worth noting the *COSY Infinity* is a multi-platform system and is capable of producing a graphical output on every platform it is supported on (including Windows, Linux and MaxOS), interfaces modern programming languages such as C++ and Fortan90 and, finally, it is easily extensible. The code base is still under active development, new features are being developed and added, and, as such, are available to users upon request. The general policy is to include them in a standard distribution available to the entire user community only after extensive testing and fine-tuning. This policy ensures that system remains consistent and robust. Current features under development includes the language-level parallelism and high-precision numbers arithmetic.

Such a framework provides a user with a versatile set of tools capable of elegantly solving a lot of otherwise computationally hard (or even unsolved) problems. *COSYScript* applications include, but are not limited to:

– high-order Automatic Differentiation of the functions [4],

– the verified and non-verified integration of the Ordinary Differential Equations and Differential Algebraic Equations [168],

– rigorous and verified numerical methods with highly suppressed dependencies [168],

– rigorous global optimization [118],

– Beam Theory, where it is applied to a variety of problem, e.g. the analysis of the high-order effects, high-order fringe fields treatment in accelerators and spectrographs, rigorous long-term stability studies [15,19,21,24,25,76,96,113–116,119,120,123,124,126,154,177].

A.2 Syntax

From a programmer's point of view *COSY Infinity* consists of the three main parts:

1. An elementary operations package written in Fortran 77 which implements operations on various COSY data types such as Differential Algebra, Taylor Models and interval arithmetic: **dafox.f**.

2. An optimization package (**foxfit.f**), a graphics package (**foxgraf.f**) and a compiler and executor package (**foxy.f**) which combines all these to implement the *COSYScript* language and *COSY Infinity* front end. All these are written in Fortran 77, same as the kernel.

3. Packages written in *COSYScript*: the Beam Theory package (**cosy.fox**), the Rigorous Computing package (**TM.fox**), the Rigorous Global Optimizer and Rigorous Global Integrator (currently distributed separately).

Although it is such an extensive and developed, environment for scientific computations, *COSY Infinity* still has some aspects that could benefit from improvement. At the time *COSY Infinity* was initially designed and developed it was not possible to predict how successful it would be. Thus, the built-in mechanism to allow *COSYScript* code to be modular was relatively simple and rudimentary (by the code modularity we hereby mean the ability to store the source code in more or less self-contained modules). When the code is modular, most of the services are provided by well-defined interfaces to these modules with the implementation details hidden from a user.

A common program in such a framework simply includes the required modules to import their services and call the imported procedures and functions. The amount of the code in the modules is typically much larger than the code in the user program. If a user decides to build a larger program or a set of programs, he might want to implement some functionality in his own modules. This approach allows code to be clean and well-structured and permits easy reuse of the already written code. In such cases the amount of the user code can be comparable to one of the modules shipped with the system itself.

In general, *COSYScript* program is a set of nested blocks and each of them consists of three sections. The blocks are marked by the beginning and ending statements for the outermost block (the main program block):

```
BEGIN;
...
END;
```

and for all the inner blocks:

```
FUNCTION <name> {<args>};
...
ENDFUNCTION;
```

or

```
PROCEDURE <name> {<args>};
...
ENDPROCEDURE;
```

The three sections that build up a block are

1. Variables
2. Nested blocks (functions and procedures)
3. Executable code

placed in this exact order. The first two of these sections are optional and can be omitted, while the absence of the executable code results in a compilation error.

The structure of a generic *COSYScript* program along with some hints on the name scoping rules (for variables, procedures and functions) are demonstrated by the example in Figure A.1. Note that *COSYScript* is a case-insensitive language, hence the variables, functions, and procedures that are defined in the same scope whose names differ only in case refer to the same variable, function, or procedure, correspondingly.

A.2.1 Problems

Inclusion Mechanism

While the code's nested structure makes it tree-like, the original inclusion mechanism supported by *COSY Infinity* is linear. It is implemented via a pair of commands:

```
BEGIN;
    VARIABLE main_var1 1;
    ...
    VARIABLE main_var8 10;
    ...
    PROCEDURE Proc1 arg1 arg2;
        VARIABLE proc1_var1 5 2;
        PROCEDURE Proc1_Proc1 arg1;
            VARIABLE proc1_proc1_var1 11;
            ...
            { Commentary: code for Proc1_Proc1 }
            proc1_proc1_var1 := 'Hello';
            proc1_var1 := 'world!';
            ...
        ENDPROCEDURE;
        { Commentary: code for Proc1 }
        proc1_var1 := 'Goodbye!';
        ...
    ENDPROCEDURE;
    ...
    FUNCTION Func1 arg1 arg2 arg3;
        { Commentary: code for Func1 }
        Func1 := (arg1 + arg2 + arg3) * main_var1;
    ENDFUNCTION;

    { Commentary: code for main block }
    write 6 'Hello, world!';
    ...
END;
```

Figure A.1: COSYScript *program structure*

```
SAVE <name>
```

and

```
INCLUDE <name>
```

The first of these commands is used at the end of the *COSYScript* file that is included. It precompiles the source code from the **<name>.fox** file and then saves it in the binary form to the file file named **<name>.bin**. Then the

```
INCLUDE <name>
```

command used in some other file includes the compiled **<name>.bin** into it. However, each file can contain only one inclusion command (at the beginning), thereby only linear "chain inclusion" is supported. The first file calls

```
SAVE <name>
```

in its last line and every next file starts from

```
INCLUDE <name_previous>
```

and ends with

```
SAVE <name_current>
```

in order to incrementally save both the code it includes from the previous files and the code it contains. The last file starts with

```
INCLUDE <name_previous>
```

and includes all the code that was gathered by the **name_previous.fox** (and thus the code of all the files from the inclusion chain).

Note that while the code saved by **SAVE** is precompiled (which saves the source processing time), the action of the **INCLUDE** is essentially equivalent to copying the content of the file that was saved with the **SAVE** statement (not including the statement itself) and then replacing corresponding **INCLUDE** statement with this copied code.

The problem with this approach is in the fact that the sections in the blocks of any *COSYScript* program have to be exactly in the order mentioned previously and cannot be broken into parts. Suppose the first file in the inclusion chain opens the main block with **`BEGIN`**, then specifies variables, functions, and procedures, and then precompiles and saves its contents as described.

The next file includes it in-place, thereby starting where the previous file left off, i.e. in the middle of the section with functions and procedures. In this case some of them are already defined in the included files. Thus the including file cannot add variables to the main block, rather it can only continue adding functions and procedures to the Nested Blocks section, and then some code to the Executable Code section. If the first file is saved only after some code is added to the Executable Code section, then the next file in inclusion chain can only add code to the Executable Code section of the main block; it has no access to Variables or Nested Blocks sections.

The frequently used workaround for such an approach is to define the procedure called `RUN` and then use it as shown in Figure A.2. Note that we have to put **`END`** at the end of this code even though we did not explicitly put a corresponding **`BEGIN;`** to this file. It is hidden behind the **`INCLUDE`** statement in the file we are including.

Names defined in the enclosing block are visible and accessible in the enclosed block. They can even be overridden by the names local to the enclosed block. However, the opposite is not true: the variables defined in the enclosed block are not visible to the enclosing block and thus cannot be used by it. This is perfectly legit from the encapsulation point of view, since the external block should be isolated from the intrinsic implementation details of the internal block. This concept is successfully applied to enhance modularity in many existing programming languages.

```
INCLUDE 'file'
    PROCEDURE RUN;
        { Private variables }
        VARIABLE i 1;
        ...
        { Private nested blocks }
        FUNCTION max a b;
            ...
        ENDFUNCTION;
        ...
        { Private code }
        i := max(1,2);
        ...
    ENDPROCEDURE;
    { Main block code, just call the package code }
    RUN;
END;
```

Figure A.2: *Example of inclusion workaround*

Suppose, however, we want to save the code in the file from Figure A.2 and then include the result into another file. We have several options: we can save the code inside the `RUN` procedure, outside of it, in the main block before the first line of code, in the nested blocks section, or in the code section of the main block. Let us consider each of these options. If we save the file anywhere outside of the `RUN` procedure, and then include it in the next file in chain, the next file can only call the `RUN` procedure. It will have no access to the variables, procedures or functions defined in the `RUN` procedure. Hence either RUN should serve as a call dispatcher, providing the access to the names defined inside it (which is a cumbersome and hardly maintainable solution), or we have to call **`SAVE`** inside the `RUN` procedure. But then with the next file in chain we will be in the same situation: it would have to define its own `RUN` enclosed in the `RUN` of the previous file in chain. As the chain grows, the *resulting file*

```
file1 code
file2 code
...
...
...
...
...
...
...
...
file11 code
```

file1

```
file1 code
SAVE file1;
```

file2

```
INCLUDE file1;
file2 code
SAVE file2;
```

...

file11

```
INCLUDE file10;
file11 code
```

Figure A.3: *Example of the inclusion chain*

level of nesting increases, and keeping track of the names visibility and nesting level can become an issue.

Consider the case where a user did not build the code of the whole chain but receives only a file in precompiled binary form. He than has no way to determine the number of files that was used to compile the received file and the combination of methods they used to cope with the nesting problem in inclusion. It makes the derivation of the proper syntax for ending statements a matter of trial and error and clutters user's program with unneeded code.

We can see that such an approach to inclusion is suitable for small projects. However, for large projects, especially the ones developed by by several authors, it does not provide enough support to maintain modularity of the program and manage the dependencies transparently. All these consequences follow from each file in the inclusion chain not being a syntactically separate compilation unit and is bound to be made aware of its place in the chain of inclusion (see Figure A.3).

Size of Dynamic Global Variables

COSYScript is an interpreted language, and it each program is interpreted from beginning to end, following a nested blocks structure. In each block interpreter starts from the variables section, processing all the variables declarations and storing the corresponding entries in the symbols table. Then, it checks the declarations of the procedures and functions in the Nested Blocks section without looking into their code. It then proceeds to the Executable Code. When the procedure or function call is found, the interpreter searches for the name of the procedure or function in the symbol table, initializes the arguments (provided the name was found) and recursively continues interpreting the nested block defined by a procedure or function, thereby following the tree-like structure of the program.

COSYScript uses the following syntax for the variable declaration:

```
VARIABLE <name> <size> { <dim1> {<dim2>...}};
```

Here `name` is a variable name, `size` is a variable size (where unit is the size of `REAL` data type), and dimensions are used to make multi-dimensional arrays of the objects. Note that the type of the variable is not declared and is deduced dynamically at run time. In this definition, size and dimensions can be valid *COSYScript* expressions. The validity of the expression includes both the syntactic validity (i.e. it should form a syntactically correct *COSYScript* expression) and interpretation validity (i.e. should be interpretable: all the names used in the expression must have their declarations already processed). It is thus possible to have the size of a variable or its dimensions to be set by nother variables, but the variable containing size must be declared earlier.

Consider an example in Figure A.4 where we want the variable `dynamic_var` to be declared such that its size is defined by another variable. Here we want to set the size of the `dynamic_var` to `10` but this is not what would happen. The declaration of the

```
...
VARIABLE size 1;
VARIABLE dynamic_var size;
...
size := 10;
...
```

Figure A.4: *Dynamic size of the variable in* COSYScript *(non-working)*

`dynamic_var` is processed after the declaration of the `size` variable but before the size variable gets the value of `10`. The syntax of the variable declaration in *COSYScript* does not allow to define the initial variable value. All variables in *COSYScript* are initially initialized to `RE(0)` which is `0` of the `REAL` data type. Hence the `dynamic_var` in the example is not declared as we intended. It is not even a valid declaration of a *COSYScript* variable since it really declares a variable of zero size and in *COSYScript* variable size must be a positive integer.

To solve this problem it is currently suggested to use a technique similar to the one to avoid the inclusion problems [23]. Some procedure or function encloses the variable whose size is to be defined by another variable, thus placing the size variable in the outer block. Going back to the example in the Figure A.4, we modify it using the current guidelines to get a proper dynamically sized variable definition. Such modification ensures that both a declaration and an initialization of the variable `size` are done before the declaration of the `dynamic_var` thus we achieve the result desired.

However, there are two problems with the proposed solution. First, it clutters a program with the unnecessary code, second, it cannot be used for the variables declared in the main block since it serves as a root of the tree of all blocks. Hence it is therefore not enclosed by any other block where the size variable can be declared and initialized.

```
...
VARIABLE size 1;
...
PROCEDURE RUN;
    ...
    VARIABLE dynamic_var size;
    ...
ENDPROCEDURE;
...
size := 10;
...
RUN;
...
```

Figure A.5: *Dynamic size of the variable in* COSYScript *(working)*

User Interface

Apart from the problems already mentioned, the execution options for the *COSYScript* interpreter are very limited. In fact, it does not accept any options directly from user. During startup, it looks for the file **foxyinp.dat** in the current directory, reads its first string and then displays the *COSY Infinity* logo and the user prompt. In the prompt it allows the user to either enter a new *COSYScript* filename for execution or press «Enter» to execute the file read from **foxyinp.dat**. If the filename is correct, it interprets the file and displays either the list of interpretation errors if interpretation fails or the output of the program if it succeeds. The text output is shown in *COSY Infinity* window, optional (depending on the driver selected) windows containing the graphical output can be opened separately.

After executing the script, the file interpreter exits, leaving two intermediate processing files with the names **<name>.cod** and **<name>.lis** (useful for debugging) and the files produced by the script. The user cannot specify the name of the script to execute from the command line, cannot ask to cleanup the intermediate files, specify the path to search for the files to include or pass command line arguments to the *COSYScript* script. The command to call external programs through the OS shell was added to *COSYScript* just recently so the only way to interface *COSY Infinity* was through the files containing input parameters and output results.

Much of these limitations for such a powerful language and framework are attributes to the fact that it was designed and developed by a very small group of people in a very limited amount of time with primary focus on scientific methods and new algorithms. User interface was never a top priority. Enhancing *COSY Infinity* user interface and providing tight integration with another well-developed and mature programming language to use some of its features seemed as a good approach to make the scientific computing features of *COSY Infinity* more attractive to users.

A.3 COSY++

A.3.1 Introduction and Features

To address the problems described in the previous section and enrich the user's experience with *COSY Infinity* we designed and implemented the *COSY++* extension package. Its main features include, but are not limited to:

– new mechanism for *COSYScript* source files inclusion that provides better separation of modules from user code;

– Active Blocks mechanism that enables the use ofthe Perl programming language ([153]) as a macrolanguage for *COSYScript* programs. Applications: conditional *COSYScript* code generation, command-line arguments processing, macroprogramming, data preprocessing, and *COSYScript* libraries configuration;

– new libraries for vector manipulations, coordinate conversions to/from MARS [137] and ICOOL [68], logging, timing scripts execution and debug output;

– GATool library for real-valued functions optimization with callback interface (allows a user to interact with the optimizer on each step, fits into the general *COSY Infinity* optimization package design (see Appendix B for details);

– front end for the *COSYScript* interpreter that allows a user to specify the script (and even several scripts) to interpret and execute from the command line, pass command line arguments to the scripts, use different *COSY Infinity* executables (e.g. complied with different compiler optimizations to benchmark the speed and precision), perform cleanup after an execution, and save script output to a file and specify a search path for the library files (this allows user to store common libraries in one central location and use them from any script in any directory);

– automatic conversion of old *COSYScript* scripts, i.e. compatibility mode.

We will now describe these features and implementation details in greater depth.

A.3.2 Sections Assembler

Sections Assembler addresses inclusion mechanism problems discussed in section A.2. Instead of linear inclusion it uses the concept of assembly. In order to produce the resulting file from the user's code and the code of the modules (libraries) it assembles them together using the structure of the *COSYScript* program, its blocks ordering, and special markup commands inside *COSYScript* comments (this makes them transparent to the conventional *COSYScript* interpreter).

As we discussed in section A.2), a *COSYScript* program is constructed from blocks. Each of these is built from the sections (some can be omitted). Consider the root block that consists from HEADER section

```
BEGIN;
```

VARIABLES section

```
    VARIABLE main_var1 1;
    ...
    VARIABLE main_var8 10;
    ...
```

FUNCTIONS section

```
    PROCEDURE Proc1 arg1 arg2;
        VARIABLE proc1_var1 5 2;

        PROCEDURE Proc1_Proc1 arg1;
            VARIABLE proc1_proc1_var1 11;
            ...
            { Commentary: code for Proc1_Proc1 }
            proc1_proc1_var1 := 'Hello';
            proc1_var1 := 'world!';
            ...
        ENDPROCEDURE;
        { Commentary: code for Proc1 }
        proc1_var1 := 'Goodbye!';
        ...
    ENDPROCEDURE;
    ...
    FUNCTION Func1 arg1 arg2 arg3;
        { Commentary: code for Func1 }
        Func1 := (arg1 + arg2 + arg3) * main_var1;
    ENDFUNCTION;
```

CODE section

```
    { Commentary: code for main block }
    write 6 'Hello, world!';
    ...
```

FOOTER section

```
END;
```

The goal here is to first mark up sections of the root blocks of the *COSYScript* files. Then, if a *COSYScript* file requests to add marked up file to an assembly, the result is created by merging the corresponding sections of the included file with the ones of the including file, instead of performing in-place insertion.

Intrinsically, the assembly is represented by a list of sections. All the contents of these sections are empty on initialization. The included file is parsed into the sections, and these sections are added to the assembly, and then the same procedure is repeated for the including file. Thus the including file's sections are effectively appended to the corresponding sections of the included file. If there are more files to assemble, the process becomes multi-step. The assembler goes through the list of the files to assemble, parsing them into sections and adding sections to the assembly. When the parsing process is completed, sections are output to the assembled file in the specified order. Thus they form correctly structured *COSYScript* code which can then be interpreted by *COSYScript*.

As an example, consider the two files with their sections marked up in Figure A.6. The result that comes out of the Sections Assembler after processing and merging is in Figure A.7. Note that the function `FUNC1` and the variable `FOO1` used in including file were not declared in it, hence any attempt to use this file without assembly would result in *COSYScript* interpretation errors. Also note, that after assembly these names are correctly ordered, thus the compilation and the execution of the resulting file is possible. The order of the `'Hello, world-1,2'` statements in the assembled file is meant to give hints on the assembly order.

The general rule is that the content of included files in assembled sections precede the contents of corresponding sections in including files. Section mark up statements

```
{ #section <name> }
...
{ #section <name> }
```

are preserved in the form of sections beginning markers in order to increase the readability of the resulting file. These statements are transparent for *COSYScript* since they look like valid comments to its parser.

