Ruhr-Universität Bochum

Institut für Experimentalphysik I

Application of Evolutionary Strategies to Automated Parametric Optimization Studies in Physics Research

Dissertation zur Erlangung des Grades
Doktor der Naturwissenschaften
am Institut für Experimentalphysik I
der Ruhr-Universität Bochum

Vorgelegt
von
Rüdiger Berlich
aus Wuppertal

Bochum, im November 2003

1. Gutachter: Dr. M. Kunze
2. Gutachter: Prof. Dr. H. Koch

Abstract

Conducting the analysis of particle physics events requires the processing of large amounts of data. Imposing criteria such as a restriction of the allowed range of values ("cut") on kinematic variables allows to filter wanted from unwanted events. The result will usually be a signal in a histogram ("peak"). A numerical figure of merit that is assigned to the peak, such as its significance, can be treated as a direct function of the cuts. This makes particle physics analyses accessible to automated parametric optimization techniques, where the figure of merit is maximized by an appropriate variation of the cuts.

This thesis presents the application of Evolutionary Strategies, an efficient optimization algorithm, to particle physics problems taken from the *BABAR* experiment. An additional usage pattern involving a Dalitz plot analysis is demonstrated using data from the CB/ELSA experiment.

The "EVA" library, a new implementation of distributed Evolutionary Algorithms that was developed as part of this thesis combines as one of the first implementations the ability to perform the optimization in parallel, on devices ranging from single processor- over SMP-machines all the way to clusters and the Grid, with an intuitive interface to a standard tool in particle physics analysis – the ROOT framework.

Contents

Chapter 1

Introduction

Computing has a long history, going back to the early fifties. But not until the late seventies has it become common for particle physicists to use computers as a standard means of analyzing their data. Early particle physics experiments dealt with only dozens or hundreds of "events" to be analyzed, whereas advances in accelerator and detector design and complete changes of paradigms have today led to an inflation of data production. Where early particle physicists dealt with the analysis of single photographic plates or photographs of decays monitored using a bubble chamber, doing particle physics today has to do much more with statistical analysis, "filtering" out the desired event types from possibly petabytes[1] of data. Where the decay of a single pion could initially be observed with photographic exposures, today "b-factories" (such as the *BABAR* experiment at the Stanford Linear Accelerator Center) produce billions of $b\bar{b}$ pairs[2]. The amount of data that will be produced by the next generation of particle detectors has even enforced a new computing paradigm – Grid computing: As it is no longer possible to store and process all data locally, new techniques need to be developed that help to seamlessly integrate distributed computing resources.

Numerous minimization or maximization tasks are implicitly already part of any standard analysis in particle physics – be it the reconstruction of a charged track in a drift chamber or the "fitting" of a mathematical function to a histogram. But some optimization tasks require the processing of such a large amount of data that being able to perform this computation in parallel on a set of, possibly globally distributed, compute nodes represents a significant advantage. Two such examples are presented in this thesis - the optimization of an analysis through the maximization of a figure of merit assigned to a particle signal, using measured data from the *BABAR* experiment, and the minimization involved in doing Dalitz plot analysis at the CB/ELSA experiment.

Part of this thesis comprises a new software library of Evolutionary Algorithms that, as one of the first implementations, combines the ability to perform optimizations in parallel,

[1] 1 Petabyte = 10^6 Gigabyte

[2] being recorded with an electronic data acquisition (DAQ) system

on devices ranging from multi-processor machines over cluster systems to the World Wide Grid, with an intuitive interface to a standard tool of particle physics analysis – the ROOT framework. The library is targeted at particle physicists with the need for long lasting, computationally expensive optimization studies.

1.1 Overview

In order to offer a better understanding of the topics discussed in this thesis, a short overview of the material presented in each chapter is given below :

- Chapter 2, *Physics Motivation and Definitions*, introduces two physics examples – the optimization of particles in histograms and Dalitz plot analysis – that involve the processing of very large amounts of data, thus making their optimization a daunting task.

- Chapter 3, *Evolutionary Algorithms*, introduces Evolutionary Algorithms as a means to perform generic minimization and maximization tasks.

- Chapter 4, *The EVA Library*, discusses the design and implementation of "EVA", a software library of parallel **EV**olutionary **A**lgorithms. The design principles are followed by a discussion of the most important classes of the library. The chapter concludes with a number of performance measurements, most of them being taken at the GridKa compute center of Forschungszentrum Karlsruhe.

- Chapter 5, *BABAR Data: Enhancement of Particle Signals*, discusses the application of the EVA library's implementation of *Evolutionary Strategies* to the optimization of a set of cuts in the reconstruction of three decay modes of the $D_s^{\pm}$ in the *BABAR* experiment. The chapter also includes a discussion of the creation of fake peaks and introduces the *BABAR* experiment.

- Chapter 6, *CB/ELSA Data: Dalitz Plot Analysis*, discusses the application of the EVA library's implementation of *Evolutionary Strategies* to a Dalitz plot analysis done by the CB/ELSA collaboration. The chapter furthermore includes a discussion of the CB/ELSA detector system and quantitatively compares Evolutionary Strategies with a gradient descent optimization method (TMinuit).

- Chapter 7, *Summary and Outlook*, provides a summary and conclusion of this work and tries to highlight possible future developments.

- Appendix A, *Using the EVA library*, gives an overview of the EVA user interface, using programming examples. In addition, the chapter highlights a few consistency checks used to thoroughly test the EVA library.

- Appendix B, *Using Neural Networks to Optimize Significance*, discusses a new method, that has the potential to combine the advantages of Feedforward Neural Networks with parametric optimization studies in physics research. In particular, this approach would be able to exploit correlations between cut variables.

Chapter 2

Physics Motivation and Definitions

Throughout the history of doing physics with computers, minimization and maximization techniques[1] have become a mandatory component of every analysis. Simple examples are the reconstruction of charged particles from data measured in drift chambers in particle physics experiments or the "fitting" of a given mathematical function to a histogram.

Some optimization tasks, however, are on the verge of what can be realistically handled by a single computer, be it because of a particularly large amount of data to be processed or due to intrinsic difficulties of an analysis, such as noisy input or many local optima in a figure of merit to be optimized. As a consequence, despite their potential usefulness for and benefit to particle physics analysis, some optimization paths are rarely pursued.

This chapter discusses two particularly daunting physics examples – the optimization of signals in particle physics histograms and doing Dalitz plot analysis.

Chapters 3 and 4 will then introduce a possible solution to this dilemma - Evolutionary Algorithms and a distributed implementation thereof, which was developed as part of this thesis. Chapters 5 and 6 will then discuss the application of these techniques to real data taken at the *BABAR* and CB/ELSA experiments.

2.1 Optimizing Cuts in Particle Physics Analysis

Searching for a new particle or the reconstruction of a known particle means to first understand its decay modes. As one usually only has data available for its decay products and not for the particle itself, it can only be reconstructed using the data recorded for the final state particles. This can also involve the reconstruction of intermediate decay products that are then in turn used to reconstruct the original particle.

Many kinematic variables help to reduce the amount of unwanted "background" data

[1] or in short: "optimization" techniques

and to increase the fraction of correctly reconstructed particles. For example, in a decay $1 \rightarrow 2+3$ of a particle 1 into particles 2 and 3, the two latter particles can only be produced under a 180^o angle in the rest frame of particle 1. The mass of particle 1 will put further constraints on the energy available for the production of 2 and 3, and not all particle types can be produced due to energy and momentum conservation as well as other conservation laws.

Consequently it is possible to enhance the amount of wanted over unwanted data by the application of restrictions ("cuts") on – usually floating point – variables representing kinematic or otherwise important properties of the decay products. The result of a particle physics analysis is usually a histogram, with a signal ("peak") representing the reconstructed particle and background data representing wrongly reconstructed events. A numeric figure of merit assigned to the peak measures its quality. It can be interpreted as a direct function of the cuts that were applied to the kinematic variables. This, in turn, makes particle physics analysis accessible to automated parametric optimization techniques that try to maximize the figure of merit by varying the cuts in an appropriate way. The definition of the quality or figure of merit is up to each individual researcher. We will present two common examples in the next section. Definitions needed to understand this section are explained in figure 2.1.

It should be obvious that the automatic determination of optimal cuts is a desirable goal. But at the same time checking the quality of a peak usually requires the processing of potentially hundreds of thousands of recorded events from particle physics experiments. Automated parametric optimization techniques usually require hundreds of different sets of cuts to be checked for their quality, and each check triggers a new re-processing of the event data. And so the required amount of computing power to perform such an analysis is immense. In particular, a single computer is unsuitable for this task, both because of memory constraints and due to a lack of computing power.

2.1.1 Quality measures

2.1.1.1 Significance $\mathcal{S}$

A common task in particle physics is the measurement of decay rates, defined as $DR = \frac{N}{\varepsilon}$, or decay fractions, defined as $BR = \frac{DR_1}{DR_2} = \frac{N_1\varepsilon_2}{N_2\varepsilon_1}$. N is the number of decays of a particular type that were found in an analysis, ε is their reconstruction efficiency.

Using the standard gaussian error propagation one can calculate the error on BR as

$$\Delta BR = \sqrt{\left(\frac{\partial BR}{\partial N_1}\Delta N_1\right)^2 + \left(\frac{\partial BR}{\partial N_2}\Delta N_2\right)^2 + \left(\frac{\partial BR}{\partial \varepsilon_1}\Delta \varepsilon_1\right)^2 + \left(\frac{\partial BR}{\partial \varepsilon_2}\Delta \varepsilon_2\right)^2}$$

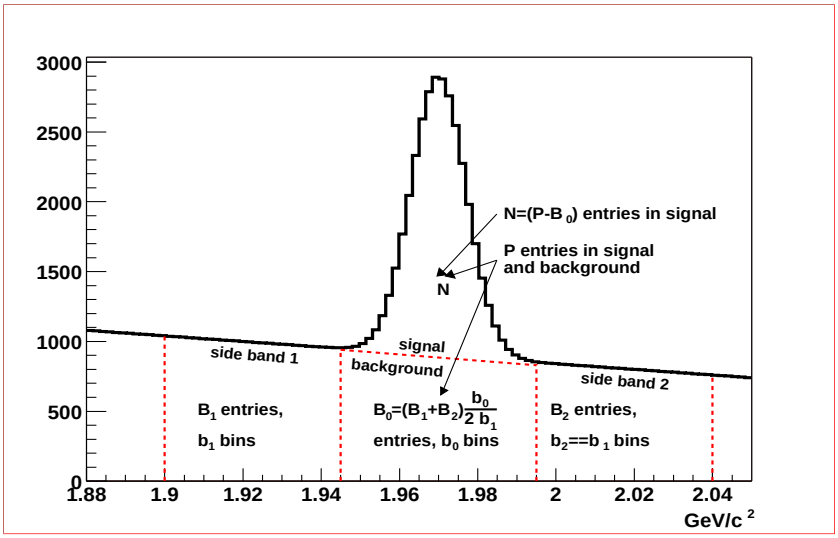

Figure 2.1: *Definitions for a particle signal and background in a histogram. The number of entries N in a particle signal can be calculated with the help of the number of background events B_0 below the peak. An estimate for B_0 can be calculated from the number of entries in the sidebands, if the background is flat.*

$$
\begin{aligned}
&= \sqrt{\left(\frac{\varepsilon_2 \Delta N_1}{\varepsilon_1 N_2}\right)^2 + \left(\frac{\varepsilon_2 N_1 \Delta N_2}{\varepsilon_1 N_2^2}\right)^2 + \left(\frac{N_1 \varepsilon_2 \Delta \varepsilon_1}{N_2 \varepsilon_1^2}\right)^2 + \left(\frac{N_1 \Delta \varepsilon_2}{N_2 \varepsilon_1}\right)^2} \\
&= \sqrt{\left(\frac{\varepsilon_2 N_1}{\varepsilon_1 N_2}\frac{\Delta N_1}{N_1}\right)^2 + \left(\frac{\varepsilon_2 N_1}{\varepsilon_1 N_2}\frac{\Delta N_2}{N_2}\right)^2 + \left(\frac{\varepsilon_2 N_1}{\varepsilon_1 N_2}\frac{\Delta \varepsilon_1}{\varepsilon_1}\right)^2 + \left(\frac{\varepsilon_2 N_1}{\varepsilon_1 N_2}\frac{\Delta \varepsilon_2}{\varepsilon_2}\right)^2} \\
&= \frac{\varepsilon_2}{\varepsilon_1}\frac{N_1}{N_2}\sqrt{\left(\frac{\Delta N_1}{N_1}\right)^2 + \left(\frac{\Delta N_2}{N_2}\right)^2 + \left(\frac{\Delta \varepsilon_1}{\varepsilon_1}\right)^2 + \left(\frac{\Delta \varepsilon_2}{\varepsilon_2}\right)^2}
\end{aligned}
$$

With the definition of $\mathcal{S} := \frac{N}{\Delta N}$, henceforth called "significance", and $\mathcal{E} := \frac{\varepsilon}{\Delta \varepsilon}$ one can conclude that

$$\Delta BR = \frac{\varepsilon_2}{\varepsilon_1}\frac{N_1}{N_2}\sqrt{\left(\frac{1}{\mathcal{S}_1}\right)^2 + \left(\frac{1}{\mathcal{S}_2}\right)^2 + \left(\frac{1}{\mathcal{E}_1}\right)^2 + \left(\frac{1}{\mathcal{E}_2}\right)^2} \tag{2.1}$$

Obviously it is possible to minimize the error ΔBR by maximizing the significances $\mathcal{S}_1$ and $\mathcal{S}_2$. It is thereforece necessary to take a closer look at the error ΔN in $\mathcal{S} = \frac{N}{\Delta N}$.

It is assumed that the background B_0 below a signal in a histogram is calculated using the number of entries in the sidebands. Please note that in order to achieve satisfactory results using this method, it is necessary for the background to be flat.

Using the definitions made in picture 2.1, one can conclude that the error on $N = P - \left(\frac{b_0}{2b_1}\right)(B_1 + B_2)$ is

$$\Delta N = \sqrt{\left(\frac{\partial N}{\partial P}\Delta P\right)^2 + \left(\frac{\partial N}{\partial B_1}\Delta B_1\right)^2 + \left(\frac{\partial N}{\partial B_2}\Delta B_2\right)^2 + correl.term}$$

$$= \sqrt{\left(\sqrt{P}\right)^2 + \left(\frac{b_0}{2b_1}\sqrt{B_1}\right)^2 + \left(\frac{b_0}{2b_1}\sqrt{B_2}\right)^2 + correl.term}$$

or, when simplifying this expression,

$$\Delta N = \sqrt{P + \left(\frac{b_0}{2b_1}\right)^2 (B_1 + B_2) + correl.term} \tag{2.2}$$

The number of entries in the two sidebands, B_1 and B_2, will be low if the background B_0 below the peak is low, and vice versa. This means that the number of histogram entries $P = N + B_0$ in the peak region is correlated to the number of entries in the side bands. Thus a correlation term must be added to the error-contributions of the side band-entries in equation 2.2.

As $B_0 = \frac{b_0}{2b_1}(B_1 + B_2)$ (see figure 2.1) and $P = N + B_0$, equation 2.2 can also be written as

$$\Delta N = \sqrt{N + \left(1 + \frac{b_0}{2b_1}\right) B_0 + correl.term} \tag{2.3}$$

As it is the purpose of this thesis to demonstrate the effects of parametric optimization in particle physics experiments rather than to perform an entire analysis, a simplified version of equation 2.3 will be used in the following to calculate the significance, neglecting the correlation term, but taking into account the statistical error on the side bands. They are assumed to be of the same size as the peak region : $b_0 = 2b_1$.

The significance $\mathcal{S}$ can then be written as

$$\mathcal{S} = \frac{N}{\sqrt{N + 2B_0}} \tag{2.4}$$

A real-life analysis would have to calculate the background from Monte Carlo data, ignoring the side bands. This method eliminates the correlation between the alleged number of background events and the number of entries in the peak. If B_{MC} is the number of Monte Carlo events assigned to the background, the significance would become

$$\mathcal{S}_{MC} = \frac{N}{\sqrt{P + B_{MC}}} \tag{2.5}$$

in this case. No systematic errors have been taken into account.

If $B_{MC} \approx B_0$, then, due to $P = B_0 + N$, equation 2.5 will in fact assume very similar values to equation 2.4. Neglecting the correlation term in equation 2.2 and assuming that $b_0 = 2b_1$ will thus yield realistic results, and equation 2.4 will be used in the following as

the basis for a figure of merit to be optimized. But as the calculation of a square root is computationally expensive, the "dilution" $\mathcal{D} = \mathcal{S}^2$ will be used instead of $\mathcal{S}$:

$$\mathcal{D} = \mathcal{S}^2 = \frac{N^2}{(N + 2B_0)} \tag{2.6}$$

As a side note, another common figure of merit for the quality of a peak used for example by the *BABAR* collaboration is simply

$$\hat{\mathcal{D}} = \frac{N^2}{N + B_0} \tag{2.7}$$

This is the form that equation 2.6 would have in the case of very large sidebands ($2b_1 \gg b_0$ – please also compare the definition of ΔN in equation 2.3).

An important feature of the significance $\mathcal{S}$, when being used as a quality measure, is that it is not always easy to see whether a peak has a high quality or not. It is necessary to calculate the significance in order to find out about the quality of a peak. As an example, increasing the number of background entries B_0 by a factor of 2, while increasing the number of signal entries N by a factor of 1.5, can still lead to an increased significance, although the new histogram might "look" worse than the first one. E.g., if $B_0 = 1000$ and $N = 200$, then

$$\frac{(1.5 * N)^2}{1.5 * N + 2 * 2B_0} = 20.93 > \frac{N^2}{N + 2B_0} = 18.18 \tag{2.8}$$

This is unlike the signal-to-noise ration R_{sn} being discussed in the next section and makes the maximization of $\mathcal{S}$ an outstanding candidate for automated parametric optimization techniques.

The application of parametric optimization techniques to the maximization of $\mathcal{S}^2$ will be discussed in chapter 5 on the example of reconstruction of three decay modes of the D_s, using data measured at the *BABAR* experiment.

2.1.1.2 Signal/Background R_{sn}

A more naive measure for the quality of a peak in a histogram is the signal-to-noise ratio $R_{sn} = \frac{N}{B_0}$. It is immediately clear that a large number of entries N in a peak with a low number of background events B_0 will yield better results in an analysis than a low N with a high B_0. The implicit assumption in this statement, however, the expectation to have enough entries in the peak to actually perform such an analysis, can prove to be wrong.

As it is possible to maximize R_{sn} by minimizing the number of background events, an algorithm using R_{sn} as a quality measure will tend to enforce very tight cuts, indeed

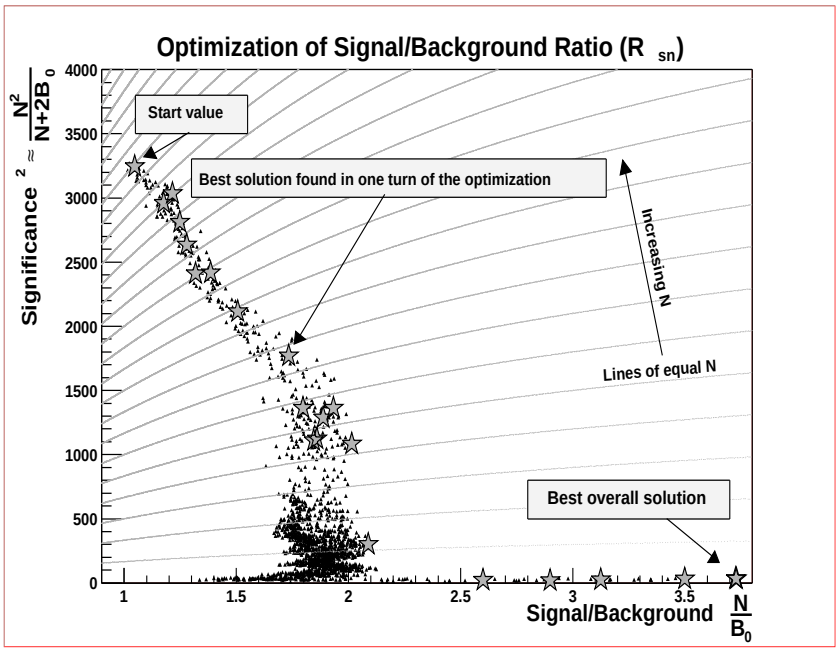

Figure 2.2: *Squared Significance vs. Signal-to-noise ratio "R_{sn}" during an optimization with R_{sn} as a measure of the quality of a D_s peak. The small triangles stand for each solution that was investigated. Maximizing R_{sn} leads to signals with a small number of entries and low significance (see text). R_{sn} can thus not be considered to be a suitable quality measure in parametric optimization studies.*

resulting in a low B_0, but also in an insufficient N. Figure 2.2 shows different solutions on the example of the decay $D_s \to \Phi\pi$, taken from data in the *BABAR* experiment[2], with R_{sn} being used as a figure of merit in an "optimization" run. The squared significance $\mathcal{S}^2 = \frac{N^2}{N+2B_0}$ is plotted against R_{sn}. For the understanding of the plot it is important to know that, in the $\mathcal{S}^2 - R_{sn}$ plane, equal values of the number of entries N in the peak can be represented as lines. The small black triangles stand for each solution that was investigated in the optimization process. The evaluation is done in turns, and the larger stars represent the best solution found in each turn of the optimization.