The instruction to add a file to an assembly has the form

```
{ #assemble 'file_name' }
```

and can be put anywhere in the file. Assembly instructions are processed before running *COSYScript* interpreter hence they are not ruled by the *COSYScript* syntax rules. The name of a file specified in the instruction is searched in the current directory and in the assembly path, which can be specified via the *COSY++* front-end. Standard suffixes (by default **.fh** and **.fox**) can be omitted and will be automatically appended during library search. See section A.3.8 on configuring the *COSY++* front end for details on the search path and standard suffixes.

The file added to assembly can itself contain assembly instructions. In this case processing continues recursively, forming the assembly tree, which is traversed from the leaves to the root in the depth-first manner. For example, if **file1** instructs the assembler to assemble **file2**, which in turn requests to assemble **file3**, then every section's contents will be built from the contents of the corresponding sections: first from **file3**, then from **file2** and only then from **file1**. Such ordering rules allow **file2** and **file3** to be libraries, and for the **file2** to use **file3** (possibly to implement parts

Including file

```
{ #assemble 'lib_file' }

{ #section HEADER }
BEGIN;
{ #endsection }

{ #section VARIABLES }
VARIABLE FOO2 1;
{ #endsection }

{ #section FUNCTIONS }
PROCEDURE PROC2;
    ...
    RES := FUNC1(FOO1, FOO2);
    ...
ENDPROCEDURE;
{ #endsection }

{ #section CODE }
WRITE 6 'Hello, world-2!';
{ #endsection }

{ #section FOOTER }
END;
{ #endsection }
```

Included file (lib_file)

```
{ #section VARIABLES }
VARIABLE FOO1 1;
{ #endsection }

{ #section FUNCTIONS }
FUNCTION FUNC1;
    ...
ENDFUNCTION;
{ #endsection }

{ #section CODE }
WRITE 6 'Hello, world-1!';
{ #endsection }
```

Figure A.6: *Including and included files with their sections marked up*

```
{ #section HEADER }
BEGIN;

{ #section VARIABLES }
VARIABLE FOO1 1;
VARIABLE FOO2 1;

{ #section FUNCTIONS }
FUNCTION FUNC1;
    ...
ENDFUNCTION;

PROCEDURE PROC2;
    ...
    RES := FUNC1(FOO1, FOO2);
    ...
ENDPROCEDURE;

{ #section CODE }
WRITE 6 'Hello, world-1!';
WRITE 6 'Hello, world-2!';

{ #section FOOTER }
END;
```

Figure A.7: *Assembled file*

of its own functionality). In this example **file3** can be a user application using the features from the library in **file2** (and thus implicitly from **file3**).

The Sections Assembler also has a mechanism to prevent the multiple file inclusion into an assembly in order to avoid duplicating code across sections. If the file that is already in the assembly tree is encountered again during assembly processing, the warning is issued and the file is not processed for the second time around. Note that this mechanism also prevents the assembly tree from the loops (**file3** includes **file2**, which, in turn includes **file3**) that can lead to an infinite cycle in the assembly process.

The algorithm is straightforward: whenever a filename is encountered in the assembly process, a unique name based on the file's base name, size, and creation time is created. The name generation process is guaranteed to create the same unique name for the same file (in the same location with the same attributes) whether it is specified by an absolute path or a relative path. This generated name is then looked up in the list of the unique filenames for the files that are already in assembly. If this name is found, the file is considered to be already assembled, otherwise its unique name is stored in the table and the file is added to the assembly.

Apart from the HEADER, VARIABLES, FUNCTIONS, CODE and FOOTER sections described earlier, *COSY++* recognizes a DESCRIPTION section that precedes all other sections in the assembly. It is intended to serve as a placeholder for the library descriptions but it can also contain assembly instructions and the Active Blocks initialization code. An example of the description section taken from the **logging.fh** library is shown in Figure A.8.

```
{ #section DESCRIPTION }
{****************************************************************************}
{ Copyright (C) Michigan State University 2007, All Rights Reserved,        }
{ \COSYS{} logging routines library by Alexey Poklonskiy                    }
{****************************************************************************}
{ Control variables and their defaults                                      }
{  LogLevel = 1 (0 -- logging is off, 1 -- logging is on)                   }
{  nMaxLogFiles = 10                                                        }
{  iFirstLogFileDescriptor = 50                                             }
{  LogFilesDir = './log/'                                                   }
{****************************************************************************}
{ #endsection }
```

Figure A.8: *Description section of the* ***logging.fh*** *library*

A.3.3 Active Blocks

The idea of Active Blocks is inspired by the Active Server Pages technology developed by Microsoft [85]. It allows one to build interactive HyperText Markup Language (HTML) pages by adding the ability to embed VBScript or any other Active Scripting Engine language code into the otherwise static web page source code.

The embedded code in the HTML text is recognized by special beginning and ending markers, `<%` and `%>`. Everything in between these markers is treated as a code written in one of the scripting languages mentioned earlier and is executed during the HTML page rendering phase. Since VBScript is a feature-rich scripting language with an extensive object model providing access to many OS services, it gives user a plethora of tools to make HTML pages dynamic. A detailed description of the technology is out of scope for this work and can be found in [85].

Similar design lies in the foundation of *COSY++* Active Blocks (ABs). Everything enclosed by `{%` and `%}` markers in the *COSYScript* source code is treated as Perl programming language [173] source code and is executed during file processing.

There are two types of the Active Blocks:

non-inclusive

```
{%
...
%}
```

and

inclusive

```
{%=
...
%}
```

The difference between them is that apart from the effects produced by the execution of the Perl code inside the non-inclusive Active Block, it does not affect the *COSYScript* source code in any way. A return value of the inclusive Active Block is inserted back into the file and is thus interpreted as a part of the *COSYScript* program. This makes inclusive Active Blocks useful for conditional *COSYScript* code generation. As an example, consider the code of the tracing procedure from the **trace.fh** library:

```
procedure Trace sMsg;
{%=
    my $result = "";
    if($TraceLevel){
        $result = "   write 6 sMsg;";
    }
    else{
        $result = "   CONTINUE;";
    }
    return $result;
%}
endprocedure;
```

If `$TraceLevel` is set to any nonzero value in one of the earlier Active Blocks, this inclusive Active Block definition inside the *COSYScript* procedure declaration is replaced by:

```
procedure Trace sMsg;
    write 6 sMsg;
endprocedure;
```

But if it is not set or set to a zero, this Active Block is replaced by:

```
procedure Trace sMsg;
    CONTINUE;
endprocedure;
```

One more example of Active Block usage comes from the logging library **logging.fh**:

```
{%
    # Create the directory for log files if it was not created and the
    # logging is on
    if($LogLevel and not -d $LogFilesDir){
        mkdir($LogFilesDir) or
            AB::die("Can't create directory \"$LogFilesDir\" ".
                    "for log files: $!");
    }
%}
```

Here AB is used during library initialization. It checks if the directory to store the log files exists and creates it if it does not. In the code from **utils.fh**, ActiveBlock is used to access Perl's random number generator:

```
{ Get the inintial random value by using Perl's rand() which     }
{ calls srand() to get random seed automatically                 }
time := {%= rand(); %}*1000;
```

COSYScript does not provide an interface to directory creation and its pseudorandom number generation capabilities are limited by the fact that it always starts from the same seed. Therefore these examples clearly demonstrate how *COSYScript* can be enhanced by providing access to Perl's services via Active Blocks. Other examples can be found in the **examples** directory of the *COSY++* distribution. It is also worth noting that all Active Blocks in one *COSYScript* file share the same Perl environment thus they can communicate with each other via shared variables.

In the previous paragraph we briefly considered the aspects of Active Block execution. Now we will examine them in more detail. After an Active Block is found by *COSY++*, its type (inclusive, non-inclusive) is determined, beginning and ending marks are stripped, its contents are extracted, and then it is executed as Perl code. Perl, as well as many other scripting languages, provides dynamic compilation and execution of its own code during the main program execution. This feature allows a user to generate subroutines in the run-time or execute code passed to the script as text. The compilation and execution of this code is performed in Perl via the **`eval`** operator. It accepts a string with Perl code as an argument, compiles it, and then executes it in the context of the script. AB execution order is the order in which they are specified in the source file.

The problem with this approach is that in this case the code passed to the **`eval`** operator has access to all variables of the program executing the **`eval`** within the scope of the **`eval`** statement. This code can accidentally or intentionally modify these variables thereby altering the calling program execution flow, and in the worst case leading to a data corruption or a crash. *COSY++* itself is written in Perl, it executes ABs written in Perl using the mentioned operator, and thus it is potentially susceptible to this problem.

The solution we implemented is to use special Perl module `Safe`, which provides so called "sandboxes" or compartments for a safe code evaluation. A code executed in such a compartment is unaware of this fact, but it is effectively unable to access any data belonging to the program that initiated the code execution (and in some cases even to some of the Perl services), unless such permissions are explicitly granted. By applying this technique we ensure that all the sensitive *COSY++* data is protected from the code in Active Blocks and all its services are provided only through a well-defined interface, which we describe later.

There are two different symbol scoping mechanisms in Perl: dynamic and lexical (for further details on Perl programming refer to [173]). Dynamically scoped variables are accessible globally. They always belong to some package and can be accessed via fully a qualified name (e.g. `$Package::Variable`). Alternatively, a short name (e.g. `$Variable`) can be used instead if the variable belongs to a current package (maintained by Perl) (`$CurPackage::Variable`). The current package can be set via the **`package`** `<name>` operator, the default package name is `main`. A current package declaration stays in effect from the place it is made in the current program to the end of the block it is specified in.

The visibility of lexically scoped variables is defined by the blocks they are declared in. Their scoping rules are the same as the one for *COSYScript* variables, i.e they are accessible after their declaration in the block they are defined in and in all the blocks enclosed by the block they are defined in. The difference between Perl's lexically scoped variables and *COSYScript* variables is that Perl's variables can be declared anywhere in the block. This is possible because Perl does not separate blocks into sections while *COSYScript* does and, moreover, allows variable declarations only in the first of the sections of the block. Lexically scoped variables do not belong to any package and are not influenced by **`package`** statements.

The reason for such an extensive treatment of this seemingly irrelevant topic is that individual Active Blocks are internally executed as if they were different blocks of the same Perl program, hence all of the hereby mentioned scoping rules apply to them. It is possible for Active Blocks to communicate and control each other via

COSYScript file

```
{ #package PackageName }
COSYScript code
{%
    Active Block 1 code
%}
\COSYS{} code
{%=
    Active Block 2 code
%}
\COSYS{} code
{%
    Active Block 3 code
%}
...
```

Corresponding Perl file

```
{
    package PackageName;
    Active Block 1 code
}
{
    package PackageName;
    Active Block 2 code
}
{
    package PackageName;
    Active Block 3 code
}
...
```

Figure A.9: *COSYScript file as a Perl file view from Active Blocks perspecitve*

this program's execution environment, defined by variables and functions, and these scoping rules must be taken into account.

Active Blocks execution order (affects the scoping and the availability of the variables declared in ABs) is the order in which they are declared in the source file. *COSYScript* file containing Active Blocks can be viewed as a Perl script, as demonstrated in Figure A.9. Note that since Active Block 2 is of the inclusive type, its return value (defined by the Perl **`return`** statement or by the result of the evaluation of the last statement in a block, if the **`return`** statement is not specified) is inserted into *COSYScript* program after evaluation.

Using the view from Figure A.9 and the mentioned scoping rules, one can see that in the Active Block body, dynamic variables used without package name are binded to the package PackageName. For example, the unqualified dynamic variable `$Variable` in any of the Active Blocks 1, 2, 3 in the above example is internally the qualified dynamic variable `$PackageName::Variable`. Dynamic variables accessed via a fully qualified name with a package name explicitly given are bound to this package, e.g. `$AnotherPackageName::Variable`. Dynamic variables are accessible from anywhere in the program across the AB boundaries and, as such, are useful to pass information between ABs of the *COSYScript* program.

Lexical variables are declared via the **`my`** Perl operator. They are visible only in the lexical block they are declared in, thus they cannot be accessed outside the AB itself. If AB consists of several Perl blocks, then each block's lexical variables are visible in the corresponding block only.

`PackageName` in the example earlier is a package name for the current file. It is either a unique name that is automatically generated from the filename or a name set by the user via the `package` pragma:

```
{ #package package_name }
```

In the case when there are several package name declarations, the name from the first pragma is used.

In both cases `PackageName` is set once per file. Technically a user can change it from ABs using Perl, but such practice is not recommended in order to avoid possible problems. We recommend to naming packages in order to offer some clue of the services provided by the package. For example, the **timers.fh** library uses the `Timers` package name (thus the variables are syntactically in the `COSY::Timers::` package), and **logging.fh** uses the `Logging` package name (thus the variables are syntactically in the `COSY::Logging::` package). Note, however, that the **cosy.fh** library uses the `BeamTheory` package name, as it better describes the library's services.

A user's code can omit a package name declaration and rely on automatic package name generation unless his intent is to build a library to share it with other people. In this case it is strongly advised to invest some time in chosing a descriptive package name in order to make the library configuration variables accessible via the `COSY::<PackageName>::` prefix. The unique package names generated for the file in the absence of an explicit definition by the `package` pragma are not guaranteed to not change in the next *COSY++* run.

Also note, that Perl itself internally maintains the `__PACKAGE__` variable to store the current package's name. It is often used by *COSY++* libraries to mark their variable and function sections with

```
{%=     "{*** ".__PACKAGE__." functions ***}\n"     %}
```

that is replaced by the *COSYScript* commentary

```
{*** package_name functions ***}
```

during *COSY++* processing.

From experience we suggest the following scheme of Perl variable usage in Active Blocks: to create the variables used in the current Active Block only, use lexical scoping. For variables accessible in all Active Blocks of a single file (e.g. file-specific configuration options and flags), use the unqualified dynamic variables (and we advise not changing the current package in Active Blocks so that they remain easily accessible). In order to create variables visible to other files (e.g. public configuration options) or access such variables defined in other files, use dynamic variables that are fully qualified by the name of the package. For example, `$COSY::Timers::nMaxTimers` is a variable used to define a maximum number of timers supported by a timers library, and it can be accessed from any file, not just from the file with the library itself. The dynamic variables should be used with great care and properly initialized prior to their usage. They are accessible from anywhere in the program and thus can be accidentally or intentionally set to unexpected values causing problems that are hard to debug.

A.3.4 Full Processing

So far we reviewed the individual pieces of the *COSY++*. Here we review the general algorithm it follows to process a file. When the file for processing is specified by the user, *COSY++* initializes a new processing session by setting the parameters of the processing, checking the file for existence, and creating and initializing safe compartments for Active Blocks execution. Then it creates a new Sections Assembler, initializes it, and finally adds the specified file to the assembly. Before parsing the file into the sections and creating the assembly, the Sections Assembler pre-processes the file's pragmas and Active Blocks in the three passes:

1. `package` pragmas: searches for the first package and saves the package name for Active Blocks evaluation;

2. Active Blocks: executes everyting in the respective package; replaces their definitions in the source code with a result of the execution if AB is inclusive or with nothing, if it is non-inclusive);

3. The `assembly` pragmas, as described in section A.3.2.

Note that all ABs in the file are processed and executed before the `assembly` pragmas are porcessed, i.e. it does not matter if they are located before or after the pragmas.

The `assembly` pragmas processing is performed in the following order: every `assembly` pragma is parsed and the filename to assemble is extracted and then added to the assembly. The Sections Assembler then pre-processes this file using the same 3-pass processing and adds it to the assembly. On the third pass assembler can encounter other `assembly` pragmas, in which case the assembly process recursively continues forming assembly tree along the way.

Consider an assembly tree with the assemblying files on the top and the assembled files on the bottom. In this tree the Active Blocks are executed starting from the root (the first file added to the assembly) to the leaves, depth first. The sections, on the contrary, are assembled from the leaves to root, depth first. This approach allows the user to configure libraries he is adding to the assembly before adding their code. This logic is common to the source file inclusion processing mechanisms used by many other programming languages and macrolanguages. Note, however, two major differences between *COSY++* and these other languages:

- All Active Blocks in a file are executed prior to processing the first `assembly` pragma.
- The assembly method works differently from the commonly used whole-file-in-place inclusion method as described above.

A.3.5 Libraries

Libraries generally provide some additional functionality to a user and are packaged together by using a common factor in the services they provide. For example, the **logging.fh** library provides functions to open, write to, and close log fies, **conversions.fh** provides beam physics coordinate conversion routines, and **utils.fh** provides various utility functions to operate on vectors and other built-in *COSYScript* datatypes. Some of the libraries included to *COSY++* package provide configuration interfaces via Active Block variables, such as `$LogLevel`, `$nMaxLogFiles`, `$iFirstLogFileDescriptor`, `$LogFilesDir` in the **logging.fh** library. Special care must be taken in providing default values for these variables since a user can change them prior to configure library prior to use. Hence we need to check if the variable was already assigned a value before initializing it to a default value in the library.

Another issue is that having errors in Active Blocks does not stop the processing of the file and assembly. Rather, these errors are trapped in **eval**, shown to a user during processing, and otherwise ignored. Sometimes the severity of these errors is such that we want to stop further processing and exit with an error message. However, snce Active Blocks are executed by Perl in safe compartments (see the details before), termination of the processing is not possible unless it is explicitly permitted.

To address these (and possibly other) issues, the `AB` package provides two subroutines: `init_config_var` and **`die`**. The first subroutine takes a configuration variable name and either a reference to its initial value (can be any of Perl's basic data types: SCALAR, ARRAY, or HASH [173]) or its SCALAR value itslef (in case of the scalar variable initialization as a shortcut for this most frequently used datatype). It checks if this variable of this datatype (in Perl there can be several variables with the same name but different datatype) has not already been assigned a value. If it has not, then it is initialized with the provided value. Otherwise, the subroutine does nothing, thus preserving the value set by the user or some other library. The second subroutine, **`die`**, takes one argument: a message to show to the user. It outputs this message to the standard error stream and then stops any subsequent processing. Note that from Active Blocks these two subroutines should be called `AB::init_config_var` and `AB::`**`die`**, correspondingly.

As a self-explanatory example of the library variables initialization we take the real code from **logging.fh**:

```
{%
    AB::init_config_var(LogLevel, 1);
    AB::init_config_var(nMaxLogFiles, 10);
    AB::init_config_var(iFirstLogFileDescriptor, 50);
    AB::init_config_var(LogFilesDir, "./log/");
%}
```

A.3.6 Compatibility Mode

In order to make the user experience with the new system as painless as possible and to allow user to reuse the large code base accumulated over the years of *COSY Infinity* active usage, *COSY++* has the old syntax conversion mode. Before processing a file, *COSY++* tries to determine if the file was written in the pure *COSYScript* by checking for `section` pragmas and Active Blocks. If they are not found, the file is considered to be using the old syntax and an attempt to convert it to the new syntax is performed prior to the further processing. After replacing the original *COSYScript* inclusion with the assembly instructions and marking up the sections, the file is processed by *COSY++* using the new syntax rules.

Here we provide more details of the process. *COSY++* first checks if the file begins with the **`INCLUDE`** statement or the **`BEGIN`** statement. If it starts from the inclusion statement, *COSY++* extracts the filename from the statement and searches for it in the list of the available libraries. Two of the most commonly used *COSYScript* libraries are **cosy.fox** and **tm.fox** and they both have substitutes: **cosy.fh** and **tm.fh**, correspondingly. If a substitute has been found, the **`INCLUDE`** statement is replaced by the corresponding `assemble` pragma.

During the second and final step of this conversion, *COSY++* tries to determine the boundaries of the sections of the root block (see section A.2) and properly mark them with `section` pragmas. Note that this search used heuristics, hence it can fail for some files. Our tests, however, show that most of the old files can be converted, processed, and executed without problems.

One error case can occur if the sections' boundaries are placed to the comments that are put before the actual section boundary. *COSY++* does not recognize *COSY Infinity* comments so the sections in this case are marked up incorrectly. Anothe case is missing sections. Care is taken to correctly process these cases, but we cannot guarantee this will work correctly in all circumstances. Correct treatment of the sections requires a complete rewrite of the *COSYScript* parser which is out the scope for this work. Such rewrite might potentially be done provided significant demand from users. Some work in this direction has already been done as a part of the development of the autoconversion tool for the *COSYScript* to C++ programming language source file conversion [38].

A.3.7 Standard Libraries

COSY++ libraries are a part of the *COSY++* package and can be found in the **include** subdirectory of the package root directory. Below is a list of the currently implemented general purpose libraries:

- **conversions.fh** provides coordinate conversions between *COSY Infinity* particle coordinates and ICOOL coordinates and between *COSY Infinity* coordinates and MARS coordinates [68, 137]
- **logging.fh** provides log files manipulation and a logging interface
- **physics.fh** contains the definitions of the physical constants, their initializations and various functions for the commonly used formulae from accelerator physics

- **timers.fh** provides timing for the *COSY Infinity* scripts or the parts of them which are useful for code profiling
- **tracing.fh** provides an interface to the tracing procedures that generate output only when the certain tracing level is set; useful for degugging or maintaining output verbosity level consistency across different parts of the program
- **utils.fh** defines various convenience functions that are not built-in into *COSY Infinity*: seed-based random numbers generators, including random numbers from a Gaussian distribution, vector constructors, arithmetic and logical operations on vectors and matrices, logical indexing for them similar to the one used in Matlab, vector distances; domain scaling procedures for optimization functions

There are also libraries for optimization in the **optimization** subdirectory of the *COSY++* package:

- **gatool.fh** implements the real-valued functions optimizer based on the Genetic Algorithm described in this work (see Appendix B for details)
- **test_functions.fh** is a collection of the test functions for global minimizers; most of them possess properties that make them hard to optimize
- **lienard_jones.fh** is a set of test functions based on various Lienard Potential calculation problems; accessible from the **test_functions.fh** or as a standalone library
- **tm.fh** is a **TM.fox** standard *COSY Infinity* package containing a Taylor Models manipulation interface marked up for *COSY++*

and there are Beam Theory libraries in the **cosy** subdirectory:

- **cosy.fh** is the standard *COSY Infinity* package **COSY.fox** containing various Beam Theory computation and visualization algorithms marked up for *COSY++*

- **cosy_wrappers.fh** provides convenience functions to access certain elements of **cosy.fh**

Examples of appropriate library usage can be found in **examples** subdirectory of the *COSY++* distribution root directory.

A.3.8 Front End

In order to make all these features available to a user, the front end to the Sections Assembler and the Active Blocks processor is written in the Perl programming language. It exists in the form of a command-line script **cosy++.pl** and can be found in the root directory of the *COSY++* distribution along with a **readme** file that briefly covers its features and installation procedure. A more complete and detailed description of the command line parameters and the usage modes of the **cosy++.pl** can be obtained anytime by the calling the script from the command interpreter with the **-h** switch:

```
> cosy++.pl -h
```

COSY++ is under active development and details can change, but at the current moment it outputs the following information about the current version, developer, copyright, and usage:

```
Usage:
  cosy++.pl [-h]
  cosy++.pl [options] file1 [file2 ...]
  cosy++.pl [options] -a file [arg1 ...]
      --h[elp]
```

```
       Print usage information

     --v[erbose]
        Print information about the parsing process. Additive, i.e can be
        used several times for increased level of verbosity. Currently
        supported levels are 0, 1, 2

     --co[sy]=cosy_exec
        Execute the processed file with COSY Infinity using cosy_exec
        executable. By default uses "cosy_ni" in the directory set up
        during COSY installation

     --[no]e[xec]
        COSYInfinity execution flag. It is run if the flag is on and not
        run if it is off

     --cl[eanup]
        Perform cleanup after execution. Additive. Supported levels:
          0: no cleanup
          1: delete ".lis", ".cod" and "foxyinp.dat" COSY intermediate
             files
          2: delete processed file too

     --o[ut]
        Stores COSY Infinity output to file after execution. If not
        set or set to "", output to STDOUT (default)

     --i[nclude]=path
        Semicolon-separated list of directories to search in during
        assembly pragmas processing (usually where COSY++ libraries
        for COSY Infinity are stored)

     --s[uffixes]
        Semicolon-separated list of suffixes to be appended to files
        specified in assembly directives during assembly

     --a[arg-mode]
        In case this option is set, everything that follows first file name
        is treated as an argument to \COSYS{} and it can be accessed from
        the Active Blocks via Perl internal @ARGV array
```

```
  --p[roc-name]=processed_template_string
     String used to generate the name of the processed file. Could
     contain special variable names $base, $ext, $full which are
     replaced by basename of the file, its extension and the full
     name of file during processing correspondingly (defaul:
     "processed_$full)

  file
     File to process. In arg-mode everything that follows it is treated
     as a list of arguments to script accessible through Active Blocks

  file1 [file2 ...]
     If arg-mode is off, expect a list of files to process in the order
     they are supplied.  All options set are shared between all files
     in the list
```

As seen from the detailed help, a user can specify the path to search for the files from `assembly` pragmas, suffixes to be appended to filenames during this search, template for the processed filename(s), the name of the *COSY Infinity* executable to run the resulting file, the filename to store the textual output of the run, cleanup options and a level of the output verbosity for *COSY++*, in addtition to the name(s) of the files to process and arguments that get passed to them.

There exists another method to specify *COSY++* parameters that is particularly useful if the user typically executes *COSY++* with the same set of parameters and rarely needs to modify them. All these parameters can be stored in *COSY++* configuration files. All configuration files used by *COSY++* are named **.cosy++** but can be stored in different directories. Whenever *COSY++* is executed it sets its configuration parameters in the following order:

1. Default parameter values defined in the **cosy++.pl** source code

2. Parameters set in **.cosy++** stored in the directory with **cosy++.pl** script

3. Parameters set in **.cosy++** stored in the user home directory (obtained from the operating system)

4. Parameters set in **.cosy++** stored in the current directory (current as of the moment of execution)

5. Parameters set from the command line

Parameters that are not set on any of these stages will have the values set by the previous stage. The value of a parameter set on any one of these stages will override the value from all previous stages.

Such a scheme allows for a flexible configuration of the execution parameters, where the global parameters that are used most of the times can be stored in the main configuration file (processed on the stage 2) and the configuration parameters that are unique to some source file can be stored along with the file itself. A user does not need to set all of the parameters in this configuration file, only the ones he intends to override. For example, during the development phase a user can set the cleanup level to a minimum in the local configuration file in order to closely track all syntax processing errors. Later, when the code is stable and he does not need cleanup, he can simply remove the local configuration file to automatically switch to the settings specified in the global one. For trial runs with parameters changing from run to run, command line parameters configuration is more useful.

The syntax of the **.cosy++** configuration file is the same as what is typically used by Unix configuration files. It consists of the lines containing

```
name = value
```

pairs. Here `name` is a name of the configuration variable to set and `value` is a value to assign to this variable. Everything after the `#` symbol is considered a comment and thus ignored. A value string can contain expressions in the form `%name%` which will be replaced by the value stored in the variable `name` durping processing. This feature is useful for adding values to configuration variables instead of completely overriding them. For example, to add directory **c:/dir** to the search path for assembly files, the following syntax

```
ASSEMBLY_PATH = %ASSEMBLY_PATH%;c:/dir
```

is used. File **.cosy++** in the root directory of the *COSY++* distribution contains all the configuration parameters that *COSY++* recognizes along with their initializtion instructions.

A.3.9 Additional Features and Notes

One useful feature of *COSY++* is that the result of its processing (in case it was successful) is a valid *COSYScript* file. It can then be executed by *COSY Infinity* without any further modification. Hence, if there is a need to share the code that actively uses *COSY++* features with a plain *COSY Infinity* user, it is enough to process the file with *COSY++* and then share the results of processing.

Another feature of the *COSY++* which does not fit into the general list of features provided by Sections Assembler and Active Blocks is the extended verification of the *COSYScript* syntax. If the corresponding configuration flag is set, *COSY++* performs the check for global variable name clashes and warns the user if the global variable name is defined more than once. By global variables we mean variables defined in the VARIABLES section of the root block (see section A.2). By default, if there are several declarations of the variable with the same name, *COSYScript* silently uses the last one. With the new assembly mechanism in place, the number of the global variables can easily get very large and variable names can be unintentionally used multiple times. The behaviour of the program that results from assemblying these files is, at best, unpredictable. Bugs like this are very hard to find and fix, hence the variable name clash detection feature can be of great help to avoid them. In order to avoid problems with the global variables names clashes we recommend the user prefixes all global variables in the library by an acronym of the library name and uses long descriptive names, e.g. `GAToolStatus`. Additional *COSYScript* syntax checking can be added, e.g. *COSY++* can issue warnings about other common mistakes *COSYScript* programmers make.

Interaction between Active Blocks syntax and the *COSYScript* syntax is subtle. During the Active Blocks processing phase *COSY++* totally ignores the *COSYScript* syntax. It searches for the Active Block beginning and ending marks, and executes whatever it finds inside. Then it replaces the Active Block in the source code with the results of processing if the Active Block is of the inclusive type or with nothing if it is of the non-inclusive type. It then proceeds to the next Active Block and this process continues until the end of the file is reached. However, *COSY++* ignores *COSYScript* comments and any other syntax elements. One of the consequences of this is that an AB inside a valid *COSYScript* comment can still be processed which can contradict the user's expectation since *COSYScript* does not execute commented code. The contents of the Active Blocks are treated as Perl code, hence all comments inside them should be written in Perl style, not in *COSYScript* style which again can be confusing. Nested Active Blocks are not supported and should be avoided which can be confusing since *COSYScript* uses block structure that supports nesting.

The parsing algorithm of *COSY++* is based on heuristic and utilizes regular expressions [173] while ehe *COSYScript* structure is recursive. It is well-known that parsing of the recursive structures with regular expressions is very hard and is not even possible in general case [73]. To change this unfortunate situation we need to completely reimplement the *COSYScript* language parser. For example, we would need to omit searching for the beginnings and endings of the sections in *COSYScript* comments which can be recursive if sections are nested. It is worth noting that we did not find a lot of cases where the heuristic *COSY++* parser failed. To avoid problems with section recognition the user should try to avoid words such as "variable" in comments before the VARIABLES section, or "functions", or "procedure" before FUNCTIONS section.

Also note that while the assembly model greatly increases the flexibility and modularity of the code, it slows down the interpretation process. In the conventional *COSYScript* inclusion model the file to be included is already precompiled, while in the assembly mode it is inserted as text each time. Thus it is recompiled every time the file is processed. If the library to include/assemble is large, the conventional model provides for faster startup (the time passed between passing a filename to execute to *COSY Infinity* and the moment the program execution actually starts). We believe, however, that for large and complex projects the assembly model still constitutes a good tradeoff of some speed for significantly better code quality and modularity. For small projects it might still be worthwhile to use old model, but new convenience libraries provide a lot of useful services that speed up the development process so there are still some reasons to use *COSY++*. The old syntax conversion mode can help in making the transition from the conventional *COSYScript* usage model the to new one easier.

COSY++ code is modular, well-documented and easily extensible, so missing features can be added and limitations resolved in the future given significant demand generated by users. As of now this is out of the scope of this work.

APPENDIX B

The Genetic Algorithm Tool (GATool) in COSY Infinity

B.1 Introduction

GATool is a real-valued function optimization package for *COSY Infinity* that implements the Evolutionary Optimizer from section 2.3. It is designed in such a way that it should work well for most problems with either the default values of the parameters or a minimal amount of fine-tuning. However, some problems might be solved better and/or faster with non-default algorithm parameters thus the parameter configuration interface was developed for GATool. The parameter initialization must be done before starting the actual minimization process, as the change of the parameters during the run process is not supported and its consequences are unpredictable. The methods to get access to the statistics of the current run and the best value found on each step are also provided. Interfaces, configuration parameters and the values they can take, the default parameter set and typical GATool usage patterns, are described in this appendix.

B.2 Configuration

```
procedure SetCreationParams CreationAlg;
```

Sets the creation algorithm used in the initial population generation and regeneration of the eliminated members of the population. It supports the following `CreationAlg` values:

```
CreationAlg = 1 { UNIFORM creation algorithm, i.e. new member of the }
                { population is any point in the initial box with    }
                { uniform probability                                 }
```

```
procedure SetArealParams ivInitBox Scale ivGlobalBox IsKillingOn;
```

Sets the initial box and global box parameters. The first parameter, `ivInitBox` is a vector of intervals that defines search ranges for the coordinates. The second parameter, `Scale`, is a scaling coefficient. The effective initial box is generated from `ivInitBox` by multiplying the lengths of its sides by this coefficient (note that in such case the volume of the box changes as the n-th power of the `Scale`). The scaling coefficient is introduced to simplify the exploration of the problem, i.e. if a user wants to try running the algorithm with a smaller or larger box, he does not have to manually rescale each interval in `ivInitBox`. Instead, he can simply change the scaling coefficient. The initial population is generated in the scaled `ivInitBox`.

The third parameter, `ivGlobalBox` is a vector of intervals and in most cases it should contain the scaled initial box. It is used along with the last parameter, `IsKillingOn`. In the case that parameter is set to a non-zero value, all the members of the population outside of `ivGlobalBox` are eliminated and then replaced by new members generated using the creation algorithm specified by `SetCreationParams`.

This rule is applied on each step of the optimization process. Hence if the killing mode is on, GATool guarantees that all members of every generation stay inside `ivGlobalBox`. If the scaled initial box is not a proper subset of the global box, GATool issues a warning, but proceeds with the execution. In this case the members of the population that are in the scaled initial box but not in the global box are eliminated if the killing mode is on.

```
procedure SetInitialPopulation nInitPopSize aInitPop;
```

Some portion of or the whole initial population can be predefined with this procedure. This feature is particularly useful if a user has some insights about the function's behaviour (perhaps, obtained by using GATool with the parameters tuned for the exploration of the search space, some other optimizer, or analysis). This subroutine provides an interface to transform these insights into a hint for the GATool optimizer. The information in our case consists of the points in the search space that user considers to be potential minimizers or in their close proximity. The two parameters the procedure takes are `nInitPopSize` and `aInitPop`. They define the size of the initial population (must be less than or equal to `g_nPopSize`) and its members, correspondingly. Here `aInitPop` is an array with `g_nDim` elements. Each of the elements is a vector of `g_nPopSize` length, so that `aInitPop(i)` is a vector containing all the i-th coordinates of all the members of the initial population.

```
procedure SetReproductionParams nElite MutationRate;
```

Sets the ratio of the members of the next generation generated by each of the available new members generation methods. There are three types of members in the next generation: the elite, the mutated and those produced by a crossover. The elite members are the best members (the points that provide the smallest values to the minimized function) and as such they are transferred from the previous generation without changes. A number of these members is set by the `nElite` parameter, which thus must be a non-negative integer less than or equal to the population size `g_nPopSize`. The mutated members are the ones produced by mutating members of the previous generation using the chosen mutation algorithm. The `MutationRate` parameter defines the percentage of the new population that is generated by the mutation. It must be a real value from the $[0, 1]$ range. The actual number of mutants is then `g_nPopSize*MutationRate`. The number of elite children plus the number of the mutants must be less then or equal than the population size. If this sum is less than the population size, all the remaining members of the population are generated by crossover.

There are three forces that affect evolution in a genetic algorithm: the exploration, the exploitation and the conservation (see [148] for a more detailed study and explanation of the similar concepts of compression, transmission and neutrality selection and their interplay in the evolution process). The exploration is responsible for exploring the search space by moving in mostly random directions in hopes of finding areas of interest. The exploitation is a more careful examination and refinement of these areas aimed at finding a minimum. The conservation is responsible for preserving the best values found so far. The elite members of the population drive the preservation, mutated members drive the exploration, and the members produced by the crossover drive the exploitation. Hence by controlling the reproduction parameters, a user controls the impact of these forces on the evolutionary search and, as such, the performance of the method which can be made more exploratory or quickly converging to a local minimum. The process of selecting a right set of parameter values is mostly heuristic, involves trial and error, and non-trivially depends on the problem.

```
procedure SetFitScalingParams FitScalingAlg;
```

Sets the fitness scaling algorithm. Currently supported algorithms (described in detail in Section 2.3) include:

```
FitScalingAlg = 1 { LINEAR }
FitScalingAlg = 2 { PROPORTIONAL }
FitScalingAlg = 3 { RANK }
```

Fitness scaling transforms the function values in any finite range to the fitnesses in the $[0, 1]$ range in order to make comparison between different function values domain-independent. Since the search is directed towards the minimum of the function, larger fitnesses correspond to smaller values of the function. To perform this transformation, LINEAR and PROPORTIONAL scaling algorithms map the function values to the desired interval by means of multiplication and addition while RANK algorithm sorts them and then assigns fitnesses according to the positions in the sorted list. Of these methods, RANK is the slowest because sorting that it employs is of the order of $O(n \log n)$, where $n =$ `g_nPopSize`. However, at the same time, this scaling algorithm is the least sensitive to numerical errors since it does not involve any mathematical operations on function values. The other two involve subtraction which can lead to the cancellation errors if the function assumes small values on the search domain.

```
procedure SetCrossoverParams CrossoverAlg CrossoverParams;
```

Sets the crossover parameters. The first parameter, `CrossoverAlg`, sets the type of algorithm. Currently GATool supports only one crossover algorithm:

```
CrossoverAlg = 1 { HEURISTIC }
```

The second argument, `CrossoverParams`, is an array with the parameters of the crossover algorithm. For the HEURISTIC algorithm the following parameters are supported:

```
CrossoverParams(1) { ratio (scalar or vector) of the distance between two }
                   { parents where the child is created                   }

CrossoverParams(2) { randomization flag which determines if the effective }
                   { ratio is multiplied by a random number before usage  }
```

This algorithm creates a child on the line connecting two parents if the ratio is a scalar, or in the hypercube determined by two parents and the ratio if the ratio is a vector. If the scalar ratio is > 0.5, then the child is created closer to the better parent, if it is in $[0, 0.5)$ the child is created closer to the worse parent, and if it is 0.5 the child is created exactly in the middle. If the ratio is negative, the meanings are reversed. The recommended range of values for the ratio is $[0, 2]$. In case of the vector ratio, each of these rules applies coordinate-wise. This crossover algorithm is described in section 2.3.

```
procedure SetHeurCrossover Ratio IsRandomize;
```

More user-friendly interface to set the HEURISTIC crossover algorithm and its parameters. See the description of the `SetCrossoverParams` in this section for a description of the parameters.

```
procedure SetMutationParams MutationAlg MutationParams;
```

Sets the mutation algorithm parameters. The first parameter, `MutationAlg`, selects the mutation algorithm. Currently supported are the following types of the mutation algorithms:

```
MutationAlg = 1 { UNIFORM }
MutationAlg = 2 { FADING GAUSSIAN }
```

The UNIFORM algorithm first checks the mutation probability for each coordinate of every member and, if the mutation occurs, replaces the coordinate value by a randomly generated number from the initial box' corresponding range. The FADING GAUSSIAN algorithm generates a vector of coordinates each that from the Gaussian distribution centered at 0 with a mean equal to the width of the corresponding range of the initial box, multiplied by the scale and fading parameter, and then adds it to a mutated member to produce a mutant. Details of the algorithms are described in Section 2.3.