It is obvious in figure 2.2 that, while an almost fourfold improvement of R_{sn} could be achieved for the "best" solution[3] (labelled "Best overall solution" in the plot), this "optimization" leads to a very low N. Similarly the resulting $\mathcal{S}^2$ of the D_s peak approaches 0. The best solution found leads to a peak with just 50 entries, compared to about 9500 when the optimization started. Hence it can be concluded that R_{sn} is not a good measure of the quality of a peak when doing parametric optimizations.

The large number of entries in figure 2.2 once again underlines the fact that the large amount of data to be processed makes the optimization of particle signals unsuitable to stand-alone computers. Parallelisation of optimization toolkits is a possible workaround. Each of the entries in figure 2.2 represents the evaluation of a set of cuts, with 20 independent sets being checked in each turn of the optimization. This means that a sequential optimization would take much longer than a parallel optimization procedure that can investigate each set simultaneously on a different computer.

[2]This decay mode will be discussed in more detail in chapter 5.

[3]Best according to the value of R_{sn}

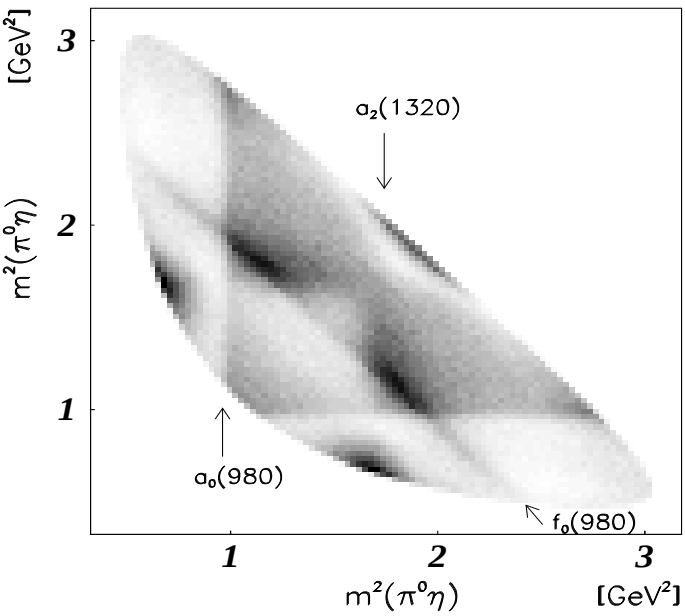

Figure 2.3: *A Dalitz plot representing the decay $p\bar{p} \to \pi^0\pi^0\eta$. The plot uses measured data from the Crystal Barrel experiment at LEAR/CERN, a predecessor of the CB/ELSA experiment being discussed in chapter 6. Due to energy- and momentum conservation not all areas of the plot are populated with events. Clear structures in the Dalitz plot indicate the presence of resonances. They are labelled in the plot. See the text for further information on Dalitz plots. (Courtesy Spanier / Meyer [SPAN98]).*

2.2 Dalitz Plot Analysis

Another example for parametric optimization problems involving very long computations is the minimization procedure done as part of a Dalitz plot analysis.

In particle decays, all essential kinematical variables of a three-body final state can be described by just two independent variables. This feature is used in *Dalitz plots*, first introduced by R.H. Dalitz in 1953.

If, in a reaction

$$1 + 2 \to 3 + 4 + 5 \tag{2.9}$$

m_{34} is the squared invariant mass of particles 3 and 4 and m_{45} the one of particles 4 and 5, then, for a given incident energy, the third squared invariant mass is fully determined by the other two, as

$$m_{35}^2 + m_{34}^2 + m_{45}^2 = m_{12}^2 + m_3^2 + m_4^2 + m_5^2 = const \tag{2.10}$$

Thus, in the most common form of the Dalitz plot, for a given set of decays ("events") with fixed center of mass Energy E_{cms}, the squared invariant masses of two of the three particle combinations are plotted against each other into a two-dimensional histogram[4].

Due to energy- and momentum-conservation, the non-void area of a Dalitz plot is confined to a well-defined region which, in the absence of interferences or resonances between the

[4]For practical reasons one has to allow E_{cms} to vary within a small range in order to have a sufficient number of events in an analysis. This will lead to structures in a Dalitz plot being smeared. This will happen the more the broader the energy range being used is.

final state particles can be shown to be uniformly populated[5]. Structures, such as bands, will be observed if there is resonant behavior between each two of the three final state particles.

An example of a Dalitz plot can be found in figure 2.3. Several hundred thousand of events of the type $p\bar{p} \rightarrow \pi^0\pi^0\eta$, measured at the CB/LEAR experiment (a predecessor of the CB/ELSA experiment discussed in chapter 6), were used to produce this plot. $m^2(\pi^0\eta)$ is plotted against $m^2(\pi^0\eta)$. The structures in the plot are clearly visible, the resonances are labelled with their corresponding particles.

Information taken from the analysis of these structures can be used to determine parameters such as the masses, widths or quantum numbers of the resonances the final-state particles have originated from[6]. The most common way of extracting this information is through a *Dalitz plot analysis*, a special form of a *partial wave analysis* ("PWA").

Only the principal procedure can described here, as a detailed description of the theory behind Dalitz plot analyses goes beyond the scope of this work. A thorough description, however, can be found in [LUED95].

In the simplified framework of the canonical formalism, weights are assigned to phase-space distributed MC events, which are then filled into a two-dimensional histogram[7]. Each weight is a function of the parameters that should be extracted from the Dalitz plot, such as masses or widths of particles or their relative contributions. The weight is adapted by varying these parameters, until the Dalitz plot for phase-space distributed Monte Carlo events is transformed into the corresponding plot for data events.

The figure of merit in this case is often simply a χ^2 representing the degree of deviation of the weighed MC Dalitz plot from the one using measured data.

A Dalitz plot analysis hence requires the minimization of a function with a large amount of free parameters ("fitting") and thus is itself a high-dimensional minimization problem.

Once the fit has converged, the free parameters of the weights, adapted by a minimization algorithm, reflect the situation present in the physical data.

As in the case of the optimization of particle signals in histograms, the minimization process has to cope with a large amount of data. The reason for this is that the weights are calculated on the basis of individual Monte Carlo events. It is not uncommon to require a 5-10 times larger Monte Carlo dataset than measured data is available, as a measurement should not be dominated by the statistical error of the Monte Carlo data.

An additional complication lies in the fact that optimization algorithms that are susceptible to getting stuck in local optima might have difficulties dealing with noisy input data, so choosing the right optimization tool is important.

[5] Please note that acceptance effects of the detector will lead to a deviation from a pure phase space distribution of entries

[6] The original publication can be found at [DAL53], [BOCK98] gives a short overview of Dalitz plots.

[7] representing a Dalitz plot

We will discuss the application of Evolutionary Strategies, an efficient optimization algorithm, to a Dalitz plot analysis done at the CB/ELSA experiment in chapter 6. The chapter also contains a comparison with a more standard algorithm based on a gradient descent ("Minuit").

Chapter 3

Evolutionary Algorithms

Just like many conventional methods, Evolutionary Algorithms provide a means of finding optimal solutions to multidimensional, numerical or discrete problems.

Many different variants of Evolutionary Algorithms are known, including Classifier Systems and Genetic Programming. They share a similarity to the principles of biological evolution, which by itself forms an optimization scheme. [EAFAQ] provides a good overview of the Evolutionary Algorithms known to date.

However, the most well known examples of this family of optimization tools are Evolutionary Strategies and Genetic Algorithms. The current chapter is exclusively dedicated to these two methods. The term "Evolutionary Algorithm" in this chapter henceforth shall denote only these two optimization schemes.

Evolutionary Strategies will be used in chapters 5 and 6 for the optimization of two computationally challenging physics examples.

3.1 Background

Evolutionary Strategies and Genetic Algorithms share many features, including a similar history, and a long lasting dispute among their inventors.

Having been developed in the mid sixties, at a time, when detailed, computer-based optimization studies were still uncommon, early development of Evolutionary Strategies seems to have been inspired more by the wish for direct practical application than by the attempt to emulate natural processes as closely as possible. Dr. Ingo Rechenberg, an engineer by profession, first published his method in his book *Evolutionsstrategie* [RECH73] in 1973. While it was subsequently refined by many other scientists, most notably Paul Schwefel, most of the core principles of the method still remain. An update to [RECH73] was published in 94 by Ingo Rechenberg [RECH94]. Early work included the optimization of fluid flow in a pipe or the improvement of supersonic jet engines. Work on both topics was done

using mechanical models rather than computers. Dr. Ingo Rechenberg today teaches at Technische Universität Berlin.

Genetic Algorithms were initiated by John Holland. Just like Evolutionary Strategies, their development commenced in the mid-sixties. Unlike Rechenberg, however, Holland's primary interest was in a close replication of natural processes and in the computer science aspects of his method [SCHO93]. He focused on the question, how nature, using only the rather simple genetic code, could create something as astonishing as intelligence, self organization and the most complex adaptation patterns. It should be noted that Holland didn't neglect the possibility of practical application of his research, especially with respect to optimization studies. A detailed description of Holland's work and the underlying principles was published by John Holland in 1975[HOLL75].

3.2 Relevance

"Automation" often means finding the optimal way to accomplish certain tasks. Here the term "optimal" is fuzzy, but almost always involves rating one possible solution over another. A scientific approach would assign values to solutions of sub-criteria of a problem and judge the overall solution according to some amalgamation of these values. Varying some of the "sub-solutions" might then lead to a better result. In other words, finding the optimal solution to an arbitrary problem implicitly resembles the procedure deployed by computerized optimization methods, and automation of problem-solving can often be expressed as the minimization or maximization of a numerical *fitness* or quality of a solution, depending on a variation of the sub-solutions of a problem. A world that increasingly tries to automate many of every day's tasks therefore relies on computerized optimization techniques.

Whether Evolutionary Algorithms are a suitable method for a given task obviously depends on the task itself. Sometimes, more standard optimization schemes, like a simple gradient descent (see e.g. 3.3.1), are able to provide solutions faster, if not better. But thanks to their easy parallelisability, Evolutionary Algorithms increasingly provide a more suitable, often superior alternative to standard methods. In some cases, like for example problems with many local optima or extremely noisy input data, Evolutionary Algorithms outperform all of their competitors. Much more than still a few years ago, when most computation was done on single processor machines or expensive multi processor systems, the current trend to large compute clusters and the rise of a new computing paradigm, called Grid computing (see e.g. section 3.6), gives new life to an approach that started more as a Gedankenexperiment than a method designed to solve even the most complicated optimization problems.

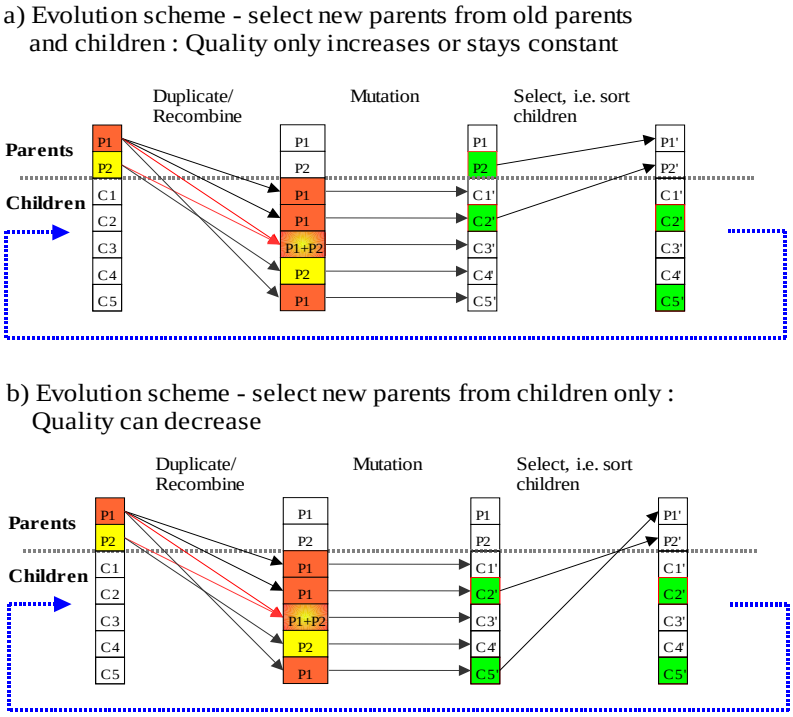

Figure 3.1: *Schematic overview of Evolutionary Algorithms. The duplication of parent individuals, possibly including a recombination of two or more parents (example "P1+P2"), is followed by the mutation of child individuals and the selection of the best individual(s) as new parent(s). This procedure forms the basis of both Evolutionary Strategies and Genetic Algorithms.*

3.3 Evolutionary Strategies

It is the purpose of this chapter to provide an introduction to the basic principles of Evolutionary Algorithms. Since Evolutionary Strategies form the basis of the parametric optimization techniques discussed in this thesis (see chapters 5 and 6), we will start with a discussion of their functionality and inner workings. New terminology will be introduced along the way.

At the heart of an Evolutionary Strategy (henceforth also called *ES*) is a *population* of structures, called *individuals* . In the case of the C or C++ programming language these structure could be implemented using a *struct* or a *class*.

Each individual contains a user-defined set of floating point variables, whose meaning is defined by an *evaluation function* common to all individuals. A simple example would be an individual containing two floating point variables x and y, that are associated with the evaluation function $f(x, y) = x^2 + y^2$, i.e. a two-dimensional parabola. In our example the task is to find the set of x and y parameters, that leads to a minimal value of $f(x, y)$, i.e. $x = 0$ and $y = 0$.

It is the purpose of an evaluation function to assign one floating point number to an individual, equivalent to its quality or *fitness*, such that one individual can be rated as being better than another, or, in other words, having a better fitness. In many cases this means finding the parameter set yielding a minimum value of the evaluation function rather

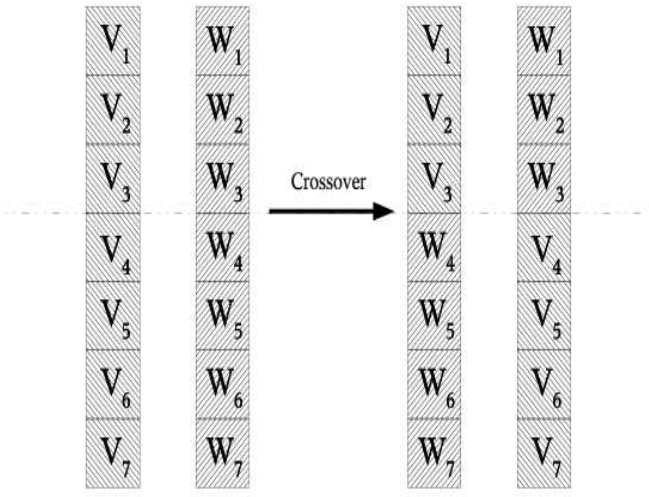

Figure 3.2: *In the context of Evolutionary Algorithms an individual is represented by a "feature" vector. In the case of Evolutionary Strategies this vector contains floating point values. Individuals being used in a Genetic Algorithm contain bits. The content of part of each feature vector is exchanged in a cross-over procedure between two individuals. Cross-over is more common in Genetic Algorithms, but can also be used in Evolutionary Strategies.*

than its maximum, i.e., in some cases a low value of the evaluation function can denote a high fitness.

Often a set of user-defined data is assigned to each individual. In the most common case this data remains constant throughout the optimization process, but one could also think of a time-dependant stream of data.

A population can be understood as a collection of a (usually constant) number of individuals. Among them one can distinguish between one or more parent individuals and a set of child individuals. Within a population, individuals undergo a cycle of *recombination* or *duplication* (that is : the creation of new children from parents), *mutation* (i.e. modification of the children according to a user defined scheme) and *selection* according to their fitness, again specified by the user. Picture 3.1 shows this process in detail.

In a first step, new children are created from a set of parent individuals[1]. This can involve the simple duplication of randomly selected parents or more sophisticated "sexual" recombination schemes using two or more parents as the basis for a new child individual. Picture 3.2 explains one of the most common sexual recombination schemes, the "cross over" of two parent individuals. In any case children inherit some of the characteristics of their parents.

In the second step, each child individually undergoes a mutation process. The most common mutation scheme in Evolutionary Strategies involves adding a random number with a gaussian[2] probability density function (PDF)[3] to each floating point variable of an individ-

[1]They are still labelled as "P" in figure 3.1 in order to reflect the fact that, at this point, they are still exact copies of one parent individual or a combination of the characteristics of two or more parents

[2]Centered around 0.

[3]Henceforth this is called a "gaussian random number"

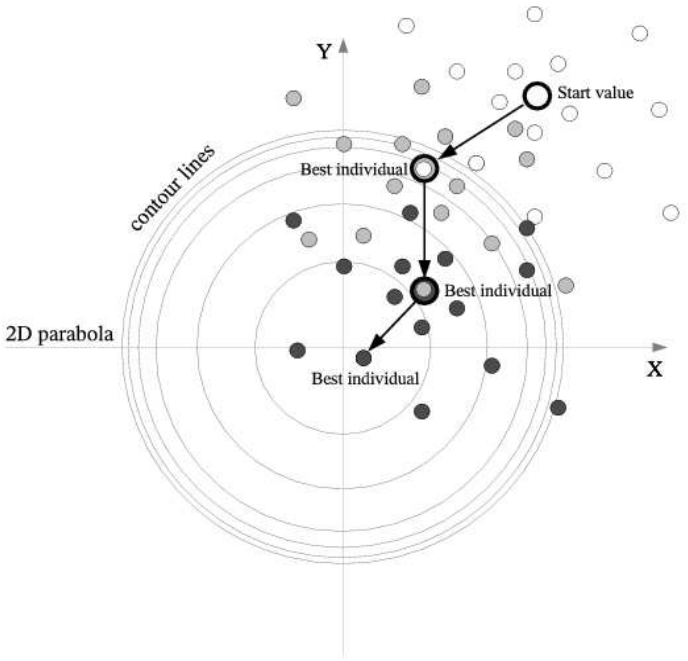

Figure 3.3: *A basic form of Evolutionary Strategies can be demonstrated on the example of a two-dimensional parabola. "Children" are scattered around a parent individual. The density of child individuals increases with decreasing distance to the parent. The best child of a generation is chosen according to its fitness (i.e. the function value of the parabola at the location of the child) and becomes a new parent. The procedure starts again.*

ual. This means that, with a high likelihood, a child that was created as an exact replica of one parent will, within the scope of the evaluation function, yield a similar result to its parent, since the floating point variables it is built from have values close to those of its parent. But it is the gaussian random numbers that help Evolutionary Strategies achieve better results for evaluation functions with many local optima than a simple gradient descent. While the latter tends to get stuck in a local optimum (although some techniques exist that try to circumvent that problem), an Evolutionary Strategy will, after a while, "jump" over such a dip in hyperspace. This is just a consequence of the fact that with a low probability a gaussian random number with a mean value of 0 can also have a very large value, leading to an individual "far" from its parent, beyond the reach of a local optimum.

In the final step the individuals of a population are sorted according to their fitness. In the most common case, both children and parents participate in the sorting, meaning that, if none of the children has a better fitness than their parents, these parents "survive". The best individuals of a population then become parents of the next generation. Alternatively, new parents can be selected from a population's children only. In contrast to the first case this means that the quality of the best individual can decrease whereas, if new parents are selected from the whole population, this quality stays constant in the worst case. The advantage of the second method is that a wider range of the input space is tested for its quality. Experience shows that, especially for computationally expensive evaluation functions, the first possibility is more suitable.

Picture 3.3 illustrates the whole process on the example of a two dimensional parabola, with one parent individual and 15 child individuals per generation. Children are scattered

around the parent, with - as a result of adding gaussian random numbers - more children being in the close proximity of the parent than far away. The best individual of a generation is then selected and becomes the parent of a new generation, at which point the algorithm starts from the beginning.

3.3.1 Evolutionary Strategies and Gradient Descents

A gradient descent can be used to search for minima (or maxima) of multi-dimensional functions, just like Evolutionary Strategies. The basic procedure involves the calculation of the gradient in a given point of an evaluation function, thereby determining the direction of the steepest descent. A "step" is then made in that direction by varying the free parameters of a function accordingly. A gradient descent involves the numerical or symbolic calculation of the partial derivatives of a function for each variable. Information about theory and practice of gradient descent can be found in [BRANDT92].

While the algorithm is well suited to many function types, it has a deficiency in that it tends to get stuck in local optima. This is easy to understand, as a gradient descent can only "walk" down-hill. If it is caught in a local optimum wider than the step width of the algorithm, no further progress is possible. This problem can, to a limited extent, be overcome by varying the step-width dynamically during the optimization, but practice shows that a gradient descent is still susceptible to this problem.

The most common implementation of a gradient descent used in particle physics is `Minuit`, part of the CERN computing library. A more modern implementation – `TMinuit` – can be found in the ROOT framework ([ROOT]).