The second argument, `MutationParams`, is an array containing parameters of the selected mutation algorithm. For UNIFORM mutation only one parameter is supported:

```
MutationParam(1) { gene mutation probability which specifies the        }
                 { probability with which each gene of every member     }
                 { of the population selected for mutation is mutated   }
```

For FADING GAUSSIAN the following parameters are supported:

```
MutationParam(1) { scale to determine the Gaussian distribution's mean   }
                 { value (scale = 1 corresponds to the full length of    }
                 { the box along coordinate)                             }
MutationParam(2) { shrink factor, that determines the speed with which   }
                 { mean value shrinks with generations (shrink factor = 0}
                 { corresponds to no shrinking; allowed values range is  }
                 { [0,1])                                                }
```

```
procedure SetUnifMutation GeneMutProb;
procedure SetGaussMutation Scale ShrinkFactor;
```

Is a more user-friendly way to set UNIFORM or FADING GAUSSIAN mutation algorithms and their parameters.

```
procedure SetSelectionParams SelectionAlg;
```

Sets the selection algorithm parameters. Currently there is only one parameter supported and it determines the algorithm that selects the members of the population for the mutation and crossover. Currently supported selection algorithms include:

```
SelectionAlg = 1 { ROULETTE }
SelectionAlg = 2 { STOCHASTIC UNIFORM }
SelectionAlg = 3 { TOURNAMENT }
```

All of these algorithms use information about members' fitnesses to select with higher probability the members with better fitness for the reproduction. The method used to exploit this information depends on the algorithm. The details of these algorithms are described in section 2.3.

B.3 Usage Scenarios

A typical usage scheme of the GATool is:

```
GA_Init ProblemDim PopulationSize RandomSeed;

{ Set initial population }
{ Set various algorithm parameters }
{ Set stopping criteria }
```

```
GA_InitProblem;
while g_GAToolStatus#0;
    g_vFValues := OBJ_FUNC(g_aNextPopulation, g_nDim);
    GA_Step;
endwhile;
GA_FinalizeProblem;
```

Here comments denote the placing of the optional GATool configuration procedures described in the previous section and `OBJ_FUNC` is a function that GATool is minimizing. Note that the function value computation method is left to a user. In order to proceed with the search process GATool only needs function values evaluated at the points stored in `g_aNextPopulation`, which is an array that contains `g_nDim` vectors with population member coordinates. The population array's i-th element is a vector containing the i-th coordinates of all the members of the population. Function values must be stored in `g_vFValues` in the same order they are stored in the population array.

A member can be extracted from the array as a vector by calling `GetPopulationMember`, the function described in this section. Note that this involves using temporary variables and does not exploit vector operators optimized by *COSY Infinity*; hence such practice is generally ineffective and should be avoided. A more efficient (however not always applicable) method is to design `OBJ_FUNC` such that it takes the population array and dimension as its arguments, returns the vector with function values at these points, and uses the vector operators to compute this result. Vector manipulation functions from the **utils.fh** library (see section A.3.7) might prove themselves particularly useful for the task. Below is an example of the function designed to take advantage of these vector operations (from the **test_functions.fh** library):

```
function SchwefelFcn x nDim;
```

```
    variable i 1;

    SchwefelFcn := 0;

    loop i 1 nDim-1;
        SchwefelFcn := SchwefelFcn + (x(i)*sin(SQRT(VectorAbs(x(i)))));
    endloop;

    SchwefelFcn := 418.9829*nDim - SchwefelFcn;
endfunction;
```

A user is free to put and execute arbitrary code before calling the `GA_Step` procedure to proceed with the next step. Such flexibility allows him to build arbitrarily complex optimization scenarios on the base of GATool. A user might perform data manipulations of his own, change the initial and global boxes (which is particularly important for the COSY-GO interaction described in details in section 2.3.6), use other optimizers, get interactive input, etc. thus fine-tuning the optimization process to his needs. If any of the stopping criteria are satisfied, GATool sets the `g_GAToolStatus` variable to zero, causing the main **`while`** loop in the example to stop execution. The `g_StopReason` variable indicates the reason for stopping.

Here is the list of the procedures used to initialize and finalize GATool, set the stopping criteria, and perform one step of the optimization.

```
procedure GA_Init Dim PopSize Seed;
```

Initializes the GATool. Sets the dimension and the population size (these can be shared by several different problems), which also implicitly defines a lot of internal parameters' and buffers' sizes. It also sets the default values of the parameters. The last parameter, `Seed`, sets the initial seed for the pseudorandom number generator and can be used to reproduce GATool runs. The pseudorandom numbers generator implemented in *COSY Infinity* produces exactly the same sequence of the random numbers being started from the same seed. Hence, two runs on the same problem with the same value of `Seed` (and other parameters) will be identical in both the intermediate and final results. If `Seed` is set to `-1`, the value of seed is generated randomly using the computer's internal clock as a source of randomness. This procedure should be called before any other GATool subroutine.

```
procedure SetStoppingCriteria nMaxGenerations nMaxStallGenerations
                              DesiredMinFValue IsStopDesiredMinFVal RelTol;
```

Sets various stopping criteria for the algorithm. The first argument, `nMaxGenerations`, if positive, sets the limit on the maximum number of generations created during the optimization (essentially the maximum number of steps). The second argument, `nMaxStallGenerations`, if positive, sets the limit on the maximum number of steps on which the best found function value (minimal in our case) changes by less than the tolerance set by the last argument, `RelTol`. The third and fourth arguments determine if the algorithm stops when the desired minimal function value is reached or exceeded. Here `DesiredMinFValue` specifies the desired minimal value and the `IsStopDesiredMinFVal` flag turns the checking on (when set to any non-zero value) or off (when set to a zero).

```
procedure SetMaxRunTime MaxRunTime;
```

Sets another stopping criteria for the algorithm: maximum run time in seconds, specified by `MaxRunTime`. It must be positive.

```
procedure GA_InitProblem;
```

Performs the initialization of the problem-specific data structures. It opens the log files, starts the timers, initializes the statistics, generates the initial population, and outputs the GATool parameters to **results** log file. Then it sets the `g_GAToolStatus` variable to a non-zero value to indicate that the search is in progress. This procedure should be called after all the parameters of the method are set but before the first call to `GA_Step`.

```
procedure GA_Step;
```

Performs one step of the search process. In order to perform it correctly, algorithm needs `g_vFValues` to contain the values of the function calculated at the points stored in `g_aNextPopulation`. The points themselves are generated by GATool but the computation of the function values is left to a user. The procedure also updates the statistics (including the current best minimizer) and writes this information to the log files. It also checks the stopping criteria. If at least one of them is satisfied, it sets `g_StopReason` to a non-zero value, indicating the exact reason for stopping, and `g_GAToolStatus` to zero, indicating that the minimization is completed. In the current version the following reasons for stopping are supported (correspond to the GATool stopping criteria):

```
g_StopReason = 1 { Maximum number of generations is reached           }
g_StopReason = 2 { Maximum number of stall generations (when the       }
                 { minimum value of the function changes by less then   }
                 { the specified tolerance) is reached                  }
g_StopReason = 3 { Desired minimal function value is reached }
g_StopReason = 4 { Time limit is reached }
```

If the minimization process is not completed, the procedure then generates the next population and stores it in `g_aNextPopulation`. It should be called after `GA_InitProblem`.

```
procedure GA_FinalizeProblem;
```

Closes the log files, shutdowns the timers, performs the internal cleanup and prints the execution timings. It should be called when the optimization process is completed.

B.4 Access to Statistics

There are many internal variables used by GATool to store the statistics, the current and next populations, the function values and fitnesses, the stopping criteria, the log files' descriptors, timers, etc. Since all of them are defined as global variables, a user can potentially access these variables directly, but is strongly discouraged to do so. The internal representation of the algorithm structures is the implementation detail and thus is subject to a future changes by a GATool developer. What the user should rely on is an open interface in the form of procedures and functions designed to provide open access to internally available information. This interface forms a contract between a tool developer and a user. For example, if the procedure is designed to return the number of the elite members of the population to a user, it would return this number even if the internal name of the variable holding this value or the whole set of the internal structures storing these members was changed. Were a user accessing this variable by its name or making any assumption about storage mechanism, he would have make changes across all of his code.

In the current version of GATool the following routines provide access to its internal statistics:

```
procedure GetCurBestMemberVec vCurBestMember FMin;
procedure GetCurBestMemberArr aCurBestMember FMin;
```

Both of these procedures return the current best member of the population (point in the search space) and the corresponding value of the function at this point (minimal since the method is performing a minimization). The difference between them is that the first one returns the current best member as a vector while the second one returns it as an array. The values are returned through the procedures' arguments.

```
function GetPopulationMember aPopulation iIndex;
```

Takes the population array and an index of the member of the population and returns the population member (a point in the search space) in vector format. In the current version of GATool there are two population arrays: `g_aCurPopulation` and `g_aNextPopulation`, The second argument, `iIndex`, can assume values from `1` to `g_nPopSize`.

B.5 Default Parameters Set

The default set of the configuration parameters is shown on the Figure B.1 It is tested to work reasonably well for a large class of the optimization problems.

```
Reproduction:    number of elite = 10, mutation rate = 0.2
Mutation:        UNIFORM, gene mutation probability = 0.1
Crossover:       HEURISTIC, ratio = 0.8, randomization is on
Fitness scaling: RANK
Selection:       STOCHASTIC UNIFORM
Creation:        UNIFORM
Areal:           initial box = [-10,10] x ... x [-10,10]
                 global box  = [-10,10] x ... x [-10,10]
                 killing is off
Stopping:        max generations = 1000,
                 stall generations = 25,
                 tolerance = 1E-5
```

Figure B.1: *GATool's default parameters*

B.6 Miscellaneous

```
procedure SetDumpFValues;
procedure UnsetDumpFValues;
```

Turns on and off the mode where all the points where the function is evaluated during the optimization along with the evaluated function values, are stored in the **f_values** log file in the log files' directory. It can be used, for example, to plot a function that is expensive to calculate.

B.7 Advanced Configuration via Active Blocks

Apart from the *COSYScript* subroutines, some of the GATool parameters can be configured using Active Blocks (described in detail in Appendix A). Some configuration parameters are available only through Active Blocks (they define dynamic variable sizes and internal GATool implementation details, especially experimental ones), some through both Active Blocks and *COSYScript* subroutines, and some only through *COSYScript* subroutines. In order to configure GATool using Active Blocks, a user should set the values of the variables described in this section before adding a library file to an assembly (as described in section A.3.5).

Configuration variables that can be set from Active Blocks only, along with their default values, are:

```
$COSY::GATool::MaxDim = 20
```

Specifies the maximum dimensionality of a problem that can be set via `GA_Init`.

```
$COSY::GATool::MaxPopSize = $COSY::GATool::MaxDim * 100;
```

Specifies the maximum population size that can be set via `GA_Init`.

```
$COSY:GATool::nMaxMutationParams = 3
```

Specifies the maximum number of mutation params allowed.

```
$COSY:GATool::nMaxCrossoverParams = 2
```

Specifies the maximum number of crossover params allowed.

```
$COSY:GATool::IsSuppressIncestCrossover = 1
```

Determines if incests are suppressed during crossover. Incest refers to the case where both parents in a crossover correspond to the same point in the search space. By the nature of the crossover algorithm implemented in GATool, this would result in a child that is exactly this point which, in turn, leads to a premature convergence of the search process. Thus this case should be avoided. The incest suppression is particularly important for small population size settings. The population can be small, for example, due to the expensiveness of the objective function calculation.

```
$COSY:GATool::IsShuffleAfterSelection = 1
```

If this flag is set to a non-zero value, an additional shuffling of indices is performed after selection. This is done to prevent the premature convergence and increase the diversity by additionally mixing the population.

```
$COSY::GATool::nElite = 10
$COSY::GATool::MutationRate = 0.2
$COSY::GATool::MaxInitBoxSize = 10
$COSY::GATool::RelTol = 1E-5
$COSY::GATool::InitPopSize = 10
$COSY::GATool::IsKilling = 0
$COSY::GATool::PopCreationAlg = 1
$COSY::GATool::FitScalingAlg = 3
$COSY::GATool::CrossoverAlg = 1
$COSY::GATool::CrossoverRatio = 0.8
$COSY::GATool::CrossoverIsRandomize = 1
$COSY::GATool::MutationAlg = 1
$COSY::GATool::MutationScale = 0.8
$COSY::GATool::MutationShrinkFactor = 0.6
$COSY::GATool::MutationGeneMutProb = 0.1
$COSY::GATool::SelectionAlg = 2
$COSY::GATool::StoppingMaxGens = 1000
$COSY::GATool::StoppingStallGens = 25
$COSY::GATool::StoppingMinFValue = undef
$COSY::GATool::MaxRunTime = undef
```

These variables set the default values of the GATool parameters (see Figure B.1).

```
$COSY::GATool::RandomSeed = -1
```

This variable sets the random seed used if the last argument to `GA_Init` is non-positive. If this variable is set to a positive value, GATool always starts from the predefined seed. Note, however, that a positive argument to `GA_Init` overrides the value set from the Active Block. It is not recommended to change the value of this

Active Block	*COSYScript*	Random Seed's Value
-	-	randomly generated
-	+	*COSYScript*: from argument to `GA_Init`
+	-	Active Block: from `$COSY::GATool::RandomSeed`
+	+	*COSYScript*: from argument to `GA_Init`

variable; this feature is mainly provided for debugging purpose.

```
$COSY::GATool::IsDumpFValues = 0
```

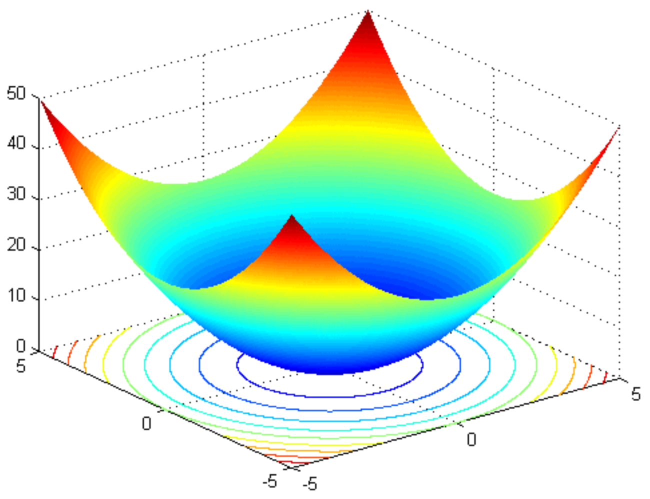
Stored the default value of the flag that controls the mode of the points and function values dumping to the log file, as described earlier.

APPENDIX C

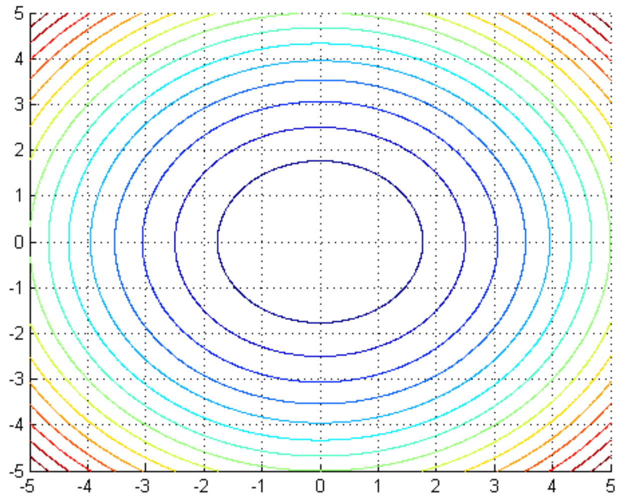
Test Problems in Unconstrained Optimization

Every new optimization method has to prove its worthiness and justify the time and effort spent in its development, implementation, and testing. Some of the optimization methods aim at achieving a reasonable performance on a larger class of the problems, others willingly narrow the target class in order to achieve better performance on it. To assess the performance of different methods, compare their strengths and weaknesses, reveal the unforeseen aspects of their behaviour, and stress-test them, a large number of test problems was invented and examined [61]. In order to test the behaviour of GATool (see section 2.3) we selected some of the most commonly used test problems representing different aspects of of the difficulty of the optimization process. In this appendix we present the formulations of these problems along with their characteristics and example plots for the 2-dimensional cases (most of these problems are defined for a general n-dimensional case).

C.1 Sphere Function



(a) 3D plot

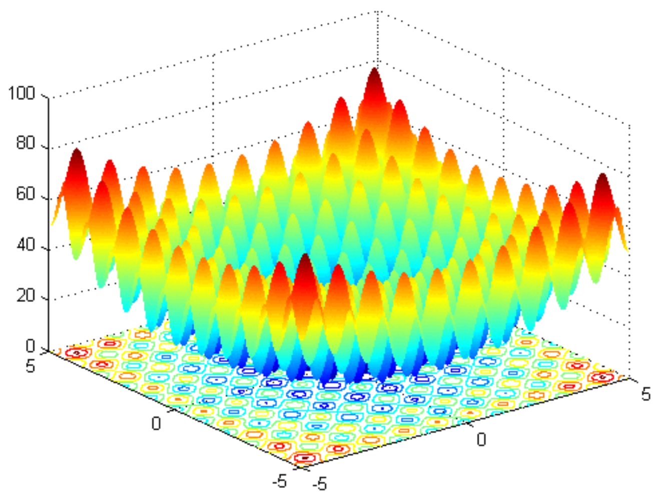


(b) Contour lines plot

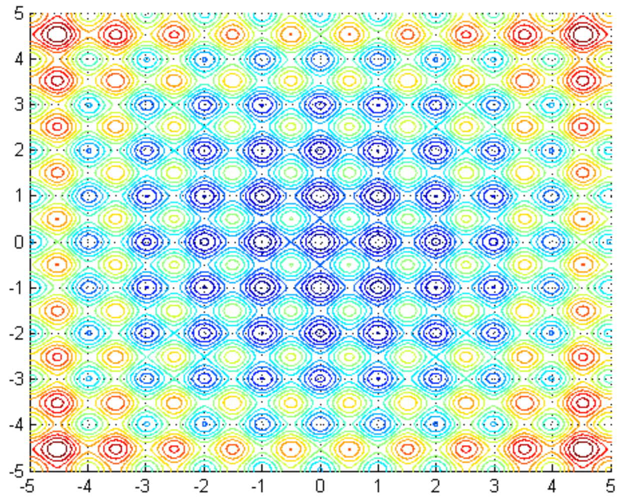
Figure C.1: *Sphere function*

```
# **Definition**
# $f(\mathbf{x}) = \sum_{i=1}^{v} x_i^2$
# **Search domain**
# $x_i \in [-6, 6], \;\; i = 1, 2, \ldots, v$
# **Local minima**
# One, same as global
# **Global minimum**
# $\mathbf{x}^* = (0, \ldots, 0), \;\; f(\mathbf{x}^*) = 0$
# **Description**
# The simplest function for the conventional minimization methods:  smooth,
# symmetric and unimodal.  The gradient is directed towards the global
# minimum at any point.  Using a right step size, a conventional minimizer
# can reach the global minimum in one step starting from any initial
# point.  This problem is used to test the performance of the global
# optimizer on the simplest case.
```

C.2 Rastrigin's Function



(a) 3D plot

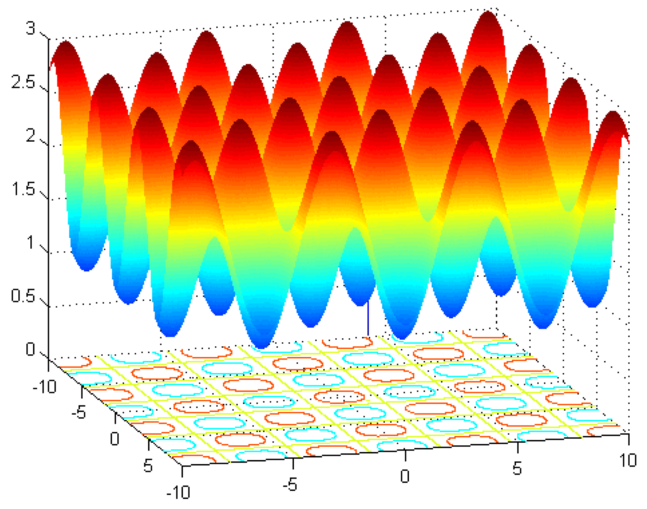


(b) Contour lines plot

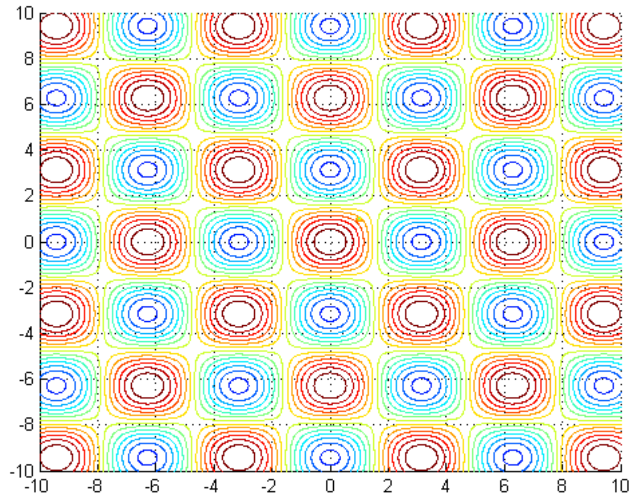
Figure C.2: *Rastrigin's function*

```
# Definition
# $f(\mathbf{x}) = 10v + \sum_{i=1}^{v} \left(x_i^2 - 10\cos(2\pi x_i)\right)$
# Search domain
# $x_i \in [-6, 6], \;\; i = 1, 2, \ldots, v$
# Local minima
# Lots
# Global minimum
# $\mathbf{x}^* = (0, \ldots, 0), \;\; f(\mathbf{x}^*) = 0$
# Description
# Sphere function with added oscillatory behaviour which leads to a
# large number of the local minima.  Conventional methods get stuck at
# one of the local minima.
```

C.3 CosExp Function



(a) 3D plot

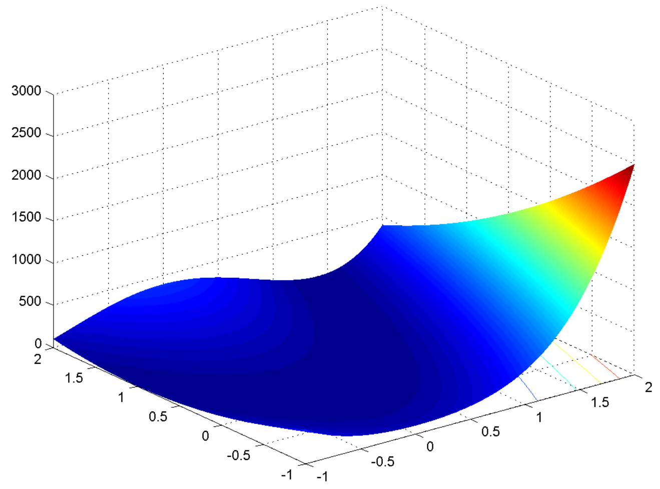


(b) Contour lines plot

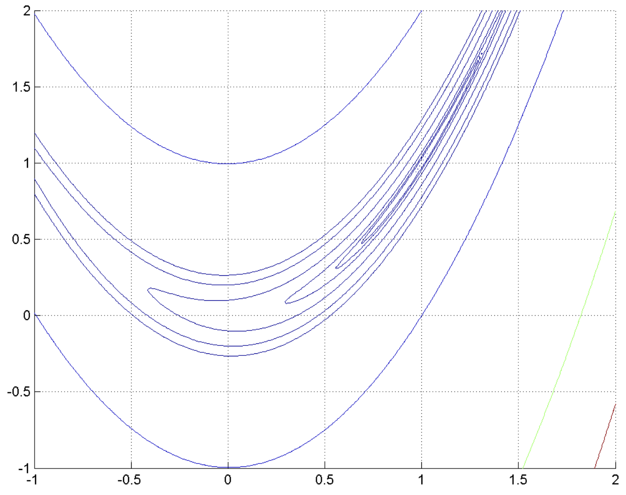
Figure C.3: *CosExp function*

```
# **Definition**
# $f(\mathbf{x}) = \prod_{i=1}^{v} \cos(x_i) - 2\exp(-10\sum_{i=1}^{v}(x_i - 1)^2)$
# **Search domain**
# $x_i \in [-4, 4],\ \ i = 1, 2, \ldots, v$
# **Local minima**
# Lots
# **Global minimum**
# $\mathbf{x}^* = (1, \ldots, 1)$
# $n = 2,\ \ f(\mathbf{x}^*) \approx -1.7081$
# $n = 3,\ \ f(\mathbf{x}^*) \approx -1.8422$
# $n = 4,\ \ f(\mathbf{x}^*) \approx -1.9147$
# $n = 5,\ \ f(\mathbf{x}^*) \approx -1.9539$
# $n = 6,\ \ f(\mathbf{x}^*) \approx -1.9751$
# $n = 7,\ \ f(\mathbf{x}^*) \approx -1.9865$
# $n = 8,\ \ f(\mathbf{x}^*) \approx -1.9927$
# $n = 9,\ \ f(\mathbf{x}^*) \approx -1.9960$
# $n = 10,\ \ f(\mathbf{x}^*) \approx -1.9978$
# **Description**
# Highly oscillatory function with many local minima and a very
# sharp and well-pronounced global minimum that has a tiny domain
# of attraction.  This property makes this minimum extremely hard
# to find especially for high-dimensional formulations.
```

C.4 Rosenbrock's Function



(a) 3D plot



(b) Contour lines plot

Figure C.4: *Rosenbrok's function*

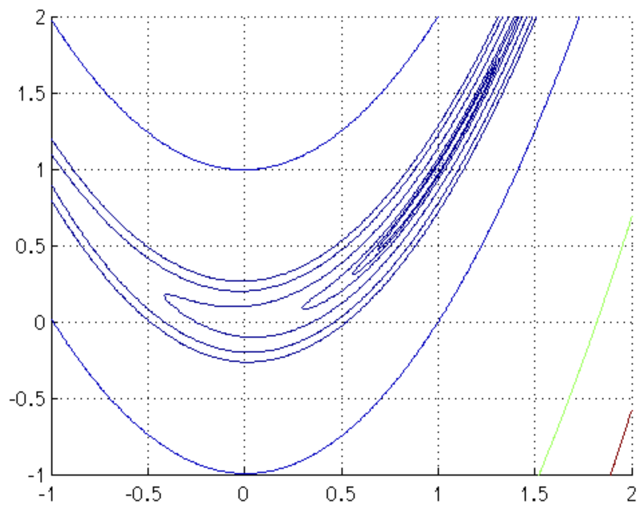


Figure C.5: *Rosenbrock's function contours near the minimum*

# **Definition**

# $f(\mathbf{x}) = \sum_{i=1}^{v-1} \left(100\left(x_i^2 - x_{i+1}\right)^2 + (x_i - 1)^2\right)$

# **Search domain**

# $x_i \in [-5, 10], \quad i = 1, 2, \ldots, v$

# **Local minima**

# One, same as global

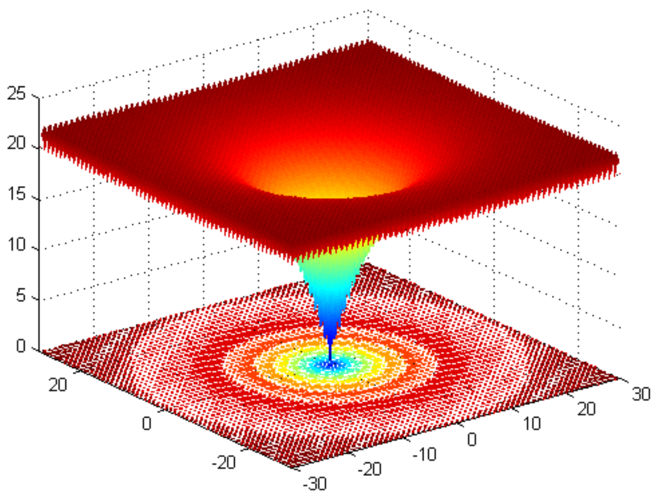
# **Global minimum**

# $\mathbf{x}^* = (1, \ldots, 1), \quad f(\mathbf{x}^*) = 0$

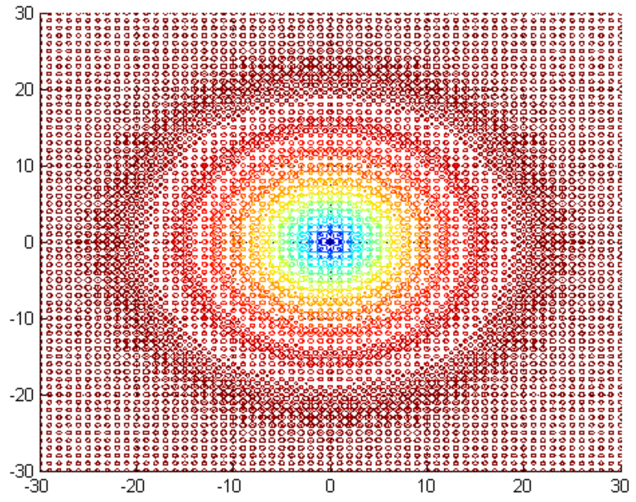
# **Description**

# Is also called “banana” function for its banana-shaped contour lines
# (see Figure C.5 for magnified contour plots near the minimum).
# It poses difficulties for conventional minimizers due to its flat
# landscape and behaviour near the minimum. In the very narrow contour
# “valleys” around it, the gradient is pointing almost perpendicular
# to the direction towards the minimum. This can result in zigzag
# movements with small step sizes during the optimization. Minimization
# process frequently stops exhausting an allowed number of steps.

C.5 Ackley's Function



(a) 3D plot

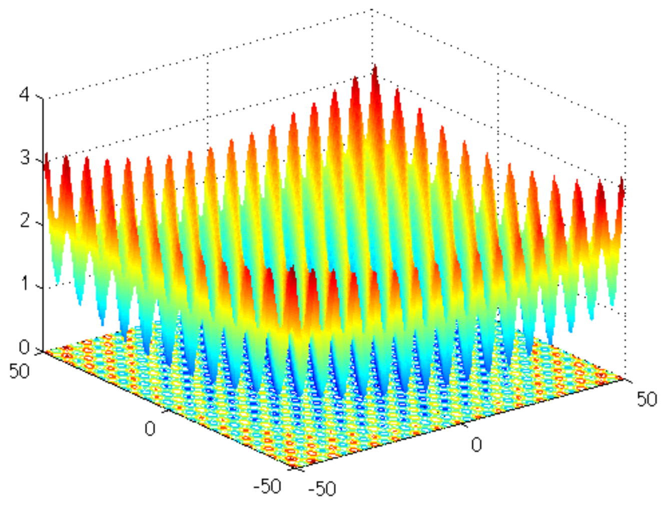


(b) Contour lines plot

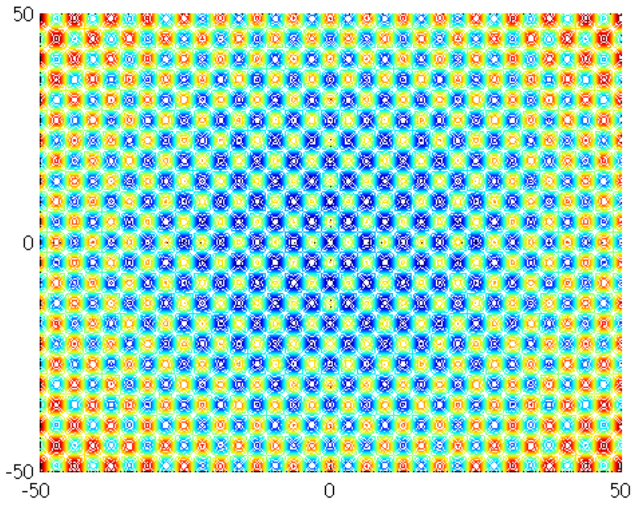
Figure C.6: *Ackley's function*

```
# **Definition**
# $f(\mathbf{x}) = 20 + 20\exp(-5^{-1}(v^{-1}\sum_{i=1}^{v} x_i^2)^{1/2}) - \exp(-v^{-1}\sum_{i=1}^{v}\cos(2\pi x_i))$
# **Search domain**
# $x_i \in [-30, 30],\ \ i = 1, 2, \ldots, v$
# **Local minima**
# Several
# **Global minimum**
# $\mathbf{x}^* = (0, \ldots, 0),\ \ f(\mathbf{x}^*) = 0$
# **Description**
# Oscillations with small amplitude, very sharp global minimum with its
# pit hidden in a large number of the local minima.  Conventional methods
# get trapped at one of the local minima.
```

C.6 Griewank's Function



(a) 3D plot

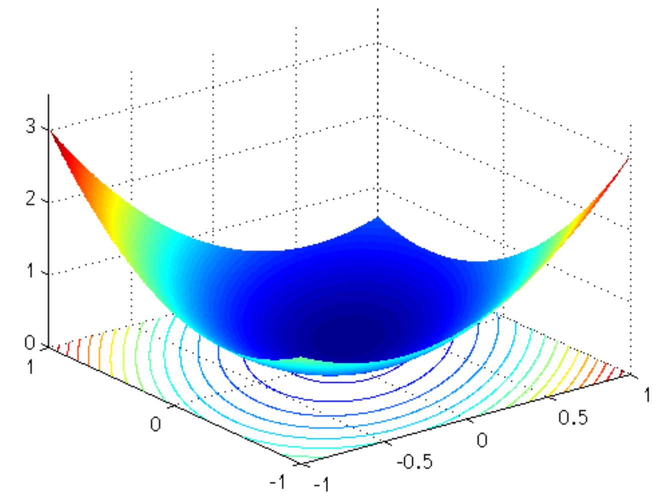


(b) Contour lines plot

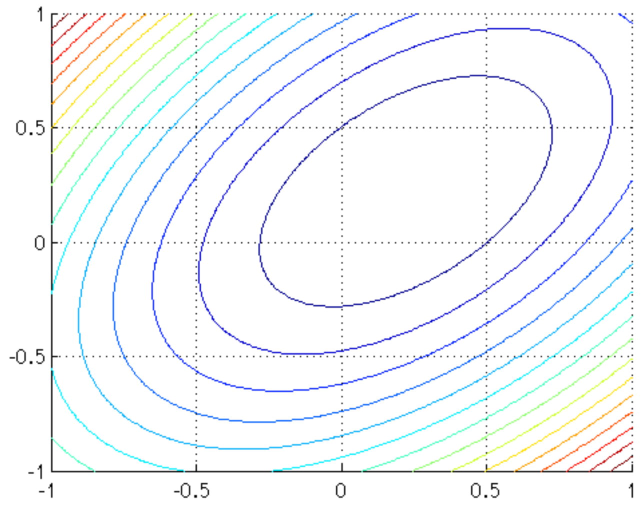
Figure C.7: *Griewank's function*

```
# **Definition**
# $f(\mathbf{x}) = 4000^{-1}\sum_{i=1}^{v} x_i^2 - \prod_{i=1}^{v}\cos(x_i \cdot i^{-1/2}) + 1$
# **Search domain**
# $x_i \in [-600, 600],\;\; i = 1, 2, \ldots, v$
# **Local minima**
# Several
# **Global minimum**
# $\mathbf{x}^* = (0, \ldots, 0),\;\; f(\mathbf{x}^*) = 0$
# **Description**
# Sphere function modulated by oscillations introduced by the cosine
# component.  Number of local minima increases exponentially with
# dimension.  It is frequently used as a test problem for the global
# optimization methods.  See, however [111] for its criticism due to a
# complexity reduction with the dimension and proposed modifications to
# improve the difficutly.
```