At this point one might have gotten the impression that Evolutionary Strategies do not add too many new features to what a gradient descent does. Indeed an ES with a single parent shares many of its inner workings with this algorithm. However, there are various characteristics that make an Evolutionary Strategy unique. First and foremost, ES has the ability to follow various concurrent optimization paths, should the user choose to run the algorithm with more than one parent individual[4]. This makes this approach more robust with respect to local optima and, if all optimization paths in the algorithm actually lead to the same optimum, provides some reassurance that the values found actually belong to the global optimum rather than local ones. An Evoltionary Strategy "scatters" child individuals around the parents, with more children located close to a parent than far away. But, with a lower probability, child individuals can reach even very remote spots. This also makes Evolutionary Strategies robust against getting stuck in local optima. A remarkable example, the minimization of the function $f(\vec{r}) = (cos(|\vec{r}|^2) + 2)|\vec{r}|^2$ is shown in section 4.5.6. The gradient descent implemented in `TMinuit` can't cope with this function and gets stuck in the very first local minimum it finds.

[4]Unlike biological evolution, Evolutionary Strategies do not require more than one parent individual to be present.

In Evolutionary Strategies, children can share parameters of more than one parent, thus giving them the ability to combine characteristics of a multitude of individuals. A nice feature of ES is that the evaluation function used to judge one individual over another makes no requirement whatsoever about the input space. It may be non-continuous or could include singularities. A significant speedup over a gradient descent can be achieved by the easy parallelisability of Evolutionary Strategies, discussed in section 3.5 (see also section 4.4.6). This is especially true if the evaluation function consumes significant amounts of compute time, as is often the case especially in particle physics.

So, while both algorithms might, at first sight, share many similarities, they are indeed very different.

In practice it is always possible to find functions that one algorithm can't cope with but the other does. For example, when minimizing a 100-dimensional parabola, a gradient descent will out-perform an Evolutionary Strategy by a large factor. But as soon as there are many local optima, an Evolutionary Strategy is much more likely to succeed. Similarly, the easy parallelisability will give ES an advantage when dealing with computationaly expensive functions to be optimized.

Section 6.3 compares gradient descents and Evolutionary Strategies using a physics example.

3.4 Genetic Algorithms

We will now shortly discuss Genetic Algorithms, as the EVA library (it is discussed in the next chapter) also implements parts of this algorithm.

Just like Evolutionary Strategies, Genetic Algorithms ("GA") follow the procedure explained in picture 3.1. In contrast to them, however, GA-individuals consist of a vector of bit values rather than floating point numbers. They are therefore more suited to discrete optimization problems, such as combinatoric, high multiplicity searches (e.g. "find the combination of charged tracks in a particle physics experiment that yield the best fit result"). Numerical problems are also accessible to GAs, by simply using the bit representation of a floating point value as a (64-)bit vector[5]. One should note, though, that the most common mutation in GAs, the flipping of a bit value (e.g. "true → false") may involve a large change in the value of a numerical variable assembled from the bits. Imagine changing the uppermost bit of the exponent. While there are ways to circumvent this problem, Evolutionary Strategies do not exhibit this problem at all, so there are only few areas where using GAs in numerical problems is a good choice. They do excel in other areas, though.

Much more often than in the case of Evolutionary Strategies, GAs use sexual recombination schemes to generate new children. The most common scheme is the cross over of two

[5] in the case of a "double" value

parents explained in Picture 3.2. Like in the case of the bit-flip, crossing over two bit vectors representing floating point variables will lead to a large change in their value.

There are also problems that involve both numerical and discrete aspects. E.g. one might ask for the street layout (one-way streets, traffic lights, etc.) that maximizes traffic flow in a city. In this example each possible solution may involve adding or removing (discrete) traffic lights at intersections. The numerical aspect here would be the duration and frequency of the green light phase of each traffic light.

3.5 Parallelization

Parallelizing an application means identifying those parts of a program that can be executed in parallel without creating an overhead that destroys any possible gain in processing time. The need for synchronization is the single most important aspect that can make parallelizing a program infeasible. Whenever a parallel thread of execution[6] needs to exchange data with another thread, both have to synchronize. Usually this means that one of them has to wait until the other reaches the synchronization point. A similar problem applies to the concurrent usage of resources that do not allow simultaneous access. An important example is the write access to global variables of a program. One thread of execution needs to hold a *lock* to the resource such that the other cannot access it while it is still being used. Acquiring and releasing this lock consumes compute time, furthermore the other thread of execution might have to remain idle until the first one releases the lock. All this will slow down the program flow. In extreme cases, a serial version of a program can then be several times faster than the parallel version.

In Evolutionary Algorithms, the duplication or recombination of parents may involve communication between individuals, if a "sexual" recombination scheme is being used. Selection of the best individual of a generation again requires a comparison of their fitness. As a consequence, both parts of the algorithm are not easily accessible to parallelization. Mutation of child individuals and the subsequent calculation of their fitness, however, usually does not involve the need for communication between them. This part of the algorithm is thus an outstanding candidate for parallelization. The implementation of parallel Evolutionary Algorithms presented in this thesis is thus targeted at problems, in which the fitness calculation for each individual is computationally expensive. Naturally, the speedup of parallel execution is the higher the longer the parallel part of the execution is compared to the sequential part. Fortunately, most problems in particle physics require processing of large datasets in order to determine the value of an individual and will therefore benefit from an implementation of parallel Evolutionary Algorithms.

Our approach to parallelization is discussed in more detail in chapter 4. Further, more

[6] Here the term "thread of execution" means "parallel part of a program" and is not limited to a specific thread implementation such as POSIX threads. The remarks also apply to programs using the **M**essage **P**assing **I**nterface ("MPI")

general information about distributed and parallel computing can also be found in chapter 4.2.

3.6 Evolutionary Algorithms and the Grid

It is the duty of a *middleware* to provide the usual functionality available in Grid environments. A middleware is a software layer arbitrating between the Grid and user-applications. On top of the middleware lie implementations of software libraries providing standardized APIs[7] for distributed computation. One of the most well known examples of such a library is MPICH-G2, an implementation of the MPI Message Passing Interface (see also [MPICH-G2]). As it is not the API that has to adapt to a Grid environment but the underlying implementation, there is in principle no difference in executing a parallel MPI program over the Grid compared to the execution on a local compute cluster. However, one has to take into account the significantly higher latency (see section 4.2.4) involved in transferring data over long distance connections, and higher requirements for error-recovery procedures apply, as one has less control over the availability and stability of remote resources.

[7] API=Application Programming Interface

Chapter 4

The EVA Library

This chapter discusses the design and implementation of "EVA", a software library of parallel **EV**olutionary **A**lgorithms. The design principles are followed by an in-depth discussion of the most important classes of the library. The chapter is concluded by a number of performance measurements, most of them being taken at the GridKa compute center of Forschungszentrum Karlsruhe. The EVA library is a tool that aims at solving the problems associated with the optimization of computationally expensive physics problems The software was developed as part of this thesis. Section A.3 in the appendix is meant to complement this chapter by giving an overview of EVA's user-interface, using programming examples.

4.1 Introduction

The EVA library implements common features of Evolutionary Strategies ("ES", see 3.3) and Genetic Algorithms ("GA", see 3.4)[1]. An implementation of both GA and ES is built on top of this code base.

The design focus was on long lasting, computationally expensive optimization studies. The main design goals included ease of use, transparent parallelizability and extensibility. While being designed with the needs of the particle physics community in mind, the library should be general enough to allow a broader audience to benefit from its features.

This introduction to the EVA library assumes prior knowledge of the inner workings of Evolutionary Algorithms, as discussed in [EAFAQ], [HOLL75], [RECH94] and [SCHO93]. Chapter 3 also gives an overview of this topic.

[1] GA and ES are commonly referred to as "**EV**olutionary **A**lgorithms" or short "EA" in this thesis. This term is also the origin of the name of this software library, EVA

Target Group

Particle physics experiments today show the need to process large amounts of data. The upcoming LHC experiments at CERN/Geneva, as an example, will produce several petabytes[2] of "event" data[3] per year as of 2007. The *BABAR* experiment at SLAC / Stanford, in contrast, is already taking data and faces similar, albeit smaller scale, challenges.

In this environment numerous optimization tasks have to be performed that involve reprocessing large amounts of data in an iterative procedure. One example are parametric optimizations in particle physics analysis (see e.g. section 5.2). Processing such a data set can involve several hours or even days of computation. Checking several parameter sets can thus often be done simultaneously on several machines, if the computational requirements are available. First and foremost this means that a suitable program library capable of doing parallel computation must exist. The main target group of the EVA library are therefore scientists and commercial ventures with a need for long lasting, computationally expensive optimization studies.

4.2 Technical principles of parallel execution

In order to allow for a better understanding of EVA's classes introduced below and to help understand the performance measurements presented at the end of this chapter, a number of general principles governing parallel execution of computer programs are discussed below, starting with the definition of two variables measuring how effective the parallelization of a program has been.

4.2.1 Efficiency and Speedup

If p is the number of processes and n the "input size"[4], then the *speedup* of a parallel program can be defined as the ratio of the runtime of a sequential solution ($T_\sigma(n)$) to the runtime of the corresponding parallel implementation ($T_\pi(n,p)$) :

$$S(n,p) = \frac{T_\sigma(n)}{T_\pi(n,p)} \tag{4.1}$$

An alternative to speedup is the *efficiency*, defined by

$$E(n,p) = \frac{S(n,p)}{p} = \frac{T_\sigma(n)}{pT_\pi(n,p)} \tag{4.2}$$

[2] 1 Petabyte = 1000 Terabyte = 10^6 Gigabyte

[3] An "event" being the recorded results of the collision of two elementary particles

[4] A quantity representing the amount of input, e.g. the amount of numbers to be sorted or the number of individuals in an Evolutionary Algorithm

As $0 < S(n,p) \leq p$, efficiency can reach a maximum of $1 : 0 < E(n,p) \leq 1$. An efficiency of 1 represents a linear speedup while in the case of $E(n,p) < 1/p$ the program is slower than the sequential solution.

In order to optimize the speedup of a parallel solution, one should note that for parallel execution $T_\pi(n,p) = T_{calc}(n,p) + T_{i/o}(n,p) + T_{comm}(n,p)$, where T_{calc} is the computing time needed, $T_{i/o}$ is the time needed for input and output and T_{comm} is the time needed for communication (i.e. networking). The latter depends on the latency and bandwidth of a network. So altogether one can say that

$$S(n,p) = \frac{T_{\sigma,calc}(n) + T_{\sigma,i/o}(n)}{T_{\pi,calc}(n,p) + T_{\pi,i/o}(n,p) + T_{\pi,comm}(n,p)} \tag{4.3}$$

It becomes immediately clear that, if $T_{calc} := T_{\sigma,calc}(n) \cong T_{\pi,calc}(n,p)$ *and* $T_{i/o} := T_{\sigma,i/o}(n) \cong T_{\pi,i/o}(n,p)$*, then one will only achieve a satisfactory speedup if* $T_{calc} + T_{i/o} \gg T_{\pi,comm}$*.*

This calculation does not yet take into account the need for synchronization (see section 4.2.3) between sub-processes, which leads to a further delay, decreasing the speedup.

4.2.2 Amdahl's Law

An upper limit to the speedup is commonly calculated using Amdahl's law. If F is the fraction of a calculation that is sequential, and $(1 - F)$ the fraction that can be split into P parallel code sequences (and each is executed on different computing devices), then the maximum speedup that can be achieved is

$$S_{max} = \frac{1}{F + \frac{1-F}{P}} \tag{4.4}$$

Consequently, if P tends to infinity, the maximum speedup becomes $\frac{1}{F}$. Consequently, if 20% of the code is sequential and the number of parallel execution units is sufficiently large, then the maximum speedup is 5.

4.2.3 Synchronization

One of the biggest problems of parallel computing is the need to synchronize parallel entities of a program running on a cluster or as threads on multiple processors of a local machine. The problem will be discussed using the example of a multi-threaded application, although the problems are the same for MPI[5] and PVM[6] programs.

[5]The **M**essage **P**assing **I**nterface (MPI) is the most common way of exchanging information and data between the different client entities and the server of a parallel program running on a cluster of compute nodes.

[6]**The P**arallel **V**irtual **M**achine (PVM) is a library of functions similar in scope to MPI.

Often, a thread cannot continue its work without data emitted by another thread. In this case it has to wait until the second thread makes this data available. Such an exchange is often done at synchronization points, i.e. sections dedicated to the exchange of data. A similar situation occurs, when one thread cannot start execution, before another hasn't finished its job.

The need for synchronization limits the speedup that can be achieved through parallel execution, as portions of the program have to remain idle part of the time. Parallel programs can also be very difficult to design, as the very fact that different portions of the code are executed in parallel can make it cumbersome to map an algorithm to its implementation.

Furthermore, synchronization implies the danger of a dead-lock, where one thread might wait for a second to finish its work, which in turn waits for the first one to send some data. The two threads will then just stop executing and might halt the program execution altogether. Such dead-locks can span more than two threads or processes and are sometimes extremely difficult to find. Parallel programs that do not need to synchronize between distributed entities are sometimes also called "embarrassingly parallel"[7].

4.2.4 Latency and Bandwidth

The most important difference between the execution of a program over a network, as compared to its execution on a multi-processor machine, lies in a networks comparatively low "speed". This "speed" is a complex variable. It consists of the bandwidth (the number of bits received per second on one end of the network) and the latency (the amount of time it has taken these bits to travel from the source to the recipient. While you can today scale the bandwidth of a network connection to virtually any level - provided you can pay for it - there are physical limits to its latency.

Data cannot travel faster than the speed of light. So there is a lower limit to the amount of time needed to transfer data, no matter how sophisticated the network hardware is. But since this data will have to pass repeaters and routers along the way, the actual latency will be much higher than the physical limit. E.g., the latency across the USA is in the range of 50 msec. Still, this is not a very large value. As a comparison, a modern IDE hard drive with 7200 RPM has mean access times in the range of 8.5 msec. While latency does form a limiting factor, and will continue to do so in the foreseeable future, network latency is already in the range of the mean access times of old MFM hard drives.

In local compute clusters, the limiting factor is more often the bandwidth of the interconnecting network. A compute node cannot get the data to be processed as fast as it can calculate the output. In a Grid environment – a globally interconnected set of compute ressources coupled through standard interfaces – the limiting factor is more often the latency.

[7] Or, more recently, "nicely parallel"

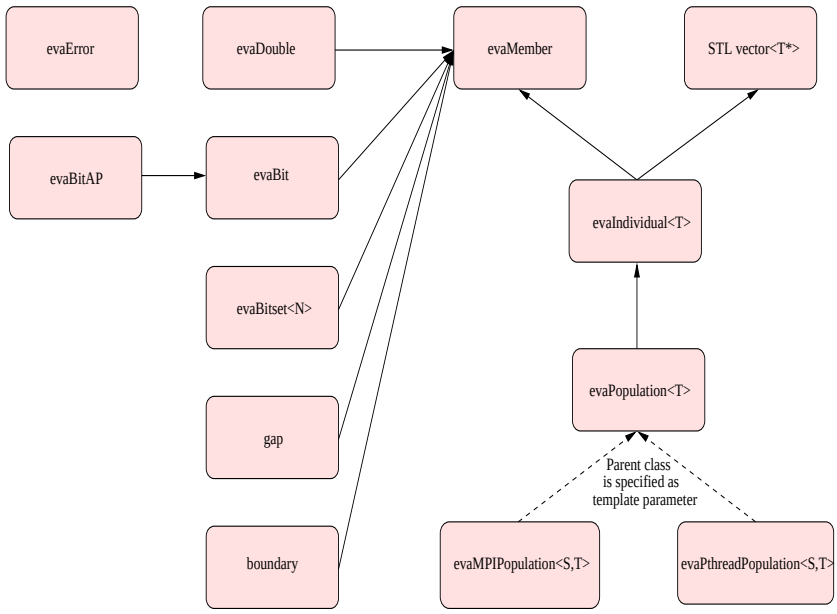

Figure 4.1: *The most important classes of the EVA library. Every C++ class that can be stored in one of EVA's population classes inhertits features from* `evaMember`*. An* `evaIndividual` *is a "member" class that can contain "value classes" like* `evaDouble`*. Population classes inherit many features from* `evaIndividual`*, thus enabling them to be stored in a population themselves and to compete with each other.*

4.3 Library Design

We will now continue with a discussion of EVA's design, including its objectives and a "mission statement".

EVA is written in C++. An overview of the most important classes is shown in picture 4.1. They will be discussed below. Please also refer to the EVA Reference Manual for a full description of all classes including their member functions and data structures. Chapter A in the appendix contains programming examples that further highlight the features discussed below.

4.3.1 Objectives and Mission Statement

> To create, within the Open Source community, a leading implementation of Evolutionary Strategies and Genetic Algorithms for long lasting, computationally expensive optimization studies, suitable for deployment in computing environments ranging from single processor machines to the World Wide Grid.

EVA does not want to replace other, existing implementations of Evolutionary Algorithms but wants to complement them with a version specifically designed for long lasting studies in particle physics rather than computations with critical (possibly real-)time requirements. In contrast to many other implementations it also does not primarily intend to be a platform

for new techniques in EA research[8] but to help users running real-world applications.

The explicit mentioning of the Open Source community in the mission statement above is meant as an invitation to jointly extend the library, such that it fits the needs of an as wide audience as possible, thus making it "a leading implementation".

4.3.2 Design Criteria, Requirements and Characteristics

A closer description of the criteria and requirements that have influenced the design decisions related to the EVA library will be presented in this section.

- **Long lasting optimization studies :** It is assumed that the computation needed for the user-defined quality assessment of an individual takes sufficiently long such that the compute time needed by EVA's core routines doesn't account for a large portion of the overall computation. A significant speedup is then achieved much more through parallel execution of user-defined code than via a high-performance design of the library itself. Code that doesn't need to take too much care of computational effectiveness can however be designed in a way that directly benefits other design considerations, namely ease of use, extensibility and features.

- **Parallelizability :** A parallelization framework is a key requirement. Indeed, in most cases quality assessment of individuals is atomic. Determining the fitness of one individual usually doesn't require interaction with other individuals. Due to the many problems involved in debugging parallel code, another key requirement is that user modules must be embeddable, without changes, also into a sequential version of the library. So the main effort in the parallelization of the EVA library has to go into a design that allows seamless transition from sequential to parallel execution. Only minimal changes in the core (sequential) code of the library should be necessary in order to achieve parallel execution, so a more bug-free and manageable code-base can be achieved. An overview of the general principles of parallel execution is given in section 4.2.

- **Ease of use :** Users with a passing familiarity of Evolutionary Algorithms should be able to use the library. However, at least a basic knowledge of "advanced" C++ constructs such as templates must be assumed. No knowledge about parallel programming is required.

- **Extensibility :** An object-oriented design of the library together with templatized code should help advanced users to easily extend existing classes of the library and thus adapt the behavior of the library to their special needs.

[8] ...although it could be used for that purpose thanks to its extensibility

- **Code Re-Use and Open Source :** Wherever possible, existing code and libraries should be re-used. To name a few examples, XML parsing and generation, multi-threaded design and container classes can be based on existing implementations, such as the GNOME foundation's libxml2, the POSIX thread implementation of Linux systems, or the Standard Template Library (STL). Reusing code would not be possible without the "Open Source" revolution that has made a huge code base available to the general public free of charge. Reusing such components, however, means that one has to adhere to their licenses, namely the GNU General Public License (GPL, see [GPL]). As a direct consequence, the EVA library itself is published under the GPL.

- **Portability :** It must be a prominent goal of the development of any larger software package to make it as portable as possible. In the early stages of development and after the initial releases, however, it is a common procedure to focus more on the establishment of a given feature set than on portability. The EVA library has indeed been developed with portability in mind, not the least by integrating existing, portable software libraries wherever possible. A conscious decision has been taken, however, to initially support only a single platform, and more work goes into the features of the library than into portability. Realistically this means, that at the moment the EVA will only be usable on the Linux/x86 platform. Porting EVA to other platforms will become an important goal in the medium term future, once EVA has established itself as a proven solution for distributed Evolutionary Algorithms.

- **General Purpose :** As few assumptions as possible about the subject of optimization have been made. However, as execution speed of library code was regarded as less important than ease of use, extensibility and features, EVA is less suitable for time-critical computations (e.g. compuations with real-time requirements).

- **Ease of installation :** Installing the EVA library should not require prior knowledge of its inner workings. So far, the best choice for the configuration and installation stage are the GNU `autoconf` and `automake` tools. It is thus a requirement to use these tools throughout the project.

4.4 C++ classes of the EVA library

EVA has a clear class structure that will be discussed below.

4.4.1 The evaMember class

In EVA terminology, every object that can be stored in one of EVA's container classes is a "member" (this term will be used as a synonym for the corresponding classes in the

following[9]). All member classes derive from the `evaMember` class.

`evaMember` also specifies the common API of these classes. Some of these interface functions are specific to every class and should be re-implemented separately for each of them. In the following, such functionality is marked with the symbol [R]. In very simple cases it is not necessary to provide all of them (see e.g. the `parabola` example in section A.3.2), but implementing these functions is recommended anyway.