C.7 An Function



(a) 3D plot

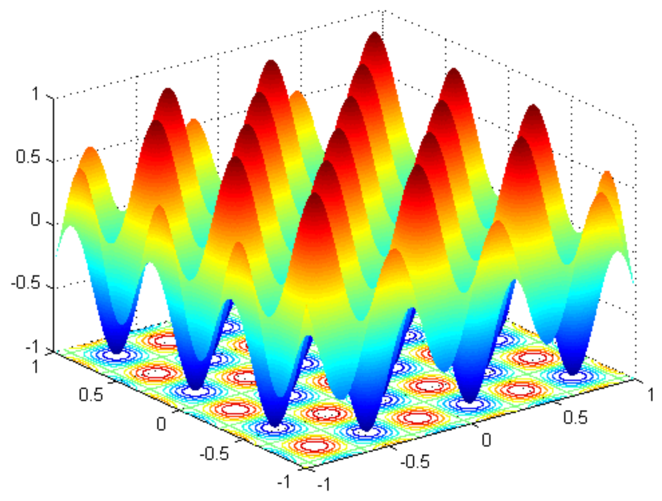


(b) Contour lines plot

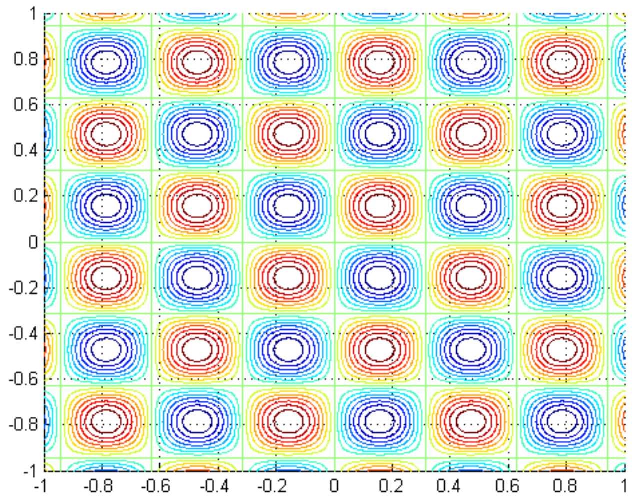
Figure C.8: *An function*

```
# Definition
# $f(\mathbf{x}) = (x_1 - (v+1)^{-2})^2 + \sum_{i=2}^{v}(x_i - (v+1)^{-2})^2 - x_i x_{i-1}$
# Search domain
# $x_i \in [-0.25, 0.25],\ \ i = 1, 2, \ldots, v$
# Local minima
# One, same as global
# Global minimum
# $n = 2,\ \ f(\mathbf{x}^*) \approx -0.0247;$
# $n = 3,\ \ f(\mathbf{x}^*) \approx -0.0273;$
# $n = 4,\ \ f(\mathbf{x}^*) \approx -0.0256;$
# $n = 5,\ \ f(\mathbf{x}^*) \approx -0.0231;$
# $n = 6,\ \ f(\mathbf{x}^*) \approx -0.0208;$
# $n = 7,\ \ f(\mathbf{x}^*) \approx -0.0188;$
# $n = 8,\ \ f(\mathbf{x}^*) \approx -0.0170;$
# $n = 9,\ \ f(\mathbf{x}^*) \approx -0.0156;$
# Description
# Similar to the Sphere function but very flat.  Poses difficulties
# for conventional optimization methods because of its flatness which
# makes them take very small steps towards the minimum.  Hence
# minimization process takes prohibitively many steps to complete.
```

C.8 SinSin Function



(a) 3D plot

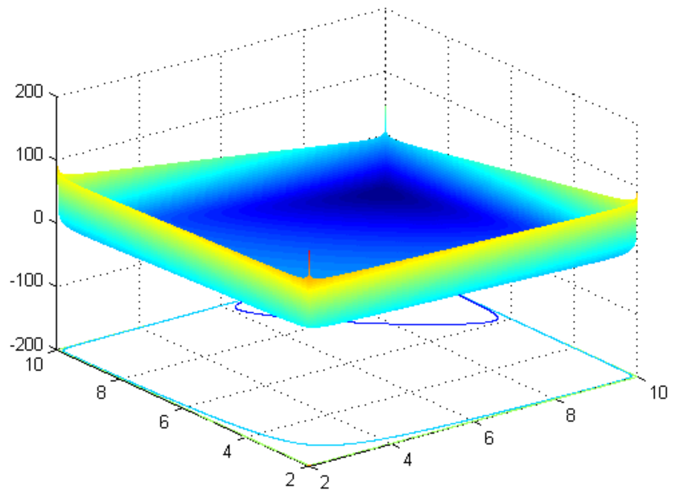


(b) Contour lines plot

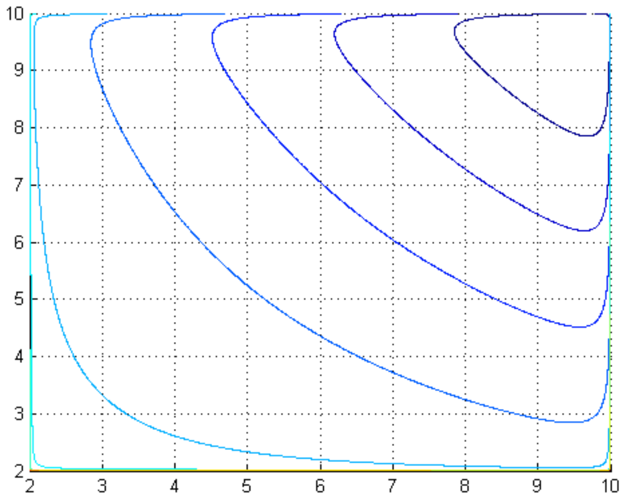
Figure C.9: *SinSin function*

```
# **Definition**
# $f(\mathbf{x}) = \prod_{i=1}^{v} \sin(10x_i)$
# **Search domain**
# $x_i \in [-1, 1],\ \ i = 1, 2, \ldots, v$
# **Local minima**
# Several, same as global
# **Global minimum**
# Several
# $\mathbf{x}^* = (-\pi/20 + 2\pi k_1, \ldots, -\pi/20 + 2\pi k_n),\ \ k_i \in \mathbb{Z}, i = 1, 2, \ldots, n$
# $f(\mathbf{x}^*) = -1$
# **Description**
# Highly oscillatory function with many global minima.  Poses significant
# difficulty for the rigorous global minimizers due to a large number of
# the regions of interest which cannot be eliminated from
# the consideration during the search process.
```

C.9 Paviani's Function



(a) 3D plot



(b) Contour lines plot

Figure C.10: *Paviani's function*

```
# Definition
# $f(\mathbf{x}) = \sum_{i=1}^{v} (\ln(x_i - 2))^2 + (\ln(10 - x_i))^2 - \prod_{i=1}^{v} x_i^{2/v}$
# Search domain
# $x_i \in [2.001, 9.999], \ \ i = 1, 2, \ldots, v$
# Local minima
# One, same as global
# Global minimum
# $n = 2, \ \mathbf{x}^* \approx (9.72, 9.72); \ f(\mathbf{x}^*) \approx -82.883;$
# $n = 3, \ \mathbf{x}^* \approx (9.64, 9.64, 9.64); \ f(\mathbf{x}^*) \approx -77.396;$
# $n = 4, \ \mathbf{x}^* \approx (9.58, \ldots, 9.58); \ f(\mathbf{x}^*) \approx -72.357;$
# $n = 5, \ \mathbf{x}^* \approx (9.53, \ldots, 9.53); \ f(\mathbf{x}^*) \approx -67.591;$
# $n = 6, \ \mathbf{x}^* \approx (9.49, \ldots, 9.49); \ f(\mathbf{x}^*) \approx -63.013;$
# $n = 7, \ \mathbf{x}^* \approx (9.45, \ldots, 9.45); \ f(\mathbf{x}^*) \approx -58.57;$
# $n = 8, \ \mathbf{x}^* \approx (9.40, \ldots, 9.40); \ f(\mathbf{x}^*) \approx -54.23;$
# $n = 9, \ \mathbf{x}^* \approx (9.38, \ldots, 9.38); \ f(\mathbf{x}^*) \approx -49.97;$
# Description
# Is a challenge for interval methods because even though it is flat, it
# has a high level of dependencies.  This property makes it hard for the
# interval methods to reject boxes until they are very finely split [27].
```

APPENDIX D

Test Problems in Constrained Optimization with Evolutionary Algorithms

The set of test problems for single-objective constrained optimization was suggested as a standard benchmark for EAs by Michalewicz [135] and later adopted to test the performance of all newly developed methods by the EA community [41, 52, 133, 149, 169]. This testbench includes various synthetic problems (G01-G13) that expose different properties of the constraints, the feasible set, and the sought minimum. Several real-life design problems originally solved by constrained EAs (vess, tens) are also used. Problems in this appendix are listed using the conventions from the optimization problem formulation (1.3.1), (1.3.2), (1.3.3), (1.3.4), (1.3.5). The search space S is given as a box, i.e. set of ranges for x_i, $i = 1, \ldots, v$. The values for global minima are listed if known. The best known values are given where the true minima are not known.

A rough empirical classification of problem difficulty and estimates for the

$$\rho = \frac{|F|}{|S|} \cdot 100$$

parameter are taken from [128]. Note that generally the most important factors that increase the difficulty of a constrained optimization problem are the presence of at least one nonlinear inequality and a high dimensionality. Note also, that even though any feasible set where one of the constraints is an equality theoretically has a measure zero, the value of the parameter ρ obtained by a finite sampling of the search space to find the feasible points can be non-zero. For practical purposes such an estimate is more useful than a purely theoretical measure. First, because, for the general set of constraints precise determination of F can be extremely difficult. Second, for practical purposes set F that consists of a single point is harder to treat than F that consists of a single line, which is, in turn harder to treat than F that consists of a plane. The small deviations of ρ from the theoretical zero obtained by sampling allow us to make such a distinction (even though only approximately). Values of ρ in the problem descriptions are obtained by sampling the search space S by $1,000,000$ random points.

DIFFICULT

$\rho \approx 0.0003$

$v = 13$

$n = 9$ (9 linear inequalities; h_1, h_2, h_3, h_4, h_5, h_6 active)

quadratic objective function

$f(\mathbf{x}) = 5\sum_{i=1}^{4}(x_i - x_i^2) - \sum_{i=5}^{13} x_i$

$h_1(\mathbf{x}) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0$
$h_2(\mathbf{x}) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0$
$h_3(\mathbf{x}) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0$
$h_4(\mathbf{x}) = -2x_4 - x_5 + x_{10} \leq 0$
$h_5(\mathbf{x}) = -2x_6 - x_7 + x_{11} \leq 0$
$h_6(\mathbf{x}) = -2x_8 - x_9 + x_{12} \leq 0$
$h_7(\mathbf{x}) = -8x_1 + x_{10} \leq 0$
$h_8(\mathbf{x}) = -8x_2 + x_{11} \leq 0$
$h_9(\mathbf{x}) = -8x_3 + x_{12} \leq 0$

$x_i \in [0, 1]$, $i = 1, \ldots, 9$
$x_i \in [0, 100]$, $i = 10, \ldots, 12$
$x_{13} \in [0, 1]$

$\mathbf{x}^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$
$f(\mathbf{x}^*) = -15$

Figure D.1: *g01 Test problem*

DIFFICULT
$\rho \approx 99.9973$
$v = 20$
$n = 2$ (1 linear inequality, 1 nonlinear inequality; h_1 almost active (-10^{-8}))
nonlinear objective function

$$f(\mathbf{x}) = -\left|\left(\sum_{i=1}^{v}\cos^4(x_i) - 2\prod_{i=1}^{v}\cos^2(x_i)\right)\left(\sum_{i=1}^{v} i x_i^2\right)^{-0.5}\right|$$

$$h_1(\mathbf{x}) = 0.75 - \prod_{i=1}^{v} x_i \leq 0$$
$$h_2(\mathbf{x}) = \sum_{i=1}^{v} x_i - 7.5v \leq 0$$

$x_i \in [0, 10]$, $i = 1, \ldots, v$

best known $f(\mathbf{x}^*) = -0.803619$

Figure D.2: *g02 Test problem (best known value from [163])*

DIFFICULT
$\rho \approx 0.0026$
$v = 10$
$n = 1$ (1 nonlinear equality; g_1 active)
nonlinear objective function

$$f(\mathbf{x}) = -v^{2/v}\prod_{i=1}^{v} x_i$$

$$g_1(\mathbf{x}) = \sum_{i=1}^{v} x_i^2 - 1 = 0$$

$x_i \in [0, 10]$, $i = 1, \ldots, v$

$\mathbf{x}^* = 1/\sqrt{v}(\pm 1, \pm 1, \ldots, \pm 1)$, any combination of ± 1's such that their product is positive
$f(\mathbf{x}^*) = -1$

Figure D.3: *g03 Test problem*

AVERAGE

$\rho \approx 27.0079$

$v = 5$

$n = 6$ (4 linear inequalities, 2 nonlinear inequalities; h_1, h_6 active)

quadratic objective function

$$f(\mathbf{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$

$$h_1(\mathbf{x}) = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \le 0$$
$$h_2(\mathbf{x}) = -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \le 0$$
$$h_3(\mathbf{x}) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \le 0$$
$$h_4(\mathbf{x}) = -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \le 0$$
$$h_5(\mathbf{x}) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \le 0$$
$$h_6(\mathbf{x}) = -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \le 0$$

$x_1 \in [78, 102]$

$x_2 \in [33, 45]$

$x_i \in [27, 45]$, $i = 3, \ldots, 5$

$\mathbf{x}^* = (78, 33, 29.995256025682, 45, 36.775812905788)$

$f(\mathbf{x}^*) = -30665.539$

Figure D.4: *g04 Test problem*

VERY DIFFICULT

$\rho \approx 0.0000$

$v = 4$

$n = 5$ (2 linear inequalities, 3 nonlinear equalities; g_1, g_2, g_3 are active)

nonlinear objective function

$f(\mathbf{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3$

$h_1(\mathbf{x}) = -x_4 + x_3 - 0.55 \leq 0$

$h_2(\mathbf{x}) = -x_3 + x_4 - 0.55 \leq 0$

$g_1(\mathbf{x}) = 1000\sin(-x_3 - 0.25) + 1000\sin(-x4 - 0.25) + 894.8 - x_1 = 0$

$g_2(\mathbf{x}) = 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0$

$g_3(\mathbf{x}) = 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0$

$x_i \in [0, 1200]$, $i = 1, 2$

$x_i \in [-0.55, 0.55]$, $i = 3, 4$

best known $\mathbf{x}^* = (679.9453, 1026.067, 0.1188764, -0.3962336)$

$f(\mathbf{x}^*) = 5126.4981$

Figure D.5: *g05 Test problem*

`AVERAGE`

$\rho \approx 0.0057$

$v = 2$

$n = 2$ `(2 nonlinear inequalities;` h_1`,` h_2 `active)`

`nonlinear objective function`

$$f(\mathbf{x}) = (x_1 - 10)^3 + (x_2 - 20)^3$$

$$h_1(\mathbf{x}) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0$$
$$h_2(\mathbf{x}) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0$$

$x_1 \in [13, 100]$

$x_2 \in [0, 100]$

$\mathbf{x}^* = (14.095, 0.84296)$

$f(\mathbf{x}^*) = -6961.81388$

Figure D.6: *g06 Test problem*

```
AVERAGE
```
$\rho \approx 0.0000$

$v = 10$

$n = 8$ (3 linear inequalities, 5 nonlinear inequalities; h_1, h_2, h_3, h_4, h_5, h_6 active)

quadratic objective function

$$\begin{aligned} f(\mathbf{x}) = {} & x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3-10)^2 + 4(x_4-5)^2 + (x_5-3)^2 \\ & +2(x_6-1)^2 + 5x_7^2 + 7(x_8-11)^2 + 2(x_9-10)^2 + (x_{10}-7)^2 + 45 \end{aligned}$$

$$h_1(\mathbf{x}) = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \le 0$$
$$h_2(\mathbf{x}) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \le 0$$
$$h_3(\mathbf{x}) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \le 0$$
$$h_4(\mathbf{x}) = 3(x_1-2)^2 + 4(x_2-3)^2 + 2x_3^2 - 7x_4 - 120 \le 0$$
$$h_5(\mathbf{x}) = 5x_1^2 + 8x_2 + (x_3-6)^2 - 2x_4 - 40 \le 0$$
$$h_6(\mathbf{x}) = x_1^2 + 2(x_2-2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \le 0$$
$$h_7(\mathbf{x}) = 0.5(x_1-8)^2 + 2(x_2-4)^2 + 3x_5^2 - x_6 - 30 \le 0$$
$$h_8(\mathbf{x}) = -3x_1 + 6x_2 + 12(x_9-8)^2 - 7x_{10} \le 0$$

$x_i \in [-10, 10]$, $i = 1, \ldots, 10$

$$\begin{aligned} \mathbf{x}^* = {} & (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, \\ & 9.828726, 8.280092, 8.375927) \end{aligned}$$
$f(\mathbf{x}^*) = 24.3062091$

Figure D.7: *g07 Test problem*

EASY

$\rho \approx 0.8581$

$v = 2$

$n = 2$ (2 nonlinear inequalities)

nonlinear objective function

$$f(\mathbf{x}) = -\sin^3(2\pi x_1)\sin(2\pi x_2)\big(x_1^3(x_1+x_2)\big)^{-1}$$

$$h_1(\mathbf{x}) = x_1^2 - x_2 + 1 \le 0$$
$$h_2(\mathbf{x}) = 1 - x_1 + (x_2-4)^2 \le 0$$

$x_i \in [0, 10]$, $i = 1, 2$

$\mathbf{x}^* = (1.2279713, 4.2453733)$

$f(\mathbf{x}^*) = -0.095825$

Figure D.8: *g08 Test problem*

AVERAGE

$\rho \approx 0.5199$

$v = 7$

$n = 4$ (4 nonlinear inequalities; h_1, h_4 active)

nonlinear objective function

$$f(\mathbf{x}) = (x_1-10)^2 + 5(x_2-12)^2 + x_3^4 + 3(x_4-11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 \\ -10x_6 - 8x_7$$

$$h_1(\mathbf{x}) = -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \le 0$$
$$h_2(\mathbf{x}) = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \le 0$$
$$h_3(\mathbf{x}) = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \le 0$$
$$h_4(\mathbf{x}) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \le 0$$

$x_i \in [-10, 10]$, $i = 1, \ldots, 7$

$\mathbf{x}^* = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)$

$f(\mathbf{x}^*) = 680.6300573$

Figure D.9: *g09 Test problem*

DIFFICULT

$\rho \approx 0.0020$

$v = 8$

$n = 6$ (3 linear inequalities, 3 nonlinear inequalities; h_1, h_2, h_3 active)

linear objective function

$f(\mathbf{x}) = x_1 + x_2 + x_3$

$h_1(\mathbf{x}) = -1 + 0.0025(x_4 + x_6) \leq 0$

$h_2(\mathbf{x}) = -1 + 0.0025(x_5 + x_7 - x_4) \leq 0$

$h_3(\mathbf{x}) = -1 + 0.01(x_8 - x_5) \leq 0$

$h_4(\mathbf{x}) = -x_1 x_6 + 833.33252 x_4 + 100 x_1 - 83333.333 \leq 0$

$h_5(\mathbf{x}) = -x_2 x_7 + 1250 x_5 + x_2 x_4 - 1250 x_4 \leq 0$

$h_6(\mathbf{x}) = -x_3 x_8 + 1250000 + x_3 x_5 - 2500 x_5 \leq 0$

$x_1 \in [100, 10000]$

$x_i \in [1000, 10000]$, $i = 2, \ldots, 3$

$x_i \in [10, 1000]$, $i = 4, \ldots, 8$

$\mathbf{x}^* = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979)$

$f(\mathbf{x}^*) = 7049.3307$

Figure D.10: *g10 Test problem*

EASY
$\rho \approx 0.0973$
$v = 2$
$n = 1$ (1 nonlinear equality; g_1 active)
linear objective function

$f(\mathbf{x}) = x_1^2 + (x_2 - 1)^2$

$g_1(\mathbf{x}) = x_2 - x_1^2 = 0$

$x_i \in [-1, 1]$, $i = 1, 2$

$\mathbf{x}^* = (\pm 1/\sqrt{2}, 1/2)$
$f(\mathbf{x}^*) = 0.75$

Figure D.11: *g11 Test problem*

EASY
$\rho \approx 4.7697$
$v = 3$
$n = 1$ (9^3 nonlinear inequalities joined by logical OR instead of AND, disjoint F)
quadratic objective function

$f(\mathbf{x}) = -100^{-1}(100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2)$

$h_i(\mathbf{x}) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0.0625 \leq 0$, $i = 1, \ldots, 9^3$,
$p, q, r = 1, \ldots, 9$
$\mathbf{x}$ is feasible if it satisfies one of h_i

$x_i \in [0, 10]$, $i = 1, 2, 3$

$\mathbf{x}^* = (5, 5, 5)$
$f(\mathbf{x}^*) = -1$

Figure D.12: *g12 Test problem*

VERY DIFFICULT

$\rho \approx 0.0000$

$v = 5$

$n = 3$ (1 linear equality, 2 nonlinear equalities; $g1$, $g2$, $g3$ active)

nonlinear objective function

$$f(\mathbf{x}) = e^{x_1 x_2 x_3 x_4 x_5}$$

$$g_1(\mathbf{x}) = \sum_{i=1}^{5} x_i^2 - 10 = 0$$
$$g_2(\mathbf{x}) = x_2 x_3 - 5x_4 x_5 = 0$$
$$g_3(\mathbf{x}) = x_1^3 + x_2^3 + 1 = 0$$

$x_i \in [-2.3, 2.3]$, $i = 1, 2$
$x_i \in [-3.2, 3.2]$, $i = 3, 4, 5$

$$\mathbf{x}^* = (-1.717143, 1.595709, 1.827247, -0.7636413, -0.763645)$$
$$f(\mathbf{x}^*) = 0.0539498$$

Figure D.13: *g13 Test problem*

AVERAGE

$\rho \approx 39.6762$

$v = 4$

$n = 4$ (3 linear inequalities, 1 nonlinear inequality)

quadratic objective function

$$f(\mathbf{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$$

$$h_1(\mathbf{x}) = -x_1 + 0.0193x_3 \leq 0$$
$$h_2(\mathbf{x}) = -x_2 + 0.00954x_3 \leq 0$$
$$h_3(\mathbf{x}) = -\pi x_3^2x_4 - 4/3\pi x_3^3 + 1296000 \leq 0$$
$$h_4(\mathbf{x}) = x_4 - 240 \leq 0$$

$x_i \in [1, 99]$, $i = 1, 2$

$x_i \in [10, 200]$, $i = 3, 4$

best known: $f(\mathbf{x}^*) = 6059.946341$

Figure D.14: *Design of a Pressure Vessel (vess) [100] (best known value from [41])*

EASY

$\rho \approx 0.7537$

$v = 3$

$n = 4$ (1 linear inequality, 3 nonlinear inequalities)

quadratic objective function

$$f(\mathbf{x}) = (x_3 + 2)x_2x_1^2$$

$$h_1(\mathbf{x}) = 1 - x_2^3x_3(71785x_1^4)^{-1} \leq 0$$
$$h_2(\mathbf{x}) = (4x_2^2 - x_1x_2)(12566(x_2x_1^3 - x_1^4))^{-1} + (5108x_1^2)^{-1} - 1 \leq 0$$
$$h_3(\mathbf{x}) = 1 - 140.45x_1x_2^{-2}x_3^{-1} \leq 0$$
$$h_4(\mathbf{x}) = (x_2 + x_1)1.5^{-1} - 1 \leq 0$$

$x_1 \in [0.05, 2]$

$x_2 \in [0.25, 1.3]$

$x_3 \in [2, 15]$

best known: $f(\mathbf{x}^*) = 0.012681$

Figure D.15: *Design of a Tension/Compression Spring (tens) [9] (best known value from [41])*

BIBLIOGRAPHY

[1] C. Albright, V. Barger, J. Beacom, S. Brice, J. J. Gomez-Cadenas, M. Goodman, D. Harris, P. Huber, A. Jansson, M. Lindner, O. Mena, P. Rapidis, K. Whisnant, W. Winter, The Neutrino Factory, and Muon Collider Collaboration. The neutrino factory and beta beam experiments and development. Technical Report BNL-72369-2004, FNAL-TM-2259, LBNL-55478, BNL, FNAL, LBNL, July 2004.

[2] Carl H. et al. Albright. Physics at a neutrino factory. Technical Report FERMILAB-FN-0692, Fermi National Accelerator Laboratory, 2000.

[3] Mohammad M. Alsharoa et al. Recent progress in neutrino factory and muon collider research within the muon collaboration. *Phys. Rev. ST Accel. Beams*, 6:081001, 2003.

[4] Martin Berz an Kyoko Makino. Higher order multivariate automatic differentiation and validated computation of remainder bounds. *WSEAS Transactions on Mathematics*, 3(Issue 1):37, January 2004. see http://www.wseas.org.

[5] M. Apollonio et al. Oscillation physics with a neutrino factory. ((g)) ((u)). *CERN Yellow Report on the Neutrino Factory*, 2002.

[6] David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2006.

[7] Mordecai Avriel. *Nonlinear Programming: Analysis and Methods.* Dover Publications, 2003.

[8] Helio J. C. Barbosa. A Coevolutionary Genetic Algorithm for Constrained Optimization. In *Proceedings of the Congress on Evolutionary Computation 1999 (CEC'99)*, volume 3, pages 1605–1611, Piscataway, New Jersey, July 1999. IEEE Service Center.

[9] Ashok Dhondu Belegundu. *A Study of Mathematical Programming Methods for Structural Optimization.* PhD thesis, University of Iowa, Iowa, USA, 1982.

[10] J. S. Berg, S. A. Bogacz, S. Caspi, J. Cobb, R. C. Fernow, J. C. Gallardo, S. Kahn, H. Kirk, D. Neuffer, R. Palmer, K. Paul, H. Witte, and M. Zisman. Cost-effective design for a neutrino factory. *Phys. Rev. ST Accel. Beams*, 9(1):011001, Jan 2006.

[11] F. Bernelli-Zazzera, M. Berz, M. Lavagna, R. Armellin, P. Di Lizia, and F. Topputo. Global trajectory optimisation: Can we prune the solution space when considering deep space maneuvers? Final Report 06/4101, ARIADNA Study, December 2007.

[12] R. H. Berry and G. D. Smith. Using a genetic algorithm to investigate taxation induced interactions in capital budgeting. In *Proc. of the International Conference on Neural Networks and Genetic Algorithms*, pages 567–574, 1993.

[13] M. Berz. Differential algebraic description of beam dynamics to very high orders. *Particle Accelerators*, 24:109–124, 1989.

[14] M. Berz. Computational aspects of design and simulation: COSY INFINITY. Technical Report 740, National Superconducting Cyclotron Laboratory, Michigan State University, East Lansing, MI 48824, 1990.

[15] M. Berz. *High-Order Computation and Normal Form Analysis of Repetitive Systems, in: M. Month (Ed), Physics of Particle Accelerators*, volume 249, pages 456–489. American Institute of Physics Publishing, New York, 1991.

[16] M. Berz. Differential algebraic formulation of normal form theory. In S. Martin M. Berz and K. Ziegler, editors, *M. Berz, S. Martin and K. Ziegler (Eds.), Proc. Nonlinear Effects in Accelerators*, page 77, London, 1992. Institute of Physics Publishing.

[17] M. Berz. Modern map methods for charged particle optics. *Nuclear Instruments and Methods*, 363:100, 1995.

[18] M. Berz. *Modern Map Methods in Particle Beam Physics.* Academic Press, San Diego, 1999. Also available at http://bt.pa.msu.edu/pub.

[19] M. Berz, B. Erdélyi, and K. Makino. Fringe field effects in small rings of large acceptance. *Physical Review ST-AB*, 3:124001, 2000.

[20] M. Berz and G. Hoffstätter. Exact bounds of the long term stability of weakly nonlinear systems applied to the design of large storage rings. *Interval Computations*, 2:68–89, 1994.

[21] M. Berz and K. Makino. Normal form methods and optimization for nonlinear properties of cooling channels - part I. In *Proc. of the APS/DPF/DPB Summer Study on the Future of Particle Physics (Snowmass 2001)*, 2002. number T504.

[22] M. Berz and K. Makino. COSY INFINITY Version 9.0 beam physics manual. Technical Report MSUHEP-060804, Department of Physics and Astronomy, Michigan State University, East Lansing, MI 48824, 2006. see also http://cosyinfinity.org.

[23] M. Berz and K. Makino. COSY INFINITY Version 9.0 programmer's manual. Technical Report MSUHEP-060803, Department of Physics and Astronomy, Michigan State University, East Lansing, MI 48824, 2006. see also http://cosyinfinity.org.

[24] M. Berz, K. Makino, and B. Erdélyi. Fringe field effects in muon rings. *AIP CP*, 530:38–47, 2000.

[25] M. Berz, K. Makino, and C. J. Johnstone. Propagation of a large-emittance muon beam through a straight, quadrupole-based precooling channel. In *Neutrino Factories and Superbeams*, volume 721, page 413. AIP Conference Proceedings, 2004.

[26] M. Berz, K. Makino, and Y.-K. Kim. Long-term stability of the Tevatron by verified global optimization. *Nuclear Instruments and Methods*, 558:1–10, 2005.

[27] Martin Berz. Private communication.

[28] George Bilchev and Ian C. Parmee. Constrained and multi-modal optimisation with an ant colony search model. In C. Parmee and M. J. Denham, editors, *Proceedings of 2nd International Conference on Adaptive Computing in Engineering Desing and Control*. University of Plymouth, Plymouth, UK, March 1996.

[29] Donato Bini, Robert T Jantzen, and Andrea Merloni. Geometric interpretation of the frenet-serret frame description of circular orbits in stationary axisymmetric spacetimes. *Classical and Quantum Gravity*, 16:1333–1348, 1999.

[30] A. Blondel and et al. (ed.). Ecfa/cern studies of a european neutrino factory complex. Technical Report CERN-2004-002, CERN, 2004.

[31] M. Born and E. Wolf. *Principles of Optics*. Pergamon Press, Oxford, 6 th edition, 1980.

[32] Stephen Brooks. Muon1 Distributed Particle Accelerator Design. on-line.

[33] J. Burguet-Castell, M. B. Gavela, J. J. Gomez-Cadenas, P. Hernandez, and Olga Mena. Superbeams plus neutrino factory: The golden path to leptonic cp violation. *Nucl. Phys.*, B646:301–320, 2002.

[34] Susan E. Carlson, R. Shonkwiler, S. Babar, and M. Aral. Annealing a genetic algorithm over constraints. on-line.

[35] G. Casadei, A. Palareti, and G. Proli. Classifier system in traffic management. In *Proc. of the International Conference on Neural Networks and Genetic Algorithms*, pages 620–627, 1993.

[36] Weng Tat Chan, T.F. Fwa, and Kh. Zahidul Hoque. Constraint Handling Methods in Pavement Maintenance Programming. *Transportation Research Part C - Emerging Technologies*, 9(3):175–190, June 2001.

[37] Amy B. Chan-Hilton and Teresa B. Culver. Constraint-handling methods for optimal groundwater remediation design by genetic algorithms. In *Proc. of IEEE International Conference on Systems, Man, and Cybernetics, vol.4*, pages 3937–3942. IEEE, 1998.

[38] L. M. Chapin, J. Hoefkens, and M. Berz. The COSY language independent architecture: Porting COSY source files. *Insitute of Physics CS*, 175:37–45, 2004.

[39] D. Cline and D. Neuffer. A muon storage ring for neutrino oscillation experiments. In *AIP Conf. Proc. 68*, page 846, 1980.

[40] Carlos A. Coello Coello and Ricardo Landa Becerra. Constrained Optimization Using an Evolutionary Programming-Based Cultural Algorithm. In I.C. Parmee, editor, *Proceedings of the Fifth International Conference on Adaptive Computing in Design and Manufacture (ACDM'2002)*, volume 5, pages 317–328, University of Exeter, Devon, UK, April 2002. Springer-Verlag.

[41] Carlos A. Coello Coello and Efrén Mezura-Montes. Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatics*, 16(3):193–203, July 2002.

[42] Carlos A. Coello Coello. A Survey of Constraint Handling Techniques used with Evolutionary Algorithms. Technical Report Lania-RI-99-04, Laboratorio Nacional de Informática Avanzada, Xalapa, Veracruz, México, 1999.

[43] Carlos A. Coello Coello. Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. *Engineering Optimization*, 32,(3,):275–308,, 2000.

[44] Carlos A. Coello Coello and Nareli Cruz Cortés. Use of Emulations of the Immune System to Handle Constraints in Evolutionary Algorithms. In Cihan H. Dagli, Anna L. Buczak, Joydeep Ghosh, Mark J. Embrechts, Okan Erson, and Stephen Kercel, editors, *Intelligent Engineering Systems through Artificial Neural Networks (ANNIE'2001)*, volume 11, pages 141–146, St. Louis Missouri, USA, November 2001. ASME Press.

[45] David W. Coit and Alice E. Smith. Penalty guided genetic search for reliability design optimization. *Computers and Industrial Engineering*, 30(4):895–904, September 1996. Special Issue on Genetic Algorithms.

[46] Muon Collider Collaboration. $\mu^+\mu^-$ collider: A feasibility study. Technical Report BNL-52503, Fermilab-Conf-96/092, LBNL-38946, Muon Collider Collaboration, jul 1996. pres. at the Snowmass'96 workshop.

[47] M. Conte and W. W. MacKay. *An Introduction to the Physics of Particle Accelerators*. World Scientific, Singapore, 1991.

[48] COSY Infinity home page. on-line. `http://www.cosyinfinity.org/`.

[49] Courant and Snyder. Theory of the alternating gradient synchrotron. *Annals of Physics*, 3(1), 1958.

[50] R. Cucchiara. Analysis and comparison of different genetic models for the clustering problem in image analysis. In *Proc. of the International Conference on Neural Networks and Genetic Algorithms*, pages 423–427, 1993.

[51] Charles Darwin. *The Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. London: John Murray, 6 edition, 1872.

[52] Kalyanmoy Deb. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation*, 7(3):205–230, 1999.

[53] D. Deugo and F. Oppacher. Achieving self-stabilization in a distributed system using evolutionary strategies. In *Proc. of the International Conference on Neural Networks and Genetic Algorithms*, pages 400–407, 1993.

[54] A. J. Dragt and J. M. Finn. Normal form for mirror machine Hamiltonians. *Journal of Mathematical Physics*, 20(12):2649, 1979.

[55] Lin Du and John Bigham. Constrained Coverage Optimisation for Mobile Cellular Networks. In Günther Raidl et al., editor, *Applications of Evolutionary Computing. Evoworkshops 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM*, pages 199–210, Essex, UK, April 2003. Springer Verlag. Lecture Notes in Computer Science Vol. 2611.

[56] Eldon Hansen (Ed.) and G. William Walster (Ed.). *Global Optimization Using Interval Analysis*. Pure and Applied Mathematics. A Dekker Series of Monographs and Textbooks. CRC, 2nd edition, 2003.

[57] A. E. Eiben, P.-E. Raué, and Zs. Ruttkay. GA-easy and GA-hard constraint satisfaction problems. In M. Meyer, editor, *Proceedings of the ECAI'94 Workshop on Constraint Processing*, pages 267–284. Springer-Verlag, 1995.

[58] A. E. Eiben and J. K. van der Hauw. Adaptive penalties for evolutionary graph coloring. In J.-K Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution'97*, volume 1363 of *Lecture Notes in Computer Science*, pages 95–106. Springer-Verlag, Berlin, 1998.

[59] Fuat Erbatur, Oğŭzhan Hasançebi, İlker Tütüncü, and Hakan Kiliç. Optimal design of planar and space structures with genetic algorithms. *Computers and Structures*, 75(2):209–224, March 2000.

[60] D. Errede, K. Makino, M. Berz, C. J. Johnstone, and A. van Ginneken. Stochastic processes in muon ionization cooling. *Nuclear Instruments and Methods A*, 519:466–471, 2004.

[61] Christodoulos A. Floudas et al. *Handbook of Test Problem in Local and Global Optimization*, volume 33 of *Nonconvex Optimization and Its Applications.* Springer, 1999.

[62] J.S. Berg et al. Iss scoping study: Accelerator design concept for future neutrino factories. Technical Report RAL-TR-2007-23, RAL, 2007.

[63] W.-M. Yao et al. Neutrino mass, mixing, and flavor change. *Journal of Physics, G*, 33(1), 2006.

[64] Neal C. Evans and David L. Shealy. Design of a gradient-index beam shaping system via a genetic algorithm optimization method. In Fred M. Dickey and Scott C. Holswade, editors, *Proceedings of SPIE*, volume 4095, pages 26–39, October 2000.

[65] Bibliography on Evolutionary Computation. on-line. `http://www.fmi.uni-stuttgart.de/fk/evolalg/eabib.html`.