A member derived from `evaMember` can be mutated (handled by the `void evaMember::mutate(void)` function) and its value or fitness can be calculated (handled by `double evaMember::value(void)`). Furthermore, a member must be able to emit information about its state (triggered via `doInfo()` [R]) and in some situations it must be able to clear its internal data structures (and recursively those of the classes it is derived from). This functionality is handled via the `clearall() [R]` function.

Each EVA member has the ability to be converted into an XML representation of its data structures and to load this data back into them. This important functionality, described in more detail in section A.3.4, is handled by `void loadFromXMLCore(xmlNodePtr)` [R] and `string convertToXMLCore(int,int)` [R]. It is used to transmit information about individuals over a network when using the parallel MPI-personality of the EVA library and to save and reload members to and from disk.

During the optimization process, members are repeatedly polled for their value, e.g. while they are sorted according to their fitness. Since, following its design goals, the EVA library is targeted at long running, computationally expensive optimization studies, it is of vital importance to only recalculate the value of a member, if its internal data structures contributing to the fitness have changed. The `evaMember` class therefore implements a caching mechanism accessible to all classes derived from it.

Changes of the data structures are almost always triggered by the assembly of a child individual from one or more parents or by the mutation of these children. Whenever this happens, a boolean variable `dirtyFlag`, private to `evaMember`, is set. `double evaMember::value(void)` itself is just a wrapper function for `virtual double customValue(void) = 0` [R], i.e. a purely virtual function that must be specified by a programmer in his derived classes. `value()` first checks, whether `dirtyFlag` is set, and if so, calls `customValue()`. Otherwise the stored value from the last fitness calculation is returned. Just like `value()` , `void evaMember::mutate(void)` is just a wrapper function for another, purely virtual, function, named `customMutate(void)` [R]. It must be specified by the user. The main duty of `mutate()` is to set `dirtyFlag` as soon as the mutation process is finished. and to trigger the recalculation of the individual's fitness.

[9] It is unrelated to the term "member function"

To summarize the above : Classes deriving from evaMember should at least implement the functions `double customValue(void)`, `void customMutate(void)`, `void loadFromXMLCore(xmlNodePtr)`, `string convertToXMLCore(int,int)` and `void clearall(void)` in order for EVA to work properly. Additional functions might be needed in special cases.

This shouldn't be a big constraint for users, as they only deal with an implementation of an individual designed to describe their optimization problem. This individual itself is derived from a set of classes that provide many utility functions facilitating the creation of the functions mentioned above. Predefined versions for common cases (e.g. : Gauss mutation in the case of Evolutionary Strategies) are available for a user to choose from. But, as EVA does not make any assumption about the design and data content of an individual, it is not possible to provide generic versions of these functions for all possible cases. The sections 4.4.3 and A.3.2 describe the process of creating an individual in more detail and it will become obvious that the creation of a new individual class is quite easy in the majority of cases.

None of the criteria and functionality discussed above are specific to either Evolutionary Strategies or Genetic Algorithms. They can be used in conjunction with both of them.

4.4.2 Values

In EVA terminology, a value is a member class (i.e., it is derived from `evaMember`) and resembles one of the standard C/C++ types. Currently available are the `evaDouble` class implementing a `double` and the `evaBit` implementing a boolean value. This infrastructure provides the necessary framework to implement both Evolutionary Strategies and Genetic Algorithms[10].

The reason for basic types to be reimplemented as classes is the functionality required for each value. E.g., like with any other member, it must be possible to mutate them. In the case of `evaDouble` this can be achieved by adding a random number with a gaussian probability density function (PDF)[11] to it. In the case of `evaBit` "mutation" means changing its value from `true` to `false` or vice versa. A value class knows about the appropriate mutations itself. Another example is the transformation to XML, which is best handled by the class itself. From the point of view of an object oriented programmer, it is much more natural to have code to modify data accompany that data as part of the same class rather than to use stand-alone functions.

It should be noted that, as an Evolutionary Algorithm naturally contains a multitude of values as part of each individual, a non-negligible overhead is incurred through this design,

[10] A more convenient GA class might be `evaBitset`, see section 4.4.4 for further information

[11] Henceforth this will be called a "gaussian random number"

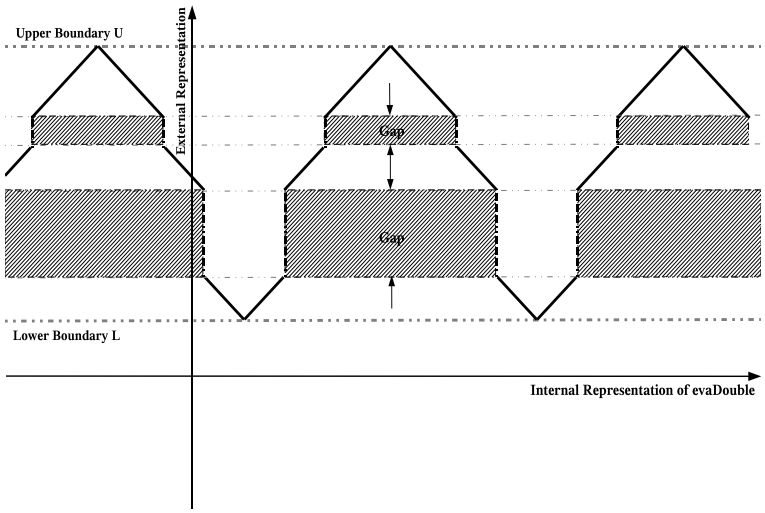

Figure 4.2: *Gaps and boundaries in the value range of a floating point variable are achieved by a transformation from an internal to a user-visible value in the* `evaDouble` *class. All mutations known to* `evaDouble` *are applied to the internal representation rather than the user-visible value. This helps to reduce code-complexity while retaining the ability to use an arbitrary amount of gaps and open or closed boundaries.*

compared to a "bare-bones" implementation using only basic C++ types. However, a significant simplification in the class tree structure could be achieved this way (see figure 4.1), as all members, from values to populations, share the same API. For a templatized individual or a population[12] there is therefore no difference, whether they contain values, individuals or populations. Sophisticated features like multi-populations[13] therefore come at no additional cost.

Since EVA's focus is on long lasting optimization studies – with long lasting calculation of an individual's fitness – the overhead incurred by this design decision doesn't play a big role, and it was possible to greatly simplify the code, thus making it less error-prone and easier to understand.

The `evaDouble` class, as an implementation of a "value" class, will now be discussed in more detail. As was said above, `evaDouble` implements the functionality of a 64 bit floating point number (a "double" value).

Using an evaDouble

The `evaDouble` class comes with a full set of utility functions implementing common operations such as multiplication, division, addition, subtraction and various kinds of comparisons with other (real or class) double values. The one difference in using an `evaDouble` is that in order to find out its value one has to use the `value()` function. As an example, in order to calculate the sine value of an evaDouble, one has to use a construct similar to

[12] Both are container classes, see below

[13] populations containing competing populations as members, again containing any possible member, etc.

```
// [...]
evaDouble x=1.,y;
y=sin(x.value());
// [...]
```

Mutating an evaDouble

The `evaDouble` class is mainly used in Evolutionary Strategies[14]. The most common mutation scheme for double values in ES is to add a gaussian random number to it. The generation of such random numbers is discussed in detail in section A.4.2. The `evaDouble` class is meant as a fully usable implementation of double values in a class framework and thus already implements its own mutation scheme (using the `customMutate()` function and `evaMember`'s value caching mechanism). All that needs to be done in order to mutate an evaDouble is therefore to issue a call to `evaDouble::mutate()`. This function is really implemented in `evaMember` and takes care that the value of the object is recalculated.

Gaps and boundaries in the value range of an evaDouble

As users frequently have to deal with variables that are only defined within upper and/or lower bounds, it is necessary to be able to restrict the values an evaDouble can assume. However, varying a floating point variable only within a given range is difficult to achieve (albeit not impossible), if this variation is done by adding a gaussian random number to its value. The problem gets worse if one of the key requirements for a variable is that it not only has upper and lower boundaries, but that there also may be intermittent gaps in its value range. An example might be a variable that may not assume values close to 0 due to a singularity in the fitness function of an individual. See section A.3.3 for an example involving the restriction of the value range using a gap.

A simple value transformation solves this problem. Internal values are mapped to their external, user-visible representation using a function similar to the one shown in picture 4.2 for a variable with a lower and upper bound and two gaps. The continuous internal representation of an evaDouble greatly facilitates the implementation and application of mutations, while the user "sees" a variable with a value range that is restricted according to his requirements. Any number of non-overlapping gaps is allowed, with global upper and lower bounds to a variable, if needed. Variable with only an upper or lower bound or no bounds and gaps at all are of course possible. A big advantage of this implementation, as compared to the more standard transformation using trigonometric functions (as is done e.g. in the ROOT framework's `TMinuit` implementation of minimization functions) is that only linear transformations, shifts and changes of signage are involved. This means that a higher accuracy can be achieved.

[14] Although they also serve a purpose in the `evaBitAP` class, which implements a bit value with dynamically adapted flip probability

4.4.3 The evaIndividual class

In EVA terminology, an "individual" is every class entity that can compete with other entities of identical structure and complies to the `evaMember` class API (i.e., is derived from it in the most common case). It is usually a container class holding other basic values or objects. Thus an individual would, in the context of Evolutionary Algorithms, be called a "gene".

The `evaIndividual<T>` class is an implementation of an EVA individual. However, an `evaIndividual<T>` is much more than just a container for objects such as `evaDouble`'s (in the case of Evolutionary Strategies) or `evaBit`'s (in the case of Genetic Algorithms). Being derived from the Standard Template Library's `vector<T>` class[15], it can hold all class objects that are derived directly or indirectly from the `evaMember` class. As the `evaIndividual<T>` class itself is derived from `evaMember` and implements its API, an `evaIndividual<T>` can hold other `evaIndividual<T>`s. This adds enormous flexibility. E.g., an individual implementing a feedforward neural network could contain a container of `evaDouble` values for each layer of a network, each container being just another `evaIndividual<T>` (see section A.4.4)..

A consequence of the decision to make the `evaIndividual<T>` class a derivative of the STL `vector<T>` class is that the data contained in it must be implemented through the use of classes. E.g., if one wants an individual to hold a set of double values, it is not possible to simply use an `evaIndividual<double>`, the reason being that the template argument of an `evaIndividual<T>` must comply to the `evaMember` API. While this adds some computational overhead, this isn't much of a problem in the context of Evolutionary Strategies. After all, there is information associated to each double in an Evolutionary Strategy, because the most common mutation is the Gauss mutation (see section 4.4.2).

Being derived from it, an `evaIndividual<T>` has the full STL `vector<T>` API. As a consequence, users can enjoy a familiar API when setting up an individual.

The size of an individual can be altered at run time, should a task need a variable number of parameters. Please note that a variable size of an individual cannot be used, if the "shift" functionality is needed in conjunction with multi-populations.

Pointers are being used for performance reasons for the members stored in an `evaIndividual<T>`.

4.4.4 The evaBit, evaBitAP and evaBitset classes

The `evaBitset` class implements a basic GA framework for the EVA library. It is an an extension to the STL class `bitset<N>`. The interface of this class is to a large extent identical to the `evaIndividual` class and also shares a large portion of its algorithms. The

[15] `evaIndividual<T>` is actually derived using "`:public vector<T *>`", so `evaIndividual<T>` contains pointers to the objects stored in the vector.

STL bitset<N> already adds quite a bit of functionality to the evaBitset<N> class, such as the ability to flip single bits. A STL bitset<N> is set up using a template parameter N that specifies its size, and evaBitset<N> shares this convention. Setting up an evaBitset<N> can thus be done similar to evaBitset<64> largebs, for a bitset with 64 bits[16]. See a manual of the STL library for further information about the STL bitset<N> class.

An alternative approach to evaBitset<N> for the implementation of Genetic Algorithms are the evaBit and evaBitAP classes. evaBit is derived from evaMember and can thus be stored in an evaIndividual<T>. A "gene" in a GA application could therefore be implemented using an evaIndividual<evaBit>. While this would certainly be a more elegant approach than the evaBitset<N>, it also causes a lot of computational and memory overhead. A class like evaIndividual<T>, but derived from the STL bitset<N> is therefore much more suitable to large scale problems requiring a high performance implementation.

evaBitAP is a class that is derived from evaBit. It implements a dynamic adaptation of its flip probability using an evaDouble.

Like an evaDouble, evaBit and evaBitAP are "value" types. In contrast to this, an evaBitset<N> is a full-scale individual.

4.4.5 The evaPopulation class

In EVA terminology, a population is an assembly of objects of same type that can compete against each other. The evaPopulation<T> class assumes that each of these objects has the evaMember API (most notably : its value can be calculated using the value() function and it can be mutated using the mutate() call). An evaPopulation<T> is thus very similar to the evaIndividual<T> class (see section 4.4.3), and indeed it is derived from it. evaPopulation<T> adds a framework to duplicate or recombine, mutate and sort its member objects. As evaPopulation<T> can store any object that has the evaMember API and, as evaPopulation<T> itself implements it, an evaPopulation<T> can store other populations and let them compete against each other. This feature is called a "multi-population". It can be used to explore different regions of the parameter space using different populations.

No difference is made between Genetic Algorithms and Evolutionary Strategies on the level of an evaPopulation<T>.

4.4.6 The evaPthreadPopulation and evaMPIPopulation classes

evaPthreadPopulation<S,T> provides seamless parallelization of Evolutionary Algorithms based on the implementation of evaPopulation<T>. Since parent classes are specified by a template parameter, customized versions of evaPopulation<T> can be paral-

[16]This can be used to implement a double class based on its bit representation

lelized. `evaPthreadPopulation<S,T>` is targeted at multi processor machines, with SMP machines being the most likely platform. g++ can't handle separate compilation of template functions, so the implementation of `evaPthreadPopulation<S,T>` can be found in the header file. The usage of this class is almost identical to `evaPopulation<T>`, as the majority of changes is confined to overloaded versions of the `variate()` and `optimize()` member functions of `evaPthreadPopulation<S,T>`. This also ensures that users do not have to care for the details of the parallelization, but can continue to use the individuals they have already used for populations with sequential execution.

`evaMPIPopulation<S,T>` is, from a users perspective, mostly identical in usage to `evaPthreadPopulation<S,T>`. The only real difference is that its constructors need to get access to the command line arguments using the main() function's `int argc` and `char **argv` parameters.

Communication with XML

The EVA library uses the help of the GNOME foundations `libxml2` library to parse XML descriptions of classes derived from `evaMember` and is able to convert any of the core XML objects to XML. This functionality also forms the basis for the parallelization of the EVA library using the "Message Passing Interface" (MPI). A server creates XML descriptions of individuals that need to be mutated and evaluated, which is done by remote client processes after parsing and loading the XML descriptions of these individuals. They are subsequently sent back, again in XML format.

4.4.7 The evaError class

This class implements the framework needed for exception handling. An `evaError` class can store information necessary to locate the source of an error, such as the name of the class and the name of the member function that an error occurred in. Furthermore it stores the error message itself. Using its `printErrorMessage()` function it is able to generate a formatted error message from this information.

4.5 Performance Measurements

This section presents a set of performance measurements for the EVA library. It should be emphasized that such measurements depend highly on the test conditions, such as speed and load of the machines being used, as well as network bandwidth and load. As it is in the nature of a random number generator to produce random values, it is possible that your results will differ from the ones presented here. All examples shown use Evolutionary Strategies rather than Genetic Algorithms, due to their importance to this thesis.

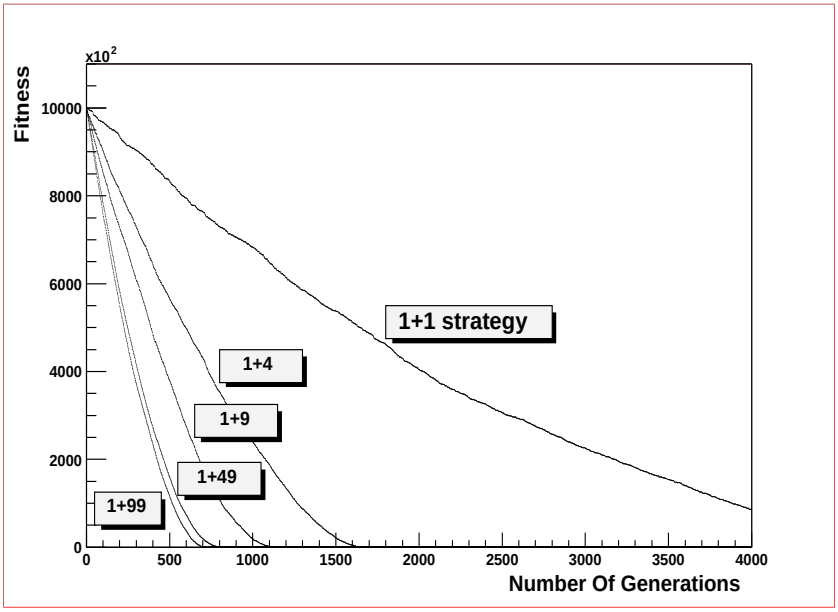

Figure 4.3: *The progress of ES populations of different sizes is shown as a function of generations. The size of a population influences the success rate. A "1+49"population (1 parent, 49 children) achieves better results than a 1+4 strategy. Only little improvement is visible when going from 50 to 100 members. The ideal size of a population depends on the optimization problem.*

4.5.1 Number of members of a population

In this example an Evolutionary Strategy searches for the minimum of a simple parabola. Picture 4.3 shows the impact of the size of a population on this optimization process. The y-value shown is the value of the parabola as a function of the input parameter. Using a strategy with just one parent and one child ("1+1"), only little progress can be seen. Using 5 individuals (1 parent, 4 childs), the optimum at 0 is approached much quicker. Even faster progress can be achieved with $1 + 9$, $1 + 49$ and $1 + 99$ strategies. It can be seen, however, that one doesn't gain much by going from 50 to 100 individuals. This is typical for Evolutionary Strategies. Unless one has hundreds of machines at hand, it is therefore recommended, when using the EVA library, to limit the size of a population. The optimal size for a given problem depends mainly on the complexity of the evaluation function.

4.5.2 Two child individuals with multi-populations

In the next example a multi-population with 1 parent population and two child populations is parallelized using `evaPthreadPopulation<S,T>` and `evaMPIPopulation<S,T>`. Each child population runs in a separate thread or process (for POSIX threads or MPI respectively). The number of generations used for the optimization in each child population can be varied. This is equivalent to varying the length of the parallel calculation, as during the optimization of each child population no data is exchanged between client and server or the threads. Picture 4.4 shows the time needed for the overall optimization as a function of the number of generations in each child population. Only a fit of a linear function to the measured points is shown, as the time needed increases linearly with the

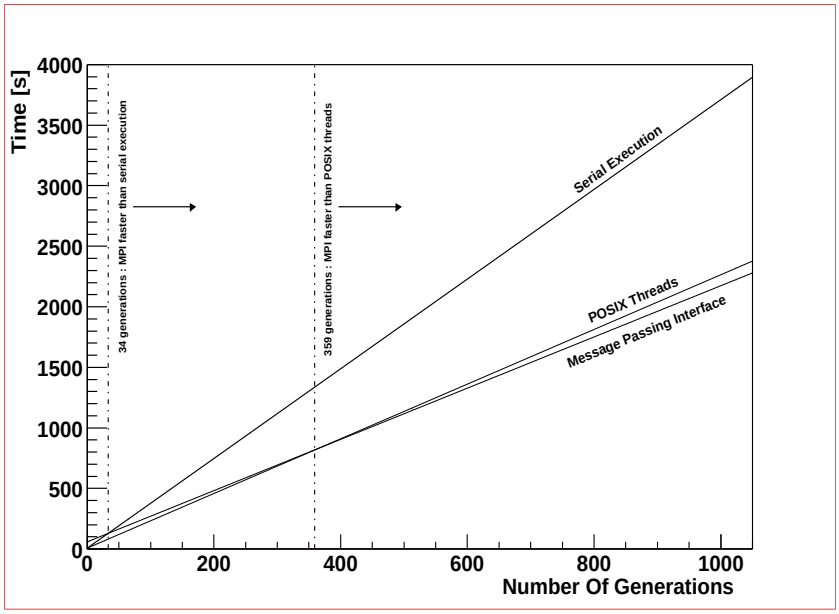

Figure 4.4: *Time needed for the optimization of a multi-population as a function of the number of generations in each child population. Multi-threaded execution and execution on a cluster using the Message Passing interface (MPI) clearly outperform sequential execution.*

number of generations in the child populations.

As expected, the MPI and POSIX thread personalities of the EVA library outperform the sequential execution of the optimization. MPI and POSIX threads are able to achieve a speedup of close to 1.8 over sequential execution for a large number of generations in this example. The speedup mainly depends on the length of the sequential portion of the computation performed by the server in-between two generations – the clients are idle during this time.

Additional tests in a *low-overhead environment* have shown, that with two MPI clients running on different machines or two threads a maximum speedup of almost 2 can be achieved. See section 4.5.3 for further information.

A closer look at the area below 5 generations of plot 4.4 reveals another feature. Picture 4.5 shows that, in this example, the MPI personality has an overhead of roughly 50 seconds over the sequential execution and the POSIX thread personality, that both have a very similar overhead. This is not an unexpected feature, as the time needed to generate and parse XML descriptions of individuals and to send such a description over a network costs time[17]. It is therefore recommended to use the MPI personality only for tasks, where the evaluation part of the calculation takes sufficiently long. Picture 4.4 shows that the MPI personality performs better than sequential execution when the optimization lasts longer than 34 generations in each sub-population. Please note that these values will vary depending on your optimization task.

Picture 4.5 shows some of the time measurements used to create the "linear" fit. It is visible, that they do not match the fit function very well in this particular area. It should

[17]Please note that the XML description of an entire population can have a size beyond one megabyte.

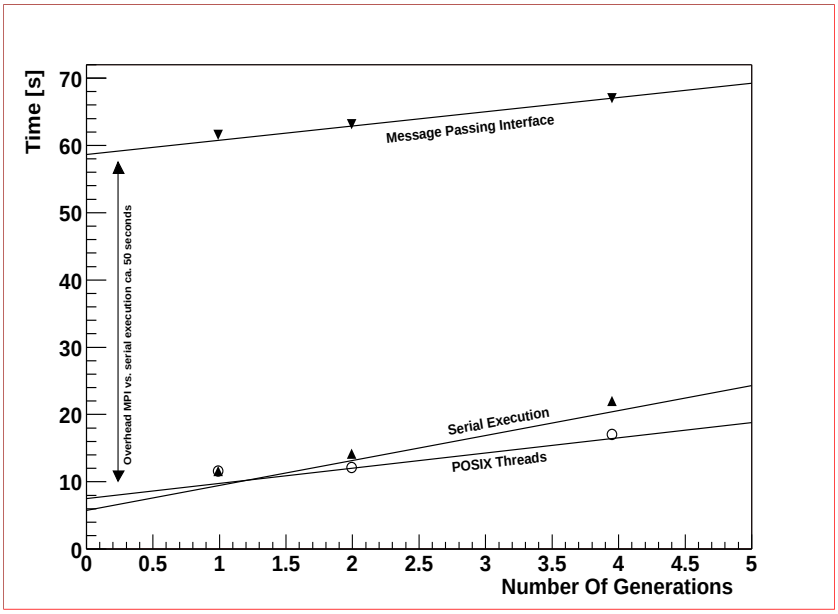

Figure 4.5: *Execution via MPI results in a certain overhead due to the need to parse and generate XML descriptions of population members. Please note that this is an extreme example, as the XML description of a whole population, as is used in multi-populations, can be several megabytes in size. Furthermore, as is shown in 4.4, the slope of the MPI curve is much lower than for sequential execution.*

be noted, however, that the area shown only represents about 0.5% of the area used for the fit and that for a low number of generations unpredictable influences such as the Operating System's service programs running in parallel to the optimization have a large effect.

4.5.3 Two child individuals with scalable time delay

In this example the evaluation function of the individuals contained a call to `usleep(microseconds)`. Using this technique it is possible to arbitrarily scale the length of the parallel calculation. No further calculation was done in these individuals. Each individual was equipped with just 4 `evaDouble` objects, resulting in only a very small overhead compared to the situation described in section 4.5.2. The maximum speedup quickly approaches the theoretical maximum of 2 for both MPI and POSIX Threads, as is shown in picture 4.6.

It is obvious that the overhead of the MPI personality is much higher than for the Posix Thread personality.

4.5.4 MPI and the Grid

From the perspective of a *user* running an MPI program on a Grid, as far as the usage is concerned, there shouldn't be many differences compared to MPI running on a local cluster. It is the MPI library and the middleware (e.g. Globus and/or the EDG[18] framework) whose

[18] **E**uropean **D**ata **G**rid

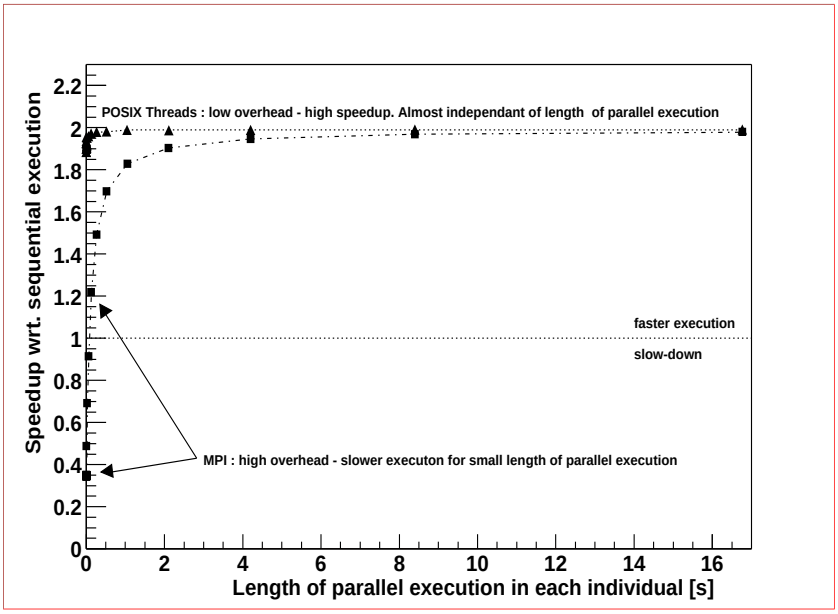

Figure 4.6: *The speedup (multiples of sequential execution) is shown for a population with one parent and two child individuals in an idealized situation (see text) for MPI and POSIX threads as a function of the length of the calculation done by each individual in parallel in each generation. It is visible that the theoretical maximum of 2 is reached within short time.*

task it is to hide the specifics from a user. Various implementations exist that let users run MPI programs over a Grid. The most well-known examples are MPICH-G2 and PACX (see [PACX]). At least in theory another possibility to run MPI programs on a Grid is to submit the MPICH `mpd` daemon as a Globus/EDG job, together with the actual program to be executed (as part of the EDG "sandbox"). Practical tests, however, have shown that there are many problems involved with this approach, apparently related to firewalls blocking ports, but also related to an odd feature of `mpd`, which truncates hostnames, transforming fully qualified names (like e.g. `tau.ep1.ruhr-uni-bochum.de`) into their corresponding base names (e.g. `tau`).

Programmers, on the other hand, should be aware that the latencies involved in running MPI over a WAN[19] are significantly higher than in a LAN[20].

Figure 4.7 shows the round trip times as returned by the `ping` command for 1000 signals, first in a 100 MBit LAN, secondly over a WAN connection between Karlsruhe and Bochum. It is obvious that a factor of 120-150 in the latency will have an impact on program performance. It must be noted, however, that the actual effect of this higher latency is confined to the transfer of data between nodes, and thus depends mainly on the amount of data being transferred and the time spent in calculations in-between data transfers. Running the same program used to gather the data for figure 4.6 over the WAN between Karlsruhe and Bochum has indeed shown that the LAN program had a speed advantage of roughly 20% for a small length of the parallel execution. For longer periods of time in between data transfers the difference between WAN- and LAN-execution almost vanished.

[19] Wide Area Network
[20] Local Area Network

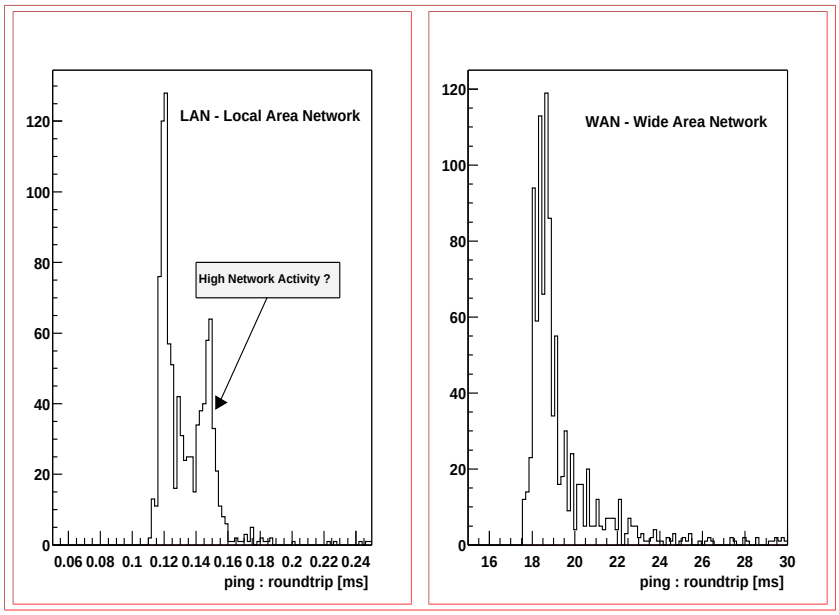

Figure 4.7: *The* `ping` *command sends a signal to another computer in a network and measures the time until the signal is returned by the remote host. This time is a measure of the latency in a network. Latencies in LANs are much lower than in WANs. Latency thus is one of the most important limiting factors for parallel execution over a wide-area network.*

4.5.5 Scaling

In this example a population with 1 parent and 128 children was executed on between 2 and 129 nodes[21]. "timedelay" individuals (as described in section 4.5.3) with a delay of 16 seconds were used. The purpose of this test was to show how well the MPI personality of the EVA library scales to large numbers of nodes.

Picture 4.8 shows the resulting speedup. It is visible, that execution time goes down from 5 hours, 42 minutes in the sequential case to 2 minutes, 50 seconds when running each individual on a separate node.

An interesting feature of this test are the steps. They are a result of the fact that a speedup can only be achieved if the maximum number of individuals per node is decreased. As an example, executing 128 child individuals on 127 nodes means that one node has to execute 2 childs. Adding just one node halves the execution time. Likewise, 64 "client" nodes will lead to the same execution time as 127 nodes.

4.5.6 Tracking an Evolutionary Strategy's success

In this example the Evolutionary Strategy's remarkable ability to find absolute minima even in the presence of local minima will be demonstrated. A test function

$$f(x,y) = (cos(x^2+y^2)+2)(x^2+y^2) \tag{4.5}$$

[21] 2 nodes means 1 server + 1 client executing 128 child individuals

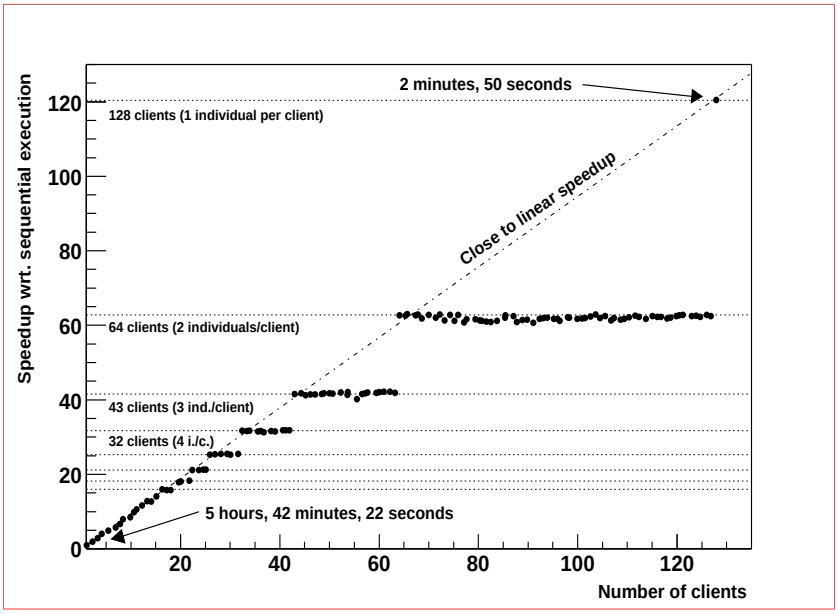

Figure 4.8: *Running an Evolutionary Strategy with 1 parent and 128 child individual in parallel on between 2 and 129 compute nodes exhibits a significant speedup. The execution time is reduced from 5 hours, 42 minutes to below 3 minutes. The "stairs" are a consequence of the fact that a speedup can only occur if the maximum number of individuals per compute node is decreased.*

shown in one dimension in figure 4.9 on $[0 : 10]$ was chosen that, in two dimensions, features concentric valleys around the global minimum at $(0, 0)$. The valleys reappear with increasing frequency with increasing distance from the global minimum. This example uses the ROOT individual discussed in section A.5.

With a start value of $(100, 100)$ and a step width of $\sigma = 1$, all parameter sets tested by EVA in the search for the minimum of equation 4.5 were then displayed in a two-dimensional ROOT histogram (see figure 4.10). The plot also shows a contour plot of equation 4.5[22].

The two parameters x and y of each individual, implemented as `evaDouble`s were given boundaries, such that no higher value than 100 could be assumed. It is obvious from figure 4.10 (upper right corner), that no values higher than $(100, 100)$ were tested by the library.

The algorithm then very quickly converges towards the global minimum at $(0, 0)$ and doesn't seem to be affected by the local minima present in the form of concentric valleys around this point.

Figure 4.10 was created using a rather small step width of $\sigma = 1$. The effect is a very narrow optimization path. Figure 4.11 shows the effect of using a larger step width. The algorithm converges faster and at the same time searches a wider area. On the negative side, once individuals close to the global optimum are found, no significant progress is made anymore, as the step width isn't adapted quickly enough to match the smaller search area. This is visible as an aggregation of individuals around $(0, 0)$.

Another effect can be demonstrated using figures 4.11 and 4.10 : If equation 4.5 would be changed in such a way that an even deeper minimum at, say, $(40, 80)$ would be present, then

[22]Please note, though, that, due to its very fine structures shown in figure 4.9, the formula is not plotted accurately, as a consequence of binning effects.

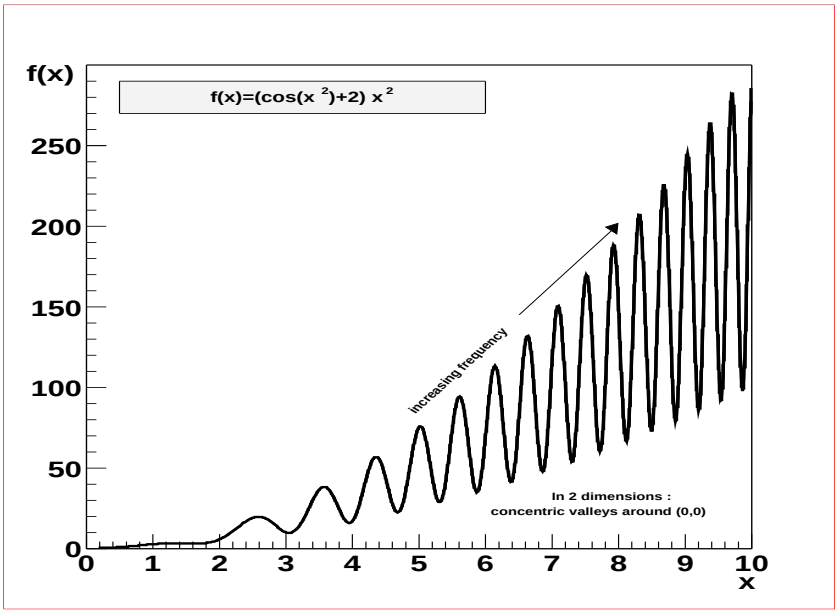

Figure 4.9: *A test function with a large number of local optima can be used to demonstrate the remarkable ability of Evolutionary Strategies to find global minima even in very complicated cases. The one-dimensional function is shown for demonstration purposes – the test was conducted using a two-dimensional version (see figures 4.10 and 4.11)*

it could only be found using a either much larger step width or a much larger population. Neither a step width $\sigma = 1.0$ nor $\sigma = 4.0$ would have been sufficient, the ES would still have converged in the then local optimum at $(0, 0)$. The choice of the step width and the size of the population is thus crucial to the success of an Evolutionary Strategy, at least when using Gauss mutation as the main means of mutation.

In a test, `TMinuit`, an implementation of a gradient descent (see e.g. section 3.3.1) wasn't able to find the global optimum, but got stuck at a very early stage. This highlights again that Evolutionary Strategies are much more suitable than a gradient descent to minimization problems involving local minima.

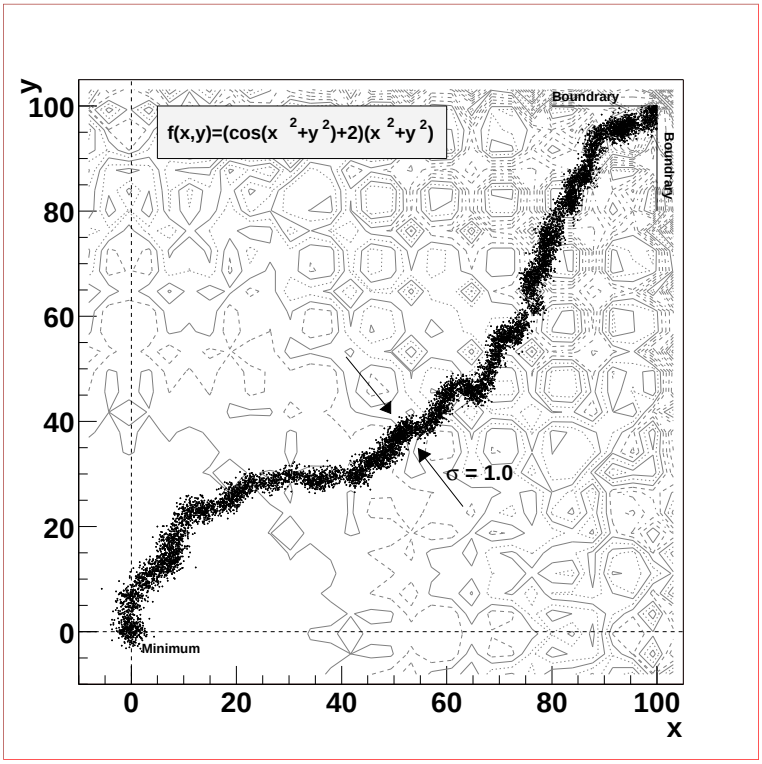

Figure 4.10: *The plot shows all coordinates tested by EVA in the search for the global minimum of equation 4.5. A small step width of 1.0 was used. Both* `evaDouble` *objects of the* `evaIndividual` *class used in this case were restricted in their value range to values below* 100*. It is visible that the Evolutionary Strategy was able to find the global optimum at* $(0, 0)$ *quickly and on a direct way. In comparison, a gradient descent would easily get stuck in a local minima.*

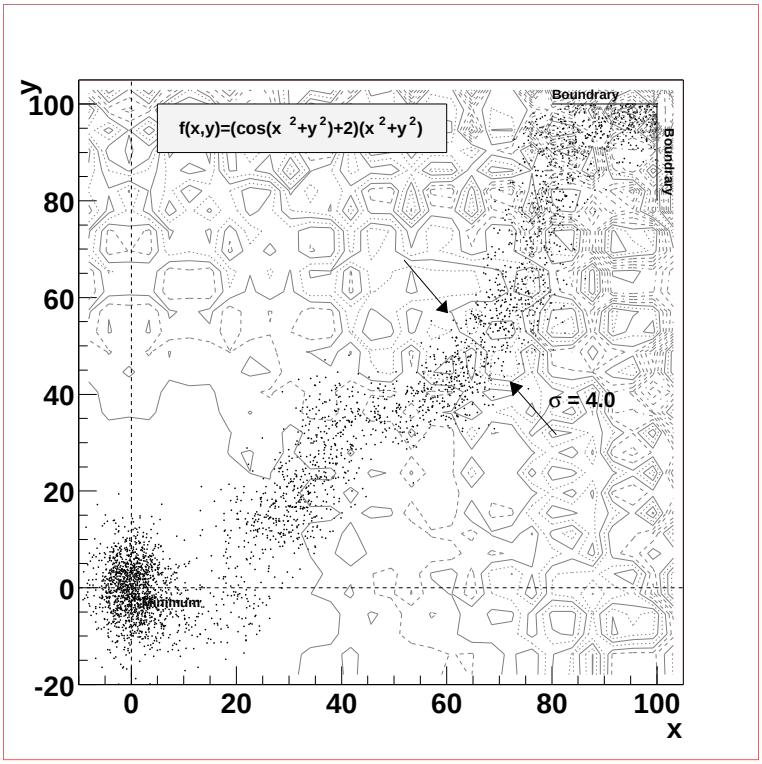

Figure 4.11: *The same optimization was performed as in figure 4.10, but with a much larger step width. It can be seen that the algorithm converges more quickly and that the search path is wider. The algorithm nevertheless has problems to find the global optimum and "hovers" around* $(0, 0)$ *due to the step width being too large. This happens despite the dynamic adaptation of the step width (see text). Choosing an appropriate step width is thus an important task.*

Chapter 5

BABAR Data: Enhancement of Particle Signals

Following the introduction of the *BABAR* detector system, this chapter discusses the application of the EVA library's implementation of *Evolutionary Strategies* to the optimization of a set of cuts in the reconstruction of three decay modes of the $D_s^{\pm}$ in the *BABAR* experiment, including a discussion of the creation of "fake" peaks.