[66] R.C. Fernow. Icool, a simulation code for ionization cooling of muon beams. In *Proc. 1999 Particle Accelerator Conference*, page 3020, New York, 1999. see http://pubweb.bnl.gov/people/fernow/.

[67] R.C. Fernow. Physics analysis performed by ecalc9. Neutrino Factory and Muon Collider Notes MUC-NOTE-COOL_THEORY-280, Brookhaven National Laboratory, 2003. see http://www-mucool.fnal.gov/notes/notes.html.

[68] R.C. Fernow. *ICOOL User's Guide.* Brookhaven National Laboratory, June 2005. see http://pubweb.bnl.gov/people/fernow/icool/readme.html.

[69] R.C. Fernow. Recent developments on the muon-facility design code icool. In *Proc. 2005 Particle Accelerator Conference*, page 2651, Knoxville, Texas, 2005.

[70] R.C. Fernow. Scattering in icool. Neutrino Factory/Muon Collider Notes MUC-NOTE-COOL-THEORY-0336, Brookhaven National Laboratory, April 2006.

[71] R.C. Fernow and J.C. Gallardo. Examination of the us study 2a neutrino factory front-end design. Neutrino Factory/Muon Collider Notes MUC-NOTE-COOL-THEORY-0331, Fermi National Accelerator Laboratory, January 2006.

[72] A.V. Fiacco and G.P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimiaztion Techniques.* Wiley, New York, 1968.

[73] Jeffrey Friedl. *Mastering Regular Expressions.* O'Reilly Media, Inc, 3rd edition, August 2006.

[74] J.C. Gallardo, J.S. Berg, R.C. Fernow, H. Kirk, R.B. Palmer, D. Neuffer, and K. Paul. New and efficient neutrino factory front-end design. Neutrino Factory/Muon Collider Notes MUC-NOTE-COOL-THEORY-0316, Fermi National Accelerator Laboratory, 2005.

[75] S. Geer. Neutrino beams from muon storage rings: Characteristics and physics potential. *Phys. Rev.*, D57:6989–6997, 1998.

[76] A. Geraci, T. Barlow, M. Portillo, J. Nolen, K. Shepard, M. Berz, and K. Makino. High-order maps with acceleration for optimization of electrostatic and radio-frequency ion-optical elements. *Review of Scientific Instruments*, 73,9:3174–3180, 2002.

[77] Philip E. Gill and Walter Murray. Algorithms for the solution of the nonlinear least-squares problem. *SIAM Journal on Numerical Analysis*, 15(5):977–992, 1978.

[78] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional, 1st ed. edition, 1989.

[79] David E. Goldberg. *The Design of Innovation. Lessons from and for Competent Genetic Algorithms.* Springer, 1st ed. edition, June 2002.

[80] J. Gottlieb, E. Marchiori, and C. Rossi. Evolutionary Algorithms for the Satisfiability Problem. *Evolutionary Computation*, 10(1):35–50, 2002.

[81] NuFactJ Working Group. A feasibility study of a neutrino factory in japan. Technical Report v. 1.0, NufactJ Working Group, May 2001.

[82] Schwefel H.-P. *Evolution and optimium seeking.* Wiley, New York, 1991.

[83] J. Rokne H. Ratschek. *New Computer Methods for Global Optimization.* Ellis Horwood Limited, Chichester, England, 1988.

[84] P. Hajela and J. Lee. Constrained genetic search via schema adaptation. an immune network solution. *Structural and Multidisciplinary Optimization*, 12(1):11–15, August 1996.

[85] Scot Hillier and Daniel Mezick. *Programming Active Server Pages.* Microsoft Press, 1997.

[86] N. Holtkamp, D. Finley, and (eds.). A feasibility study of a neutrino source based on a muon storage ring. Technical Report FERMILAB-PUB-00-108-E, Fermi National Accelerator Laboratory, April 2000. see `http://library.fnal.gov/archive/test-preprint/fermilab-pub-00-108-e.shtml`.

[87] A. Homaifar, S. H.-Y. Lai, and X. Qi. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–254, 1994.

[88] J. Horn, N. Nafpliotis, and D.E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence*, pages 82–87, Orlando, FL, USA, June 1994.

[89] S. Humphries. *Charged Particle Beams*. Wiley, New York, 1990.

[90] Philip Husbands, Frank Mill, and Stephen Warrington. Genetic algorithms, production plan optimisation and scheduling. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 80–84, 1991.

[91] L. Ingber. Adaptive simulated annealing (asa): Lessons learned. *Control and Cybernetics*, 25(1):33–34, 1996.

[92] Nelder J.A. and Mead R. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[93] J. D. Jackson. *Classical Electrodynamics*. Wiley, New York, 1975.

[94] Fernando Jiménes and José L. Verdegay. Evolutionary techniques for constrained optimiation problems. In *1th European Congress on Intelligent Techniques and Soft Computing*, 1999.

[95] C. J. Johnstone, M. Berz, D. Errede, and K. Makino. Optimization and beam control in large-emittance accelerators: Neutrino factories. In *2003 International Conference Physics and Control*, pages 964–973. IEEE, 2003.

[96] C. J. Johnstone, M. Berz, D. Errede, and K. Makino. Muon beam ionization cooling in a linear quadrupole channel. *Nuclear Instruments and Methods A*, 519:472–482, 2004.

[97] Carol Johnstone. Private communication.

[98] Jeffrey A. Joines and Christopher R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with ga's. In *Proceedings of the First IEEE Conference on Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence*, pages 579–584. IEEE Press, 1994.

[99] George Katodrytis. Genetic algorithms in design: Theory and application. In *Proc. of Computer Graphics, Imaging and Vision: New Trends, 2005*, pages 426–430, 2005.

[100] S. Kazarlis and V. Petridis. Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the 5th Parallel Problem Solving from Nature (PPSN V)*, pages 211–220, Heidelberg, Germany, September 1998. Springer-Verlag.

[101] R. B. Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer, 1996.

[102] Jouni A. Lampinenk Kenneth V. Price, Rainer M. Storn. *Differential Evolution: A Practical Approach to Global Optimization.* Natural Computing. Springer, 1st ed. edition, December 2005.

[103] Y.-K. Kim. Private communication.

[104] Y.-K. Kim and M. Berz. Parallel constructs in COSY INFINITY. MSU High Energy Phyisics Preprint MSUHEP-060805, Michigan State University, 2006.

[105] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[106] D. G. Koshkarev. Proposal for a decay ring to produce intense secondary particle beams at the sps.

[107] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Complex Adaptive Systems. The MIT Press, December 1992.

[108] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[109] K.Paul and C. Johnstone. Optimizing the pion capture and decay channel. Neutrino Factory/Muon Collider Notes MUC-NOTE-COOL-THEORY-0280, Fermi National Accelerator Laboratory, 2004.

[110] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of 2nd Berkeley Symposium, Berkeley*, pages 481–492. University of California Press., 1951.

[111] M. Locatelli. A note on the griewank test function. *Journal of Global Optimization*, 25(2):169–174, February 2003. 2003.

[112] Kai-Yew Lum, Pierre-Marie Jacquart, and Mourad Sefrioui. Constrained Optimization of Multilayered Anti-Reflection Coatings using Genetic Algorithms. In Lipo Wang, Kay Chen Tan, Takeshi Furuhashi, Jong-Hwan Kim, and Xin Yao, editors, *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'2002)*, volume 1, pages 172–177, Orchid Country Club, Singapore, November 2002. Nanyang Technical University.

[113] C. O. Maidana, M. Berz, and K. Makino. Muon beam ring cooler simulations using COSY INFINITY. *Insitute of Physics CS*, 175:211–218, 2004.

[114] K. Makino and M. Berz. Effects of kinematic correction on the dynamics in muon rings. *AIP CP*, 530:217–227, 2000.

[115] K. Makino and M. Berz. Map-based muon cooling channel simulations with COSY INFINITY. In B. King, editor, *Proc. of 6-Month Feasibility Study on High Energy Muon Colliders*, 2001. http://pubweb.bnl.gov/people/bking/mucoll/index.html.

[116] K. Makino and M. Berz. Recent applications of COSY to nonlinear beam dynamics problems. In *Proceedings of the APS/DPF/DPB Summer Study on the Future of Particle Physics (Snowmass 2001)*, 2002. number T510.

[117] K. Makino and M. Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 4(4):379–456, 2003.

[118] K. Makino and M. Berz. Verified global optimization with Taylor model methods. *International Journal of Computer Research*, 12,2:245–252, 2003.

[119] K. Makino and M. Berz. Solenoid elements in COSY INFINITY. *Institute of Physics CP*, 175:219–228, 2004.

[120] K. Makino and M. Berz. Tetra cooler ring simulation in cosy infinity. In *Neutrino Factories and Superbeams*, volume 721, page 418. AIP Conference Proceedings, 2004.

[121] K. Makino and M. Berz. Range bounding for global optimization with Taylor models. *Transactions on Computers*, 4,11:1611–1618, 2005.

[122] K. Makino and M. Berz. Verified global optimization with taylor model based range bounders. *Transactions on Computers*, 4(11):1611–1618, November 2005.

[123] K. Makino, M. Berz, D. Errede, and C. J. Johnstone. High order map treatment of superimposed cavities, absorbers, and magnetic multipole and solenoid fields. *Nuclear Instruments and Methods A*, 519:162–174, 2004.

[124] K. Makino, D. Errede, and M. Berz. Cooling channel simulations based on map methods. In *Proc. of the APS/DPF/DPB Summer Study on the Future of Particle Physics (Snowmass 2001)*, 2002. number T702,.

[125] Kyoko Makino. *Rigorous Analysis of Nonlinear Motion in Particle Accelerators.* PhD thesis, Michigan State University, East Lansing, Michigan, USA, 1998.

[126] Kyoko Makino, Martin Berz, Youn-Kyung Kim, and Pavel Snopok. Long-term stability of large accelerators. *ECMI Newsletter*, 39, 2006.

[127] G.P. McCormick. Penalty function versus nonpenalty function methods for constrained nonlinear programming problems. *Mathematical Programming*, 1:217–238, 1971.

[128] Efrén Mezura-Montes. *Alternative Techniques to Handle Constraints in Evolutionary Optimization.* PhD thesis, Computer Science Section, Electrical Eng. Department., CINVESTAV-IPN, México City, México, December 2004.

[129] Z. Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proc. of the Fourth Annual Conference on Evolutionary Programming*, pages 135–155, Cambridge, MA, 1995. The MIT Press.

[130] Z. Michalewicz and N. Attia. Evolutionary optimization of constrained problems. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 98–108. World Scientific, 1994.

[131] Z. Michalewicz and C. Janikow. Handling constraints in genetic algorithms. In *Proceeddings of the Fourth International Conference on Genetic Algorithms*, pages 151–157, Los Altos, CA, 1991. Morgan Kaufmann Publishers.

[132] Z. Michalewicz and G. Nazhiyath. Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *Proceedings of the 2nd IEEE International Conference on Evolutionary Computation, vol. 2*, pages 647–651, 1995.

[133] Zbigniew Michalewicz. Genetic Algorithms, Numerical Optimization, and Constraints. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA-95)*, pages 151–158, San Mateo, California, July 1995. University of Pittsburgh, Morgan Kaufmann Publishers.

[134] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, 2nd edition, 1998.

[135] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[136] Box M.J. A new method of constrained optimization and a comparison with other methods. *Computer Journal*, 8:42–52, 1965.

[137] N. V. Mokhov and O. E. Krivosheev. MARS Code Status. Technical Report FERMILAB-Conf-00/181, Fermi National Accelerator Laboratory, August 2000. Presented Paper at the Monte Carlo 2000 International Conference, Lisbon, Portugal, October 23-26, 2000.

[138] N. V. Mokhov and A. van Ginneken. Pion production and targetry at $\mu^+\mu^-$ colliders. Technical Report Fermilab-Conf-98/041, Fermi National Accelerator Laboratory, January 1998. Published Proceedings of the 4th International Conference on Physics Potential and Development of Muon Colliders, San Francisco, California, December 10-12, 1997.

[139] Holly Moore. *MATLAB for Engineers*. ESource. Prentice Hall, 1st edition, April 2006.

[140] N.N. Nekhoroshev. An exponential estimate of the time of stability of nearly integrable hamiltonian systems. *Uspekhi Mat. Nauk*, 32(6), January 1977.

[141] D. Neuffer. Exploration of the "high-frequency" buncher concept. Neutrino Factory/Muon Collider Notes MUC-NOTE-DECAY_CHANNEL-0269, Fermi National Accelerator Laboratory, 2003. see http://www-mucool.fnal.gov/notes/notes.html.

[142] D. Neuffer and A. Van Ginneken. High-frequency bunching and $\varphi - \delta$E rotation for a muon source. In *Proceedings of 2001 Particle Accelerators Conference*, page 2029, Chicago, 2001.

[143] David Neuffer. Private communication.

[144] David Neuffer. Beam dynamics problems of the muon collaboration: ν-factories and μ^+–μ^- colliders. Neutrino Factory/Muon Collider Notes MUC-NOTE-COOL-THEORY-0266, Fermi National Accelerator Laboratory, 2002.

[145] David Neuffer. More studies on cooling with high-frequency phase rotation. Neutrino Factory/Muon Collider Notes MUC-NOTE-COOL-THEORY-0335, Fermi National Accelerator Laboratory, 2006.

[146] The Neutrino Factory and Muon Collider Collaboration Web page. on-line. `http://www.cap.bnl.gov/mumu/`.

[147] Jorge Nocedal and Stephen Wright. *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer, July 2006.

[148] Collier TC. Ofria C, Adami C. Selective pressures on genomes in molecular evolution. *Journal of Theoretical Biology*, 4(222):447–483, June 2003.

[149] Akira Oyama, Koji Shimoyama, and Kozo Fujii. New Constraint-Handling Method for Multi-Objective Multi-Constraint Evolutionary Optimization and Its Application to Space Plane Design. In R. Schilling, W. Haase, J. Periaux, H. Baier, and G. Bugeda, editors, *Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems (EUROGEN 2005)*, Munich, Germany, 2005.

[150] S. Ozaki, R. Palmer, M. Zisman, J. Gallardo, and (eds.). Feasibility study-II of a muon-based neutrino source. Technical Report BNL-52623, Muon Collider Collaboration, June 2001. see http://www.cap.bnl.gov/mumu/studyii/FS2-report.html.

[151] Vilfredo Pareto. *Cours D'Economie Politique*, volume I. F. Rouge, Lausanne, 1896.

[152] Kevin Paul. Frontend optimization, development and application. presentation.

[153] Perl Programming Language. on-line. `http://www.perl.org`.

[154] A. A. Poklonskiy, D. Neuffer, C. J. Johnstone, M. Berz, K. Makino, D. A. Ovsyannikov, and A. D. Ovsyannikov. Optimizing the adiabatic buncher and phase rotator. *Nuclear Instruments and Methods*, 558:135–141, 2005.

[155] A.A. Poklonskiy, D.A. Ovsyannikov, A.D. Ovsyannikov, D. Neuffer, and M. Berz. Exploring the bunching section of the neutrino factory. In *Physics and Control, 2005. Proceedings. 2005 International Conference*, pages 272–277, August 2005.

[156] Alexey Poklonskiy. Studies on performance of the COSY Infinity optimizers on constraints satisfaction. High Energy Physics Preprint HEP-080317, Michigan State University, East Lansing, MI, 48823, USA, April 2008. available at `http://www.bt.pa.msu.edu/pub/`.

[157] D. Powell and M. M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proceedings of the Fifth International Convference on Genetic Algorithms*, pages 424–430, Los Altos, CA, 1993. Morgan Kaufmann Publishers.

[158] P. Prinetto, M. Rebaudengo, and M. Sonza Reorda. Hybrid genetic algorithms for the traveling salesman problem. In *Proc. of the International Conference on Neural Networks and Genetic Algorithms*, pages 559–566, 1993.

[159] P.Snopok, C.Johnstone, and M.Berz. Simulation and optimization of the tevatron accelerator. *Springer*, 50:199–209, 2005.

[160] S Ramberger and Stephan Russenschuck. Genetic algorithms for the optimal design of superconducting accelerator magnets. Technical Report LHC-PROJECT-REPORT-275, CERN Document Server [http://cdsweb.cern.ch/oai2d.py] (Switzerland), 1999.

[161] Ingo Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* PhD thesis, Technical University of Berlin, 1971.

[162] J. Richardson, M. Palmer, G. Liepins, and M.Hillard. Some guidelines for genetic algorithms with penalty functions. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[163] Thomas P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, September 2000.

[164] S. Sandqvist. On finding optimal potential customers from a large marketing database – A genetic algorithm approach. In *Proc. of the International Conference on Neural Networks and Genetic Algorithms*, pages 528–535, 1993.

[165] Johannes J. Schneider and Scott Kirkpatrick. *Stochastic Optimization.* Scientific Computing. Springer, 1st edition, November 2006.

[166] M. Schoenauer and S. Xanthakis. Constrained ga optimization. In *Proceedings of the Fifth International Convference on Genetic Algorithms*, pages 573–580, Los Altos, CA, 1993. Morgan Kaufmann Publishers.

[167] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1998.

[168] Rigorous computation – self-verified methods page. on-line. `http://bt.pa.msu.edu/index_selfvalidated.htm`.

[169] Ankur Sinha, Aravind Srinivasan, and Kalyanmoy Deb. A Population-Based, Parent Centric Procedure for Constrained Real-Parametrer Optimization. In *2006 IEEE Congress on Evolutionary Computation (CEC'2006)*, pages 943–949, Vancouver, BC, Canada, July 2006. IEEE.

[170] Jr. Stanley Humphries. *Principles of Charged Particle Acceleration.* John Wiley and Sons, 1999.

[171] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, Berkeley, CA, 1995.

[172] Almo Törn and Antanas Zilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science.* Springer, 1989.

[173] Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl.* O'Reilly Media, 3rd ed. edition, July 2000.

[174] Alex Wittig. Private communication.

[175] H. Wollnik. *Optics of Charged Particles.* Academic Press, Orlando, Florida, 1987.

[176] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions*, 1(1):67–82, April 1997.

[177] F. Zimmermann, C. Johnstone, M. Berz, B. Erdélyi, K. Makino, and W. Wan. Fringe fields and dynamic aperture in muon storage rings. Neutrino Factory/-Muon Collider Notes MUC-NOTE-NEUTRINO-SRC-0095, Fermi National Accelerator Laboratory, 2000. see http://www-mucool.fnal.gov/notes/notes.html, also CERN SL 2000-011 A-P, and CERN NuFact Note 21, CERN.

[178] Michael S. Zisman. International scoping study of a future accelerator neutrino complex. In *Proceedings of EPAC 2006*, pages 2427–2429, Edinbrugh, Scotland, 2007.