This chapter practically demonstrates an optimization problem that involves very heavy computations, as discussed in section 2.1, and can benefit from distributed optimization techniques like the ones provided by the EVA library.

The discussion of the *BABAR* experiment in the section below is kept intentionally short, as the focus of this chapter is on the application of Evolutionary Strategies on physics optimization problems. Further information on the *BABAR* experiment and the associated accelerator complex can be found in [GANJ02].

5.1 The *BABAR* Experiment

It is the primary goal of the *BABAR* experiment to systematically study CP violation in the decay of B mesons. "CP violation" refers to the fact that, unlike expectations believed to be true up until the early sixties, the laws of physics do not remain unchanged when applying both a charge ("C") conjugation and a parity ("P") transformation. The Application of a parity transformation means changing the sign of spatial coordinates, charge conjugation is the operation that transforms a particle into its anti-particle. CP violation was first observed at a level of 0.2% in 1964 in the decay of neutral K mesons by Jim Cronin and Val Fitch. CP violation is believed to be partly responsible for the apparent lack of anti-matter in the universe. An introduction to the topic of CP violation can e.g. be found in [LHCBCP], [BABARCP] and [ROB98].

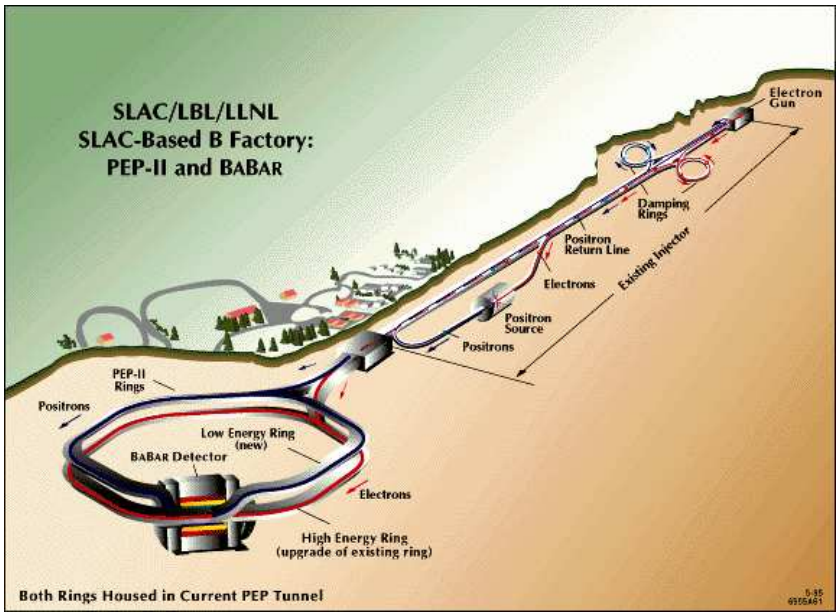

Figure 5.1: *BABAR , PEP-II and the Linear Accelerator for Electrons and Positrons at SLAC. Electrons reach* 9,0 *GeV in the PEP-II synchrotron, positrons have an energy of* 3,1 *GeV, resulting in a center of mass energy of* 10,58 *GeV at the collision point. This is also the energy of the* $\Upsilon(4S)$*, a* $b\bar{b}$ *resonance only insignificantly above the energy threshold needed for the production of* $B\bar{B}$ *pairs.*

One of the reasons for CP violation lies in an asymmetry present in the mixing of quarks, as is observed in the $K^0/\bar{K^0}$ or the $B^0/\bar{B^0}$ system. Within the Standard Model, quark mixing is described by the CKM (Cabibbo-Kobayashi-Maskawa) matrix, a 3x3 complex unitary matrix. This matrix has only 4 independent variables – 2 real numbers A and λ and a complex phase (ρ, η). It is the existence of a complex coupling that is responsible for CP violation.

A direct consequence of the unitarity of the CKM matrix is that it is possible to depict the components of the first and last column as a triangle in a complex plane, leading to a relation between the angles of this triangle and the 4 independent variables A, λ, ρ and η.

[AUBERT1] describes a measurement of the unitarity triangle's β angle using the *BABAR* detector, which can be considered to be the most significant contribution to Particle Physics originated at the *BABAR* experiment so far. The observation of a new, narrow meson decaying to $D_s^+\pi^0$ at a Mass of 2.32 GeV/c^2 (see [AUBERT2]) also has the potential to enhance todays understanding of elementary particle physics.

Before discussing the *BABAR* detector system itself, we will now take a short look at the accelerator complex PEP-II at SLAC[1] / Stanford.

[1] **S**tanford **L**inear **A**ccelerator **C**enter

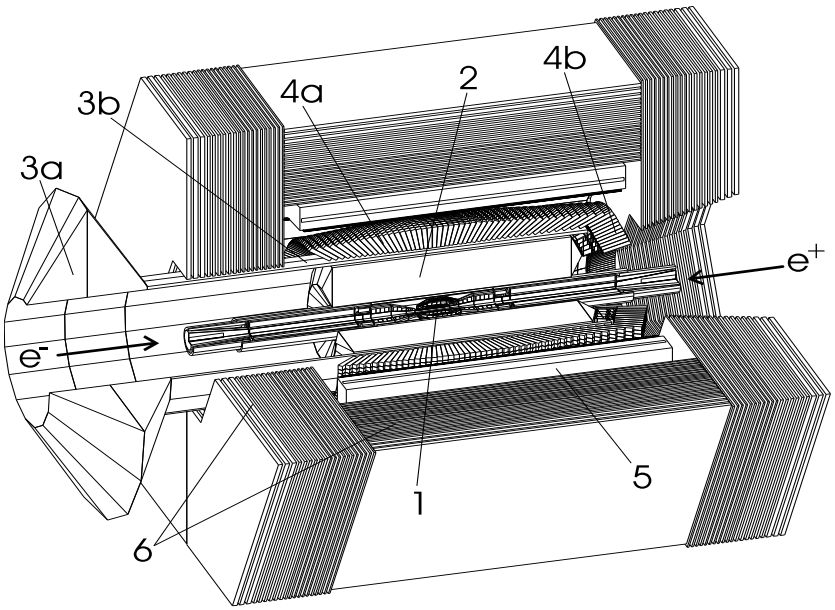

Figure 5.2: *The BABAR detector and its subsystems : (1) vertex detector, (2) drift chamber, (3) Cherenkov detector with (a) read-out (b) quartz crystals, (4) calorimeter with (a) barrel (b) end-cap, (5) superconducting coil and (6) instrumented flux return*

5.1.1 The Asymmetric B-factory PEP-II

The asymmetric PEP-II synchrotron – a descendant of the symmetric PEP ring – at SLAC consists of two separate beam pipes for electrons e^- and positrons e^+. Both particle types are accelerated to different energies, using the linear accelerator, and are then injected into PEP-II. Electrons reach 9,0 GeV, positrons have an energy of 3,1 GeV, resulting in a center of mass energy of 10,58 GeV at the collision point. This is also the energy of the $\Upsilon(4S)$, a $b\bar{b}$ resonance only insignificantly above the energy threshold needed for the production of $B\bar{B}$ pairs. The $\Upsilon(4S)$ resonance is thus a very good source for studying B mesons. See picture 5.1 for an overview of PEP-II and the linear accelerator.

Due to a limited phase space, the B mesons from $\Upsilon(4S) \rightarrow B\overline{B}$ decays are produced almost at rest in the center of mass frame. The asymmetry in the beam energies yields a boost of the B mesons with a significant momentum in the laboratory frame. This enables the measurement of the time-dependant CP asymmetry in the decays of neutral B mesons.

CP violating effects in the B system are expected to be large. But, unfortunately, the decay channels which are interesting for CP violation have a very small branching fraction (of the order of 10^{-4}). Therefore, a large sample of B's (about 10^7) is necessary to perform these measurements.

5.1.2 The *BABAR* detector system

In order to facilitate the understanding of the measurements and optimizations presented in section 5.2, the components of the *BABAR* detector system are discussed below. A good overview of *BABAR* can be found in picture 5.2.

The Silicon Vertex Tracker (SVT)

The Silicon Vertex Tracker (SVT) provides the required vertex resolution for the measurement of the *CP* violation and other decay-time dependant measurements. In addition, when charged particles have a low transverse momentum ($p_T < 120\,\mathrm{MeV}/c$) and cannot be measured by the central drift chamber any more, the SVT is capable of acting as an independent tracker.

The SVT has been designed to minimize multiple scattering. It takes into account the physical constraints imposed by the PEP-II geometry, such as the presence of the permanent magnets close to the interaction point, which are necessary to separate the beams shortly thereafter. The polar angle acceptance θ is $-0.87 < \cos\theta_{lab} < 0.96$ ($-0.95 < \cos\theta_{cm} < 0.87$) and is limited by the beam line elements. A detailed description of the SVT and its components can be found in [SVT].

The SVT consists of five double-sided layers of silicon detectors, mounted on carbon fiber support cones. The first three layers provide the tracking resolution while the last two are necessary to measure low momentum tracks independently of the drift chamber information. The modules are read out by a special, *low-noise*, radiation-hard chip mounted on a passive circuit.

The Driftchamber (DCH)

The main purpose of the Drift Chamber (DCH) is the precise and efficient measurement of charged particle parameters such as momenta and angles, with transverse momenta p_T beyond $120\,\mathrm{MeV}/c$. The DCH complements the information about the impact parameter and the direction of the track measured by the SVT. The reconstruction of decay vertices, for example of K_S^0 particles, outside of the SVT, requires – in addition to the transverse momenta and position – the measurement of the longitudinal position with an accuracy of about 1 mm. The DCH has also to provide the particle identification at relatively low momenta by measuring the ionization loss (dE/dx). Furthermore it must supply information for the charged-particle trigger.

The DCH has an outer radius of 80.9 cm and a length of 280 cm. The interaction point is shifted by 36.7 cm relatively to the center of the chamber in order to improve the forward acceptance, given the asymmetric boost for the $\Upsilon(4S)$ events. The acceptance of the DCH covers the polar angle $-0.92 < \cos\theta_{lab} < 0.96$. 40 layers of small hexagonal cells provide spatial and amplitude measurements. Each hexagonal cell consists of a $20\,\mu$m rhenium-tungsten sense wire operating nominally in the range 1900–1960 V. It is surrounded by 6 cathode wires. The longitudinal position is measured by placing the wires in 24 layers at a small angle with respect to the z-axis.

The amplitude measurement provides the energy loss (dE/dx) used for particle identification at low momenta. The mean dE/dx resolution of 7% provides good π/K separation up to $700\,\mathrm{MeV}/c$.

Each four layers are grouped in one super-layer with the same orientation for sense and field wires within a given super-layer. This structure allows to perform a fast local segment finding as the first step in pattern recognition. The information from all super-layers is included in the level-1 trigger track finding, while only the axial super-layers are involved in the level-1 trigger p_T determination.

The tracking system consisting of the central drift chamber and the vertex tracker is arranged inside a magnetic field of 1.5 T produced by a superconducting magnet.

The Cherenkov Detector (DIRC)

The study of *CP* violation requires the ability to tag the flavor of one of the B mesons and to fully reconstruct the second B decay. The maximal momenta of the kaons used for the tagging of a B via the decay cascade $b \to c \to s$ are about $2\,\mathrm{GeV}/c$. In contrast, the pions and kaons from the rare two-body decays $B^0 \to \pi^+\pi^-, K^+\pi^-$ have momenta between 1.7 and $4.2\,\mathrm{GeV}/c$.

The *Detector of Internally Reflected Cherenkov light* (DIRC) [DIRC] is the principal particle identification system of the *BABAR* detector. Charged particles, produced at the interaction point inside the detector, pass the quartz bars in which Cherenkov radiation is produced. The angle of this radiation (θ_c) with respect to the incoming particles is a measure of their velocity (v). The Cherenkov photons propagate along the rectilinear bars by total internal reflection, preserving the angular information and exiting outside the detector into a large pure water tank, called the standoff box. Using the position of the photo-tubes and the arrival time of the signals, the DIRC can be seen as a three-dimensional imaging device. Since the tracking system provides the track position and the angles, the measured photon propagation angles are used to determine the Cherenkov angle. The arrival time of the signal can be also related to the propagation angles and provides an additional constraint, which can be useful to avoid ambiguities in the signal association due to high background rates.

The DIRC consists of quartz bars inside the detector and the standoff box, supporting photo-multipliers outside the detector at the backward end. The bars are supported by a mechanical structure which is attached to the barrel iron via special structural elements. The water tank is composed of a cylinder, a cone and 12 cylindrical sections. 10752 photo-multipliers are mounted on the sectors placed at about 1.17 meters from the exit point to permit a precise measurement of the angle for each photon. The standoff box is arranged inside a special low magnetic field volume (see [MFM]) which diminishes the value of the fringe fields from the main solenoid.

The Electromagnetic Calorimeter (EMC)

The Electromagnetic Calorimeter (EMC) allows the measurement of electromagnetic showers with high efficiency and good energy and angular resolution. The energy range from

20 MeV to 9 GeV covered by the EMC is required to detect the photons coming from the decay of π^0s and ηs as well as from radiative and electromagnetic processes. The QED processes, like $e^+e^- \to e^+e^-$ (γ) and $e^+e^- \to e^+e^-$, used for the calibration and luminosity measurements set the high limit, while the capability to reconstruct the B mesons in modes containing multiple π^0s and ηs with high efficiency set the low limit.

The Electromagnetic Calorimeter (EMC) covers polar angles of $-0.78 < \cos\theta_{lab} < 0.96$. It contains 6580 CsI crystals doped with thallium. Each crystal has the shape of a trapezoidal pyramid. The crystals range from 16 to 17.5 radiation lengths in thickness. The front faces have a typical diameter of $\sim$5 cm in each coordinate.

Each crystal is wrapped with a diffuse reflective material (TYVEK) and housed in a thin egg crate-like carbon fiber composite mechanical structure. There are 280 such modules in the barrel (7 types, 40 of each type) and 20 identical end-cap modules. The crystals are read out with two independent PIN photodiodes (2 cm^2 area) glued to their rear faces. Dual-range preamplifiers are arranged directly behind the photodiodes in a shielded housing, that also provides a thermal path for heat removal. Shielded ribbon cables carry analog signals to the end flanges of the barrel and the back plate of the end-cap, where additional amplification and digitizing electronics are mounted providing a total of four overlapping linear ranges.

The energy resolution is measured directly with a radioactive source at low energies and with electrons from Bhabha scattering at high energies, yielding resolutions of $\sigma(\mathrm{E})/\mathrm{E} = 5.0 \pm 0.8\%$ at 6.13 MeV and $\sigma(\mathrm{E})/\mathrm{E} = 1.9 \pm 0.07\%$ at 7.5 GeV, respectively. The energy resolution can also be inferred from the observed mass resolutions for the π^0 and η, which are measured to be around 7 MeV and 16 MeV, respectively.

The π^0 and η data are also used to measure the angular resolution of the calorimeter. It is found to vary between about 12 mrad at low energies and 3 mrad at high energies.

The calorimeter is also used for the separation of hadrons from electrons, and in conjunction with the IFR for muon- and K_L- identification.

The Instrumented Flux Return (IFR)

The primary aim of the Instrumented Flux Return (IFR) is to identify muons with high efficiency and purity, and to detect neutral hadrons, mainly K_L^0 in a wide momentum and angle range. It plays a major role in tagging the flavor of neutral B mesons via semileptonic decays, in the reconstruction of vector mesons, mainly J/ψ and in studying rare decays of B and D mesons involving leptons. The K_L^0 reconstruction is of special importance for the study of exclusive B decays into CP eigenstates.

Single gap resistive plate chambers (RPCs) with two-coordinate readout are embedded in the steel flux return system of the magnet, which acts as a muon filter and hadron absorber. The barrel is segmented into 19 RPC layers and the end-cap into 18 layers with increasing thickness from 2 cm for the inner to 10 cm for the outer plates. Two extra layers of RPCs

are installed between the EMC and the solenoid cryostat to detect the particles escaping from the EMC.

K_L^0 and other neutral hadrons interact in the steel of the IFR and can be identified as clusters which are not associated with charged tracks. Monte Carlo simulation shows that approximately 65% of K_L^0 with momenta above 1 GeV/c produce a cluster in the cylindrical RPC's, or a cluster with hits in two or more planar RPC layers. Un-associated clusters which have an angular separation of less than 0.3 rad, are combined into a composite cluster. The direction of the neutral hadron is defined by the event vertex and the center of the neutral cluster. No information on the energy of the cluster can be obtained. Since a significant part of hadrons interact before reaching the IFR, the information from the EMC and the cylindrical RPC is combined in order to perform the matching of the clusters.

5.2 Parametric Optimization in the *BABAR* experiment

The application of the EVA library to an optimization problem involving the maximization of the significance (see section 2.1.1.1 for the definition) of particle signals in set of histograms will be discussed below. Measured data from the *BABAR* experiment, taken from an analysis done at Ruhr-Universität Bochum by Sergey Ganjour (see [GANJ02]) served as the basis for parametric optimization studies.

It was the purpose of this analysis to perform precise measurements of D_s^+ and D_s^{*+} meson production from B mesons and $q\bar{q}$ continuum events near the $\Upsilon(4S)$ resonance.

Due to its direct impact on the measured branching fractions the efficient reconstruction of D_s mesons was of special importance. The optimization of this reconstruction will be presented below, after a short general discussion of the selection of D_s decays in *BABAR* .

5.2.1 Selection of D_s^+ decays

In order to understand the parameters involved in the optimization process it is necessary to first take a closer look at the techniques used to select and reconstruct D_s mesons. Additional details can be found in [GANJ02].

Preselection of events

It is required that charged tracks originate from within ±10 cm of the interaction point along the beam direction and ±1.5 cm in the transverse plane. Each track is then required to have at least 12 hits in the drift chamber.

These are also the main requirements of the `GoodTracksLoose` list of tracks defined as part of the *BABAR* analysis framework.

Identification of particles

The *BABAR* detector features the ability to distinguish between Kaons and Pions directly using the DIRC. In addition, the energy loss dE/dx in the drift chamber and the vertex detector helps to identify at least one of the Kaons produced in the D_s^+ decay. As part of the *BABAR* analysis software, two track selectors, `KSimpleLoose` and `KSimpleTight`, are available, amalgamating the likelihood coming from each sub-detector for a track to be a Kaon to one single variable. These criteria are used henceforth in the selection of D_s^+ mesons.

In the *BABAR* experiment, a photon candidate is defined as a localized energy maximum in the calorimeter. It must be isolated from any other photon candidate or track and must have a lateral energy profile consistent with a photon shower.

Selecting multi-hadron events, Skims

Only D_s mesons coming from B mesons and the $q\bar{q}$ continuum are of interest in the context of the analysis, making it necessary to filter out data coming from beam-gas events, Bhabba events or τ pairs. The requirement of at least three `GoodTrackLoose` tracks results in the rejection of most of these events.

Due to the vast amount of data produced by the *BABAR* detector it is furthermore necessary to create a data sample that is already enriched with the desired event type. This procedure is called "skimming". The `InclDs` skim, used in [GANJ02], uses loose cuts to select predominantly decays of D_s^+ particles into $\Phi\pi^+$, $\bar{K}^{*0}K^+$, $\bar{K}^0K^+$ and $\Phi\rho^+$. Only the subsequent decay modes $\Phi \to K^+K^-$, $\bar{K}^{*0} \to K^-\pi^+$, $K_s^0 \to \pi^-\pi^+$ and $\rho^+ \to \pi^+\pi^0$ where used to reconstruct D_s mesons in the following.

Three criteria based on *particle identification*, *track quality* and *momentum* are used to produce the skims. Kaon candidates are required not to have been identified as "very tight" pions (using the `KMicroNotAPion` selector) in order to be accepted as the basis for the reconstruction of a D_s^+ candidate. One of the three tracks coming from the D_s^+ must fulfill the requirements of the `GoodTracksLoose` list. The invariant mass of the D_s^+ candidate is required to be within $\pm 160 MeV/c^2$ of the nominal D_s^+ mass, and the mass of the Φ, $\bar{K}^{*0}$, K_s^0 and ρ^+ candidates must be within $\pm 5\,\sigma$ of their observed width. Finally the momentum of the D_s^+ candidates in the center of mass system must be higher than $1.3\,GeV/c$.

5.2.2 Reconstruction of D_s^+ mesons

The reconstuction of D_s^+ mesons in three of the four decay channels mentioned in section 5.2.1 will be discussed below. Due to reasons discussed in section 5.2.3, the decay channel $D_s^+ \to \Phi\rho^+$ will be left out of the discussion.

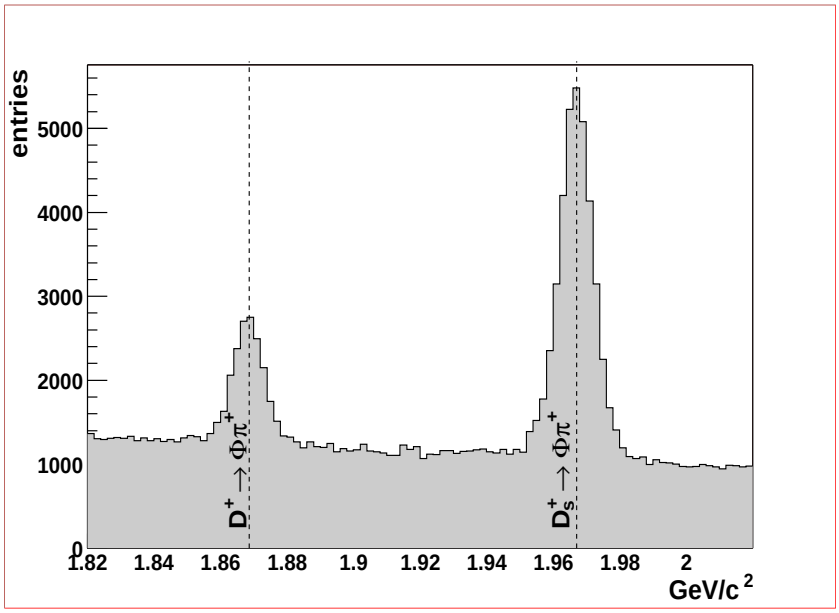

Figure 5.3: *Invariant mass distribution for the reconstruction of the $D_s^+ \to \Phi\pi^+$ decay (right peak). The plot was created using measured data taken at the BABAR experiment and uses the cuts specified in [GANJ02]. The left peak stems from the Cabibbo-suppressed decay mode $D^+ \to \phi\pi^+$.*

$D_s^+ \to \Phi\pi^+$

A D_s^+ candidate in this decay channel is reconstructed from three charged tracks coming from a common vertex. Two of these tracks are required to be identified as kaons, one using the tight list, the other using the loose list. They are required to be oppositely charged. In this particular decay the angular distribution of the kaons has a $\cos^2\theta_H$ dependence, as the ϕ meson is polarized longitudinally. θ_H is defined as the angle between the K^+ in the ϕ rest frame and the ϕ direction in the D_s^+ rest frame. As this distribution is almost flat for the background, it can be used to reject a significant amount of unwanted events. The original analysis required the invariant mass of the K^+K^- system to be within 8 MeV of the nominal mass of the ϕ. A value $|\cos\theta_H|$ >0.3 was required in addition to a fit probability $\mathcal{P}(\chi^2)_{D_s^+}$ >0.01 and a minimal momentum of the D_s^+ mesons of 1.3 GeV/c. These criteria also formed the parameters that were used in the optimization described in section 5.2.3.

Figure 5.3 shows the invariant mass distribution for the $D_s^+ \to \Phi\pi^+$ decay when using the original cuts. The lower mass peak represents the Cabibbo-suppressed decay mode $D^+ \to \phi\pi^+$. The squared significance $\mathcal{S}^2$ of the D_s peak in this plot is 3154.27.

$D_s^+ \to \bar{K}^{*0}K^+$

In the original analysis the invariant mass of the $K^-\pi^+$ system was required to be within 65 MeV of the nominal value of the $\bar{K}^{*0}$ mass. Again a cut on the helicity angle was applied, with the cut value being identical to the one used in the case of $D_s^+ \to \Phi\pi^+$. Like in this case the fit probability $\mathcal{P}(\chi^2)_{D_s^+}$ had to be higher than 0.01 and the momentum of the D_s^+ mesons had to be above 1.3 GeV/c.

$D_s^+ \to \bar{K}^0 K^+$

[GANJ02] required that the invariant mass of the $K_s \to \pi^+\pi^-$ system was within 15 MeV of the nominal mass of the K_s, and the "bachelor" kaon had to be identified using the tight criteria. In order to improve the purity of the reconstructed K_s the angle between the K_s direction and the direction defined by its decay vertex and the primary vertex of the event was required to fulfill the criterium $\cos\alpha > 0.98$. This rejects the combinatorial background while no signal is lost. Like in the previous cases $\mathcal{P}(\chi^2)_{D_s^+}$ had to be larger than 0.01 and the momentum of the D_s^+ candidate had to be above 1.3 GeV/c.

5.2.3 Optimizing the Squared Significance $\mathcal{S}^2$ using the EVA library

The optimization of $\mathcal{S}^2$ in the decay channels $\Phi\pi^+$, $\bar{K}^{*0}K^+$ and $\bar{K}^0K^+$ of the D_s^+ using the EVA library will be discussed below. The optimization of the decay channel $D_s^+ \to \Phi\rho^+$ will be left out of the discussion, due to the difficulties involved in determining the background from the sidebands in this channel. In practice the background could be obtained either from Monte Carlo data or from a fit[2].

The cuts to be optimized are those presented in section 5.2.2, the algorithm was allowed to vary four parameters for each channel. This optimization uses the techniques discussed in section 2.1.

5.2.3.1 $D_s^+ \to \Phi\pi^+$

The optimization was done independently for four data sets, each representing a quarter of the overall data, equivalent to roughly 2150000 events each. The third quarter yielded the best results. Optimizations done with the other datasets for the same decay channel also showed significant improvements of $\mathcal{S}^2$, but had worse results than for the third dataset. This could indicate that more than one local optimum exists in the quality surface, but might also be a hint that the background levels were different for part of the data, possibly due to different experimental conditions.

Figure 5.4a shows the result of the optimization, including all intermediate steps, in the $\mathcal{S}^2$ – R_{sn} plane. The large stars represent the best results of a generation, the small triangles are all other solutions that were tested. Furthermore, lines of equal N allow to conclude, whether the algorithm favored solutions with a lower background over those with a larger peak or vice versa. The initial focus of this optimization seems to have been on a lower background. Further progress could then apparently only be made by varying the cuts in a way that the total number of entries in the peak region (including the background) was

[2]Please note that the latter possibility implies additional challenges, as during the optimization stage a user would not be able to directly control the fit procedure.

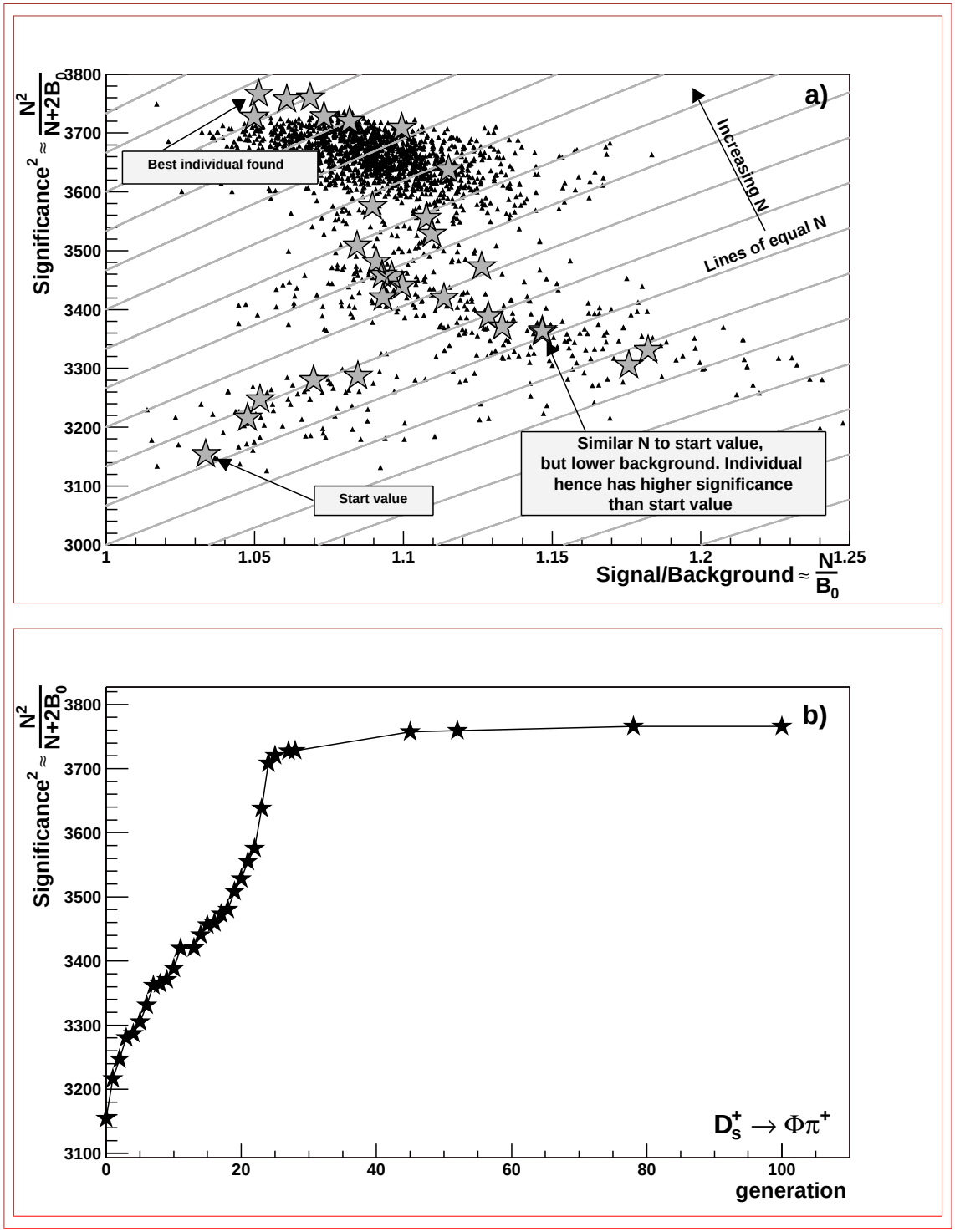

Figure 5.4: *a) The plot shows all tested solutions during an optimization of the reconstruction of the decay* $D_s^+ \to \Phi\pi^+$ *in the* $\mathcal{S}^2$ *vs.* R_{sn} *plane.* $\mathcal{S}^2$ *was used as a figure of merit in this optimization. The best solution found in each generation is shown as a star. Stars with a higher* $\mathcal{S}^2$ *were found later in the optimization than those with a lower* $\mathcal{S}^2$*. Solutions with a higher* R_{sn} *as well as a higher* $\mathcal{S}^2$ *were found during the optimization. The final result exhibited a 19.4% higher* $\mathcal{S}^2$ *(of* $3766,03$*) compared to the start values (with* $\mathcal{S}^2 = 3154,27$*). Given the large number of tested solutions it also becomes clear that parallelisation of the optimization procedure will return results quicker. b)* $\mathcal{S}^2$ *is shown as a function of the generation of the Evolutionary Strategy. The most progress is achieved during the first 25 generations.*

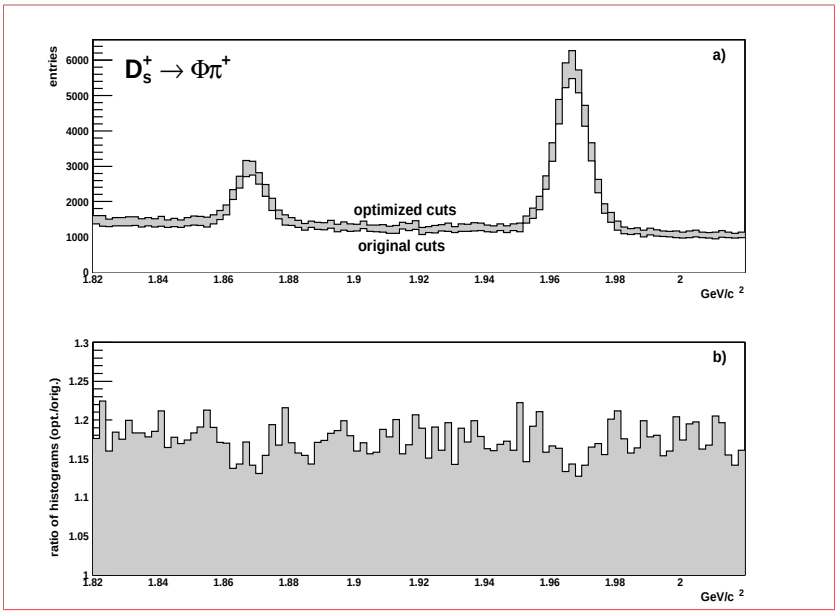

Figure 5.5: *a) The plot shows the reconstructed $D_s^+ \to \Phi\pi^+$ (right peak) before and after the optimization with $\mathcal{S}^2$ used as a figure of merit. It is visible that the procedure has led to a higher number of entries in the histogram. b) The ratio of both histograms is shown. The "dip" in the D_s^+ region shows that a higher $\mathcal{S}^2$ could be achieved despite a lower signal-to-noise ratio.*

increased.

An interesting feature of the plot is the large amount of tested (and rejected) solutions near the best solution after 100 generations. This indicates that this solution is actually close to the global optimum or at least represents a very significant local optimum, as in its proximity no better solutions could be found, despite many possible solutions being tried.

This is also reflected in figure 5.4b, which shows $\mathcal{S}^2$ as a function of the generation of the Evolutionary Strategy. While initially a better solution could be found in virtually any of the first 30 generations, only little improvement could be achieved thereafter. This is typical for the optimization with Evolutionary Strategies.

Figure 5.5 shows the histograms for this decay channel before and after the optimization (a) as well as their ratio (b). Only the first, second and fourth dataset were used to create this plot, so the effect of the optimized parameters on data sets can be shown, that were not used by the algorithm during the optimization.

Several potentially useful solutions were found during the optimization. The best solution

Cuts	original	optimized
$\mathcal{P}(\chi^2)_{D_s^+}$	> 0.01	$> 2.04 * 10^{-6}$
Φ mass window	$\pm 8 MeV/c^2$	$\pm 10.5 MeV/c^2$
p^*	$> 1.3 GeV/c$	$> 1.191 GeV/c$
$\lvert\cos\theta_H\rvert$	> 0.3	>0.44

Table 5.1: *Selection criteria for $D_s^+ \to \Phi\pi^+$ before and after optimization (see text)*

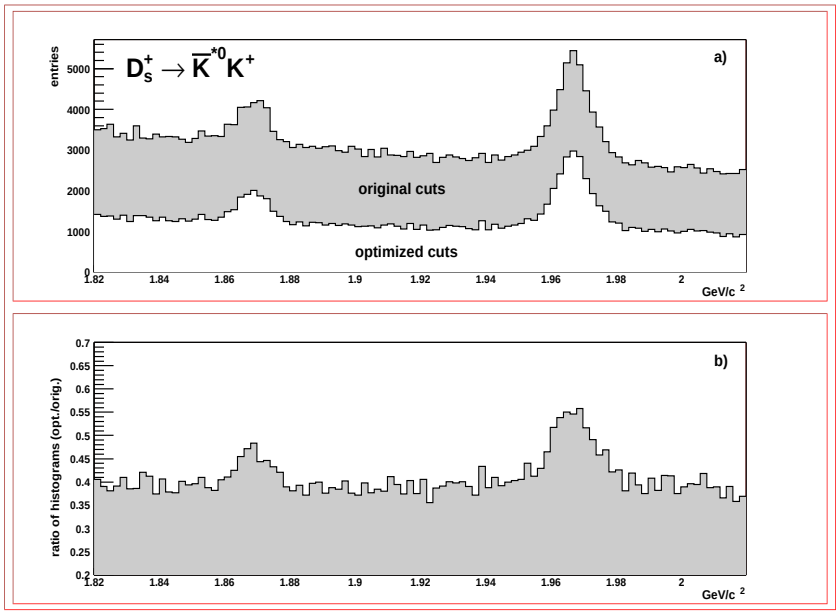

Figure 5.6: *a) The plot shows the reconstructed $D_s^+ \to \bar{K}^{*0}K^+$ decay (right peak) before and after the optimization. A higher $\mathcal{S}^2$ was achieved by a reduction of the overall statistic in combination with a higher signal-to-noise ratio, as can be seen from the ratio of the original and optimized histogram (b). $\mathcal{S}^2$ is increased by 45% when using the optimized cuts.*

(see again figure 5.5) had a slightly lower signal-to-background ratio to the start value, but a 19.4% higher $\mathcal{S}^2$. It is obvious that the higher $\mathcal{S}^2$ is achieved by scaling both the background and the signal by an almost equal factor. As $\mathcal{S}^2 = \frac{N^2}{N+2B_0}$, this will still lead to an improved figure of merit. Another solution found during the optimization (see the large label in figure 5.4a) had a higher signal-to-noise ratio than the start value while having an almost equal N. This can only be done by scaling down the background. A lower $\mathcal{S}^2$ than for the best solution was achieved, though.

Looking at table 5.1 it is possible to conclude, what changes were made by the EVA library to achieve a higher figure of merit. Both the cuts on $\mathcal{P}(\chi^2)_{D_s^+}$ and the Φ mass window were loosened, in the case of the $\mathcal{P}(\chi^2)_{D_s^+}$ even to an extent that one could safely neglect this cut. The cut on p^*, while also appearing to be looser than the original cut, could be actually again left unchanged – the minimal p^* available in the data is $1.3GeV/c$ due to the preselection. The lower boundary on this variable has been intentionally left open. Enforcing a lower limit of $1.3GeV/c$ on this variable would have meant that mutations of this variable would have always pointed away from the boundary, making it less likely for

Cuts	original	optimized
$\mathcal{P}(\chi^2)_{D_s^+}$	> 0.01	> 0.0109
$\bar{K}^{*0}$ mass window	$\pm 65 MeV/c^2$	$\pm 34.8 MeV/c^2$
p^*	$> 1.3GeV/c$	$> 1.437GeV/c$
$\lvert\cos\theta_H\rvert$	> 0.3	>0.398

Table 5.2: *Selection criteria for $D_s^+ \to \bar{K}^{*0}K^+$ before and after optimization (see text)*

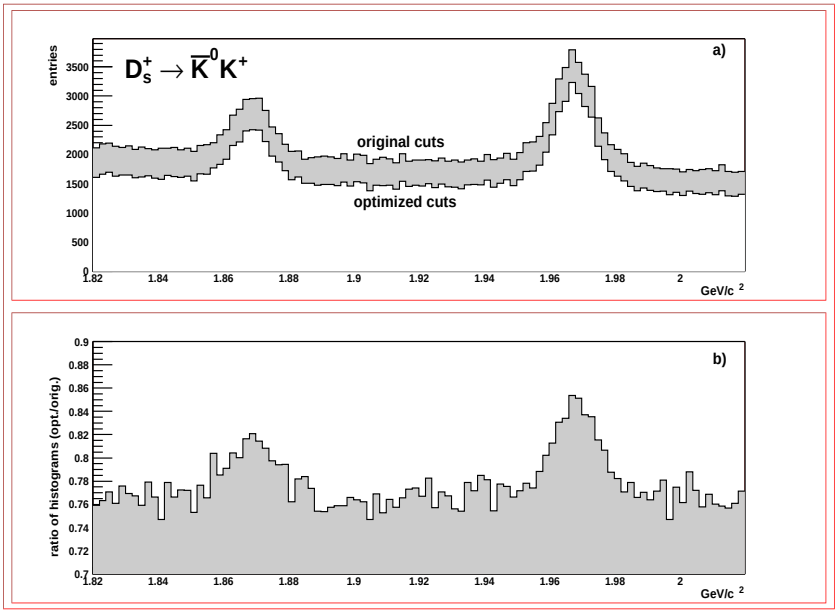

Figure 5.7: *a) The plot shows the reconstructed $D_s^+ \to \bar{K}^0 K^+$ (right peak) before and after optimization. Like in the case of figure 5.6 the increase of $\mathcal{S}^2$ was achieved by a reduction of the overall statistic in combination with a higher signal-to-noise ratio, as can be seen from (b). $\mathcal{S}^2$ is increased by 16.3% when using the optimized cuts.*

the cut to remain unchanged, thus leading to worse results. The only cut that is tighter after the optimization is on $|\cos\theta_H|$. The overall effect of these cuts is then consequently an increase of entries in the histogram. And while the signal-to-noise ratio hasn't improved, $\mathcal{S}^2$ has done so by a considerable amount. This also shows that it is not always possible to manually select the better peak, which again underlines the importance of automated parametric optimization studies.

In contrast to R_{sn}, $\mathcal{S}^2$ presents itself as a viable figure of merit for these studies.

5.2.3.2 $D_s^+ \to \bar{K}^{*0} K^+$ and $D_s^+ \to \bar{K}^0 K^+$

Figures 5.6 and 5.7 show the results of the optimization of the channels $D_s^+ \to \bar{K}^{*0} K^+$ and $D_s^+ \to \bar{K}^0 K^+$. In contrast to $D_s^+ \to \Phi\pi^+$, the increase in significance was achieved mainly by reducing the background. Especially the channel $D_s^+ \to \bar{K}^{*0} K^+$ features an impressive 45% increase in $\mathcal{S}^2$. As shown in table 5.2, this is achieved by much tighter cuts. The same applies to $D_s^+ \to \bar{K}^0 K^+$ (see table 5.3) with an increase in $\mathcal{S}^2$ of 16.3%, although the results are not as good as for $D_s^+ \to \bar{K}^{*0} K^+$.

Cuts	original	optimized
$\mathcal{P}(\chi^2)_{D_s^+}$	> 0.01	> 0.01005
$\bar{K}^0$ mass window	$\pm 15 MeV/c^2$	$\pm 6.9 MeV/c^2$
p^*	$> 1.3 GeV/c$	$> 1.352 GeV/c$
$\cos\alpha$	> 0.98	>0.9856

Table 5.3: *Selection criteria for $D_s^+ \to \bar{K}^0 K^+$ before and after optimization (see text)*

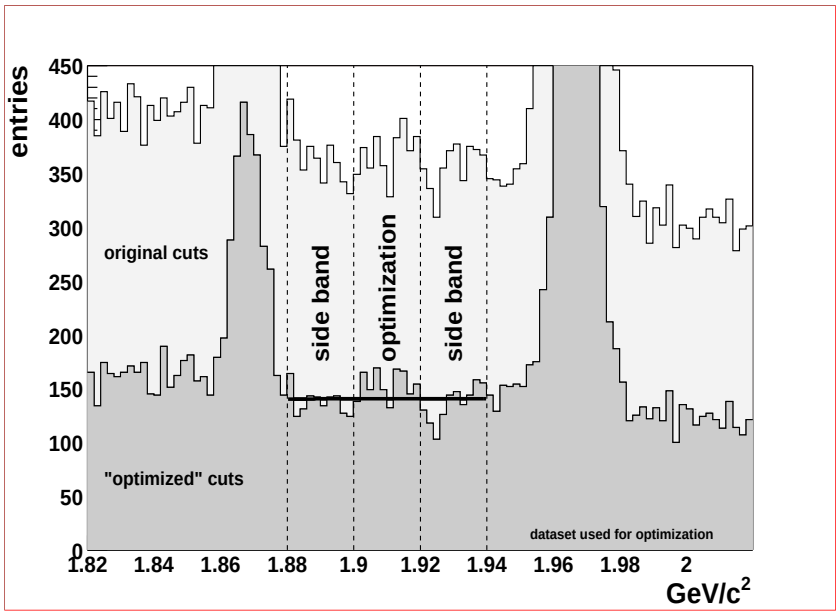

Figure 5.8: *A "fake" peak was created between the D and D_s peaks by "optimizing" the squared significance in an area where no physical peak is present. Both the histogram resulting from the original cuts and the one using the "optimized" cuts are shown. It can be seen that the fake peak is really just a random fluctuation already present in the original plot. It got more significant by a reduction of entries in the two sidebands,*

5.2.3.3 Achievements of the optimization procedure

$D_s^+ \to \Phi\pi^+$ is the most important decay channel of the D_s^+ used in [GANJ02]. The EVA library, using an Evolutionary Strategy with 20 child individuals, yields a 19.4% improvement of the chosen figure of merit $\mathcal{S}^2$ for the D_s^+ signal compared to the original cuts. The execution time goes down from 38 hours for the sequential execution to under 2 hours. An improvement of 45% can be seen in the decay channel $D_s^+ \to \bar{K}^{*0}K^+$ of the D_s^+. These examples show that using automated parametric optimization procedures can yield significant improvements of an analysis, despite the fact that a substantial amount of work was invested into choosing the original cuts.

5.2.4 Sanity checks

5.2.4.1 Creating fake peaks

When using automatic procedures to optimize parameters, care has to be taken not to accidently enhance random fluctuations in the data set, thus creating fake or artificially large peaks.

Figure 5.8 shows that this can be a real-life problem. Here an attempt was made to create a fake peak in the flat background between the Cabibbo-suppressed D and the D_s peak of figure 5.3 by optimizing the squared significance in the area labeled "optimization". As usual, the background was taken from the side bands (labeled "side band"). A straight line was then fitted to the region in order to check, how well the fake peak is still compatible with

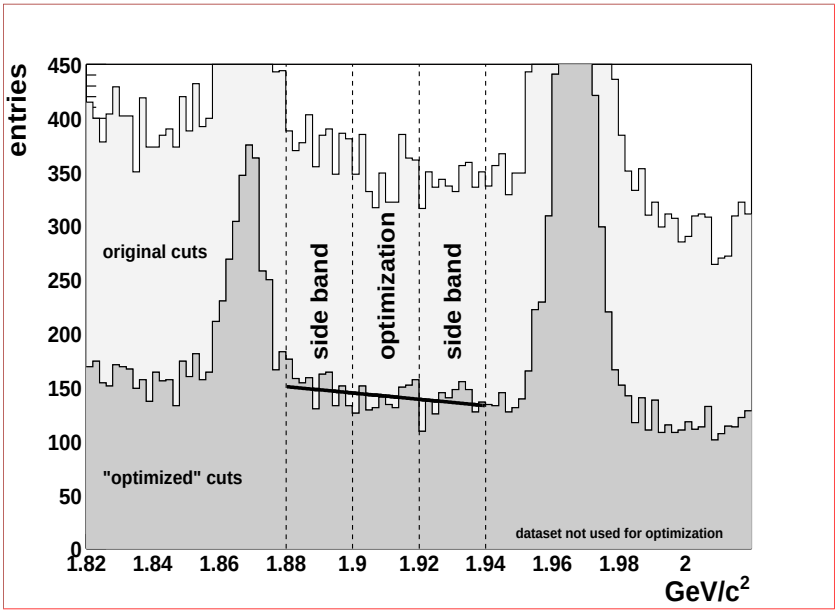

Figure 5.9: *The application of the cuts used to generate the fake peak in figure 5.8 to a different dataset shows no sign of a fake peak. This again underlines that fake peaks are the result of random fluctuations present in the data used for the optimization.*

the assumption of a flat background. The fit yielded a χ^2/NDF of 54.5/29, corresponding to a fit probability of just 0.0028. $\mathcal{S}^2$ went up from 1.003 for the original cuts to 9.726 for the "optimized" cuts.

An important feature of this plot is that, despite the low significance for the original cuts, the random fluctuation is already visible - albeit compensated by the side bands. The parametric optimization has only somewhat reduced the number of entries especially in the left side band, thus increasing $\mathcal{S}^2$. As a comparison, $\mathcal{S}^2$ for the D_s peak is 3154.27±102.8 for the original cuts. The fake peak can thus *not* be considered to be a large effect.

By the application of the same set of cuts used to create the fake peak to a different dataset not used for the optimization it can be shown that it just represents a random fluctuation. This is illustrated in figure 5.9. No fake peak is visible in this plot.

Figure 5.8 represents the worst result that could be achieved in a series of tests with datasets of varying sizes[3]. Figure 5.10 shows the squared significance of fake peaks that could be achieved for datasets varying between 50 and 400 input files.

5.2.4.2 Detecting artificial contributions to optimized peaks

While, in the above example, it is easily possible to establish the existence of a fake peak, the same does not apply to cuts enhancing a signal[4] generated from "real" (i.e. measured) data. There is no practical way how one could detect a fake contribution to a peak for the data set used for optimization, unless one is using simulated data and has access to the

[3]Measured in amount of input files

[4]such as the D_s peak visible in figure 5.3)

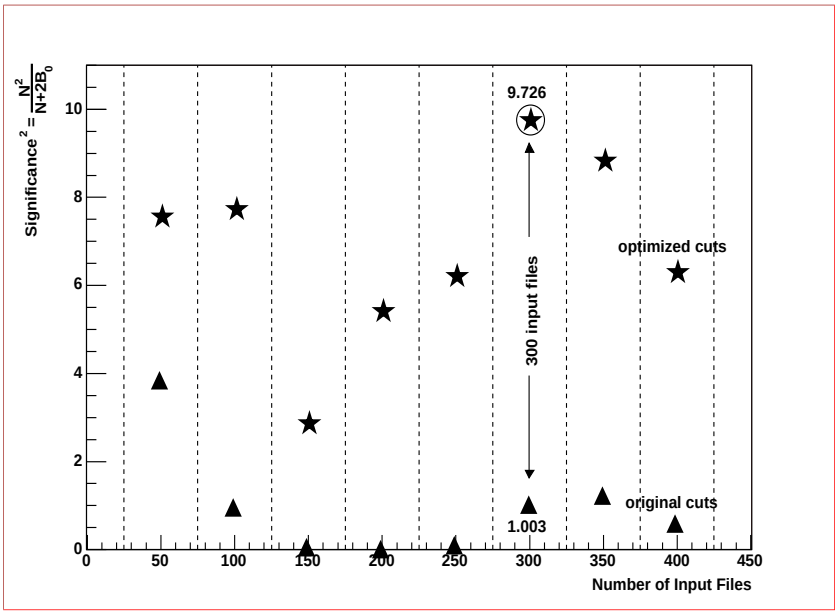

Figure 5.10: $\mathcal{S}^2$ *is shown for a fake peak (compare fig. 5.8) and the original cuts as a function of the amount of input data (measured in "number of input files"). The creation of a fake peak in between the* D *and the* D_s *peak was – with varying degrees of success – possible in all cases. Nevertheless* $\mathcal{S}^2$ *is in each case small compared to the* D_s *peak (*$\mathcal{S}^2(D_s) = 3154$ *for the unoptimized peak).*

"Monte Carlo truth"[5].

In theory, provided the amount of available data is large enough, one should see a deviation from the natural peak form (such as a gaussian), as random fluctuations are unlikely to be of the same shape as the physical contribution. In practice, however, one rarely has enough data.

It would be a good indication for a fake peak, if the $\mathcal{S}^2$ for the dataset used for optimization is significantly higher than for other datasets. As an example, the cuts used to produce figure 5.5 were obtained using just one quarter of a larger dataset. Figure 5.11 shows the squared significance for all four parts of the dataset in the case of the decay $D_s^+ \to \phi\pi^+$. Care has been taken to use an equal amount of input events, as $\mathcal{S}^2$ depends on the amount of data being used.

It can be seen that the first and fourth dataset agree very well with the third dataset, which was used for the optimization. The application of the cuts to the second dataset, however, yields slightly different results. As two of the three datasets agree very well with the one used for the optimization and the one with a larger deviation still agrees with it within its error bounds, one would conclude that the cuts found are usable in an analysis.

Indeed the procedure used here for testing the viability of the optimized cuts can be used to become independent of random fluctuations in the data. It is always possible to split the dataset in two parts and to search for two different sets of optimized cuts independently for both of them. The cuts of the first half are then applied to the second half and vice versa. This procedure will also give a good indication whether or not one has found a local optimum, as both optimizations should result in similar cuts.

[5]or one is using a "background MC".

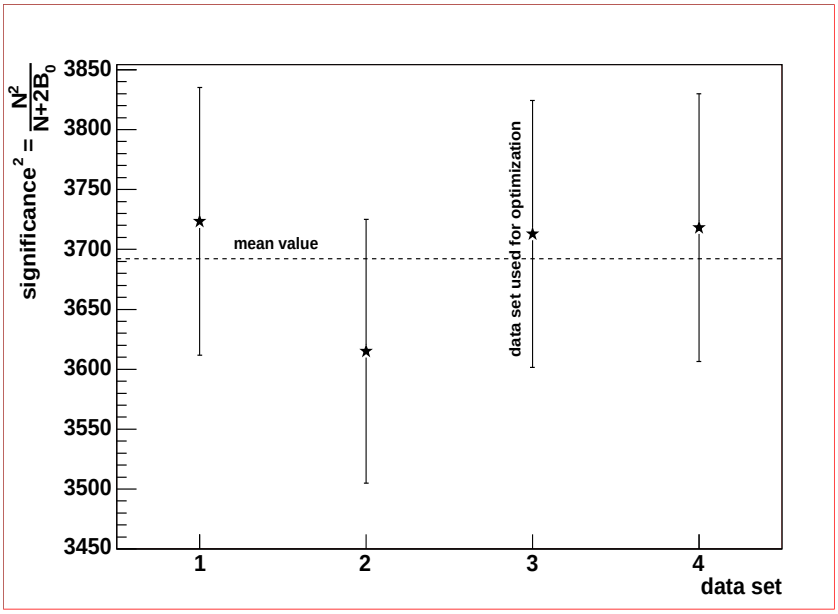

Figure 5.11: *In order to test the stability of $\mathcal{S}^2$, a set of cuts resulting from the optimization of the reconstruction of the decay $D_s^+ \to \phi\pi^+$ was applied to four different datasets of equal size. The third dataset was used for the optimization. The values agree well within their error-bounds.*

5.2.4.3 Discussion

While the danger of accidently using a set of cuts that enhances a random fluctuation is indeed real, it should again be noted that, as shown in figure 5.8, the same also applies to manually selected cuts. Furthermore, the $\mathcal{S}^2$ achieved for the fake peak was small compared to the one found for the D_s peak, so the observed fluctuation wasn't large.

One lesson to be learned from this is that optimized cuts should not be used in conjunction with the dataset that was used for the optimization.

When using the squared significance as a figure of merit, one should also be aware of the fact that – as shown in the comparison of three decay channels (see figures 5.5, 5.6 and 5.7) – it is not always a larger peak that leads to a higher significance. A lower background can have the same effect, and so the creation of fake peaks is not a danger present in all optimizations involving significance.

Chapter 6

CB/ELSA Data: Dalitz Plot Analysis

Following the (intentionally short) introduction of the CB/ELSA detector system, this chapter discusses the application of the EVA library's *Evolutionary Strategy* implementation to a Dalitz plot analysis done in cooperation with Dr. Bertram Kopf of the CB/ELSA collaboration [KOPF03]. Dalitz plot analyses, including their relation to optimization tools, are discussed in section 2.2. This is again an example of parametric optimization studies involving heavy computation.

6.1 The CB/ELSA experiment

The CB/ELSA experiment is the successor of the Crystal Barrel experiment which, between 1989 and 1996, was used to study the annihilation of protons p and anti-protons $\bar{p}$ at the **L**ow **E**nergy **A**ntiproton **R**ing (LEAR) at CERN/Geneva. CB/ELSA, now located at the electron accelerator ELSA[1] in Bonn/Germany, re-uses the 1380 module CsJ(Tl) calorimeter capable of exceptional energy and momentum resolution in the reconstruction of neutral particles in a (close to) 4π solid angle. CB/ELSA is being used to study photo-production. The detector system is discussed in the following (see also figures 6.1 and 6.2). Additional details about the CB/ELSA experiment and the associated accelerator complex can be found in [KOPF02].

Electrons with fixed energies between $0.5GeV$ and $3.2GeV$ coming from ELSA hit a radiation target, responsible for the creation of high-energetic photons through the bremsstrahlung process. The electrons then pass a magnetic field and are deflected according to their particular momentum.

Electrons that have emitted a photon now hit a tagging system, capable of measuring their energy and emitting a start (time) signal. The energy measurement is then used to in turn calculate the energy of the photons. Electrons that have not emitted a photon have a

[1] **El**ektronen **S**tretcher **A**nlage ("Electron Stretcher Device")

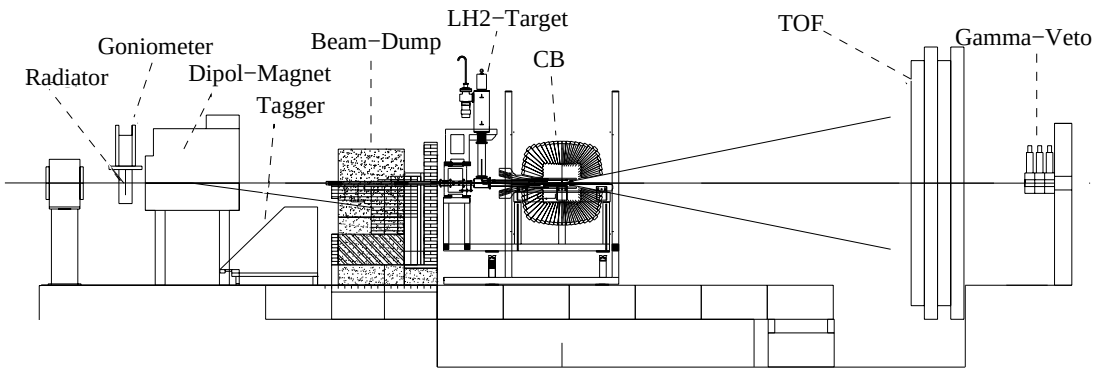

Figure 6.1: *Overview of the CB/ELSA experiment*

higher momentum and are thus deflected less in the magnetic field. They subsequently hit a beam dump. The high-energetic photons then hit a liquid hydrogen target ("LH_2–target"), where photo-production is taking place.

Particles coming from the interaction point then pass the CB/ELSA detector, where they are detected and measured.

Two sub-detectors of CB/ELSA will be discussed in the following, the inner detector and the electromagnetic calorimeter. These central components of the experiment are also shown together in figure 6.2

6.1.1 The inner detector

It is the responsibility of the inner detector to distinguish between charged and neutral final-state particles. Three layers of altogether 513 scintillating fibers, each having a length of $40\,cm$, are arranged in a cylinder around the LH_2–target. The outer layer's fibers are oriented parallel to the beam pipe, the fibers of the middle and inner layer have a $+25^o$ and -25^o angle with respect to the beam pipe. This topology allows to determine the coordinates where a charged particle has hit the detector. Together with the target coordinates it is then possible to determine, where a charged particle has hit the calorimeter (see section 6.1.2). As charged final-state particles can have enough energy to pass through the entire calorimeter, only part of their energy is deposited in it. It is thus not possible to measure the energy of charged final-state particles with CB/ELSA. Furthermore, in contrast to the original Crystal Barrel experiment at CERN, CB/ELSA uses neither a drift chamber nor a magnet, and thus cannot be used to measure the momentum of a particle.

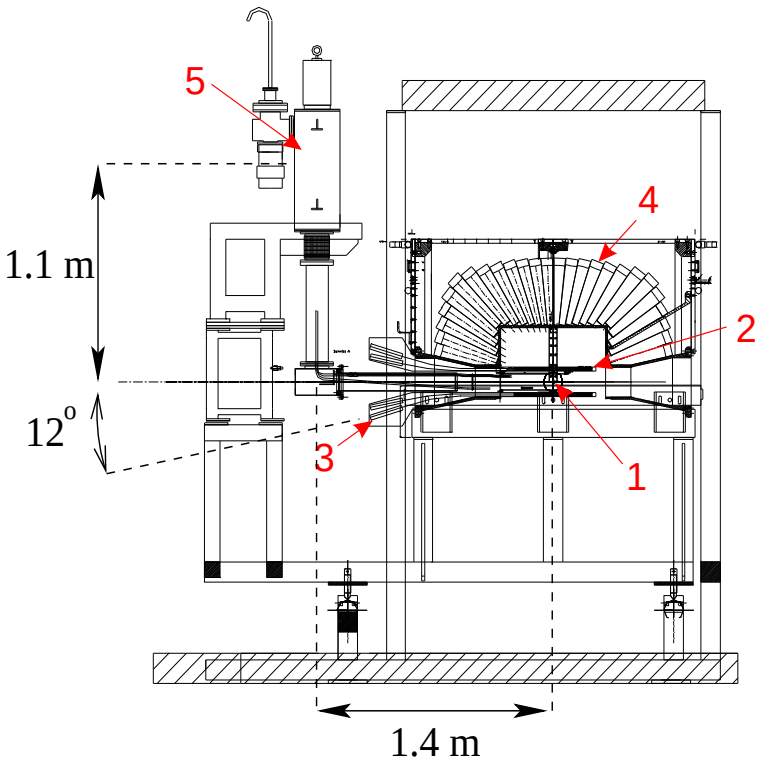

Figure 6.2: *Cross section of the CB/ELSA detector system. (1) liquid hydrogen target; (2) inner detector; (3) photo-multipliers needed for the read-out of the inner detector; (4) CsI(Tl) calorimeter with 1380 segments; (5) provision of liquid hydrogen for the target.*

6.1.2 The Crystal Barrel Calorimeter

The Crystal Barrel calorimeter provides exact energy and momentum information for photons, mostly coming from the decay of neutral mesons. Using the inner detector (see section 6.1.1) it is also able to measure charged particles, although it is usually not possible to provide full energy information.

The calorimeter is built from 1380 separate crystal modules, arranged in a barrel in 26 concentric rings around the beam pipe. In order to allow the particle beam to enter and exit the detector, the calorimeter has 12^o openings on both sides. The calorimeter nevertheless covers a solid angle of $97.8\% \cdot 4\pi$.

Every crystal covers a polar angle of $\Delta\Theta = 6^o$, making it necessary to use 13 different types of crystals. The inner crystals of type 1– 10 each cover an azimuthal angle of $\Delta\Phi = 6^o$, crystal types 11– 13 each cover $\Delta\Phi = 12^o$.

Each crystal consists of caesium–iodide, doped with thallium (CsI(Tl)). Each crystal has a length of $30\,cm$ (corresponding to about 16 radiation lengths), making it possible for photons with energies of up to $2\,GeV$ to deposit virtually their entire energy in the crystal. Thallium has been used to dope the crystals in order to shift the wavelength of the emitted light, preventing its resorption in the crystals. Each crystal is protected by a titanium cover, which is itself enclosed in a protective covering, so electric isolation between each module can be ensured.

The original Crystal Barrel experiment at CERN was located in a strong (1.5 Tesla) magnetic field. This prevented the use of photo multipliers to measure the light coming from the crystals. Photo diodes have been used instead, and CB/ELSA has inherited this set-up. The scintillation light has a maximum emission at a wavelength of $550\,nm$. The photo diodes, however, give best results for infrared light. Wavelength shifters (small plexiglass wafers) are thus used to achieve better results. Entering light is emitted through the $3\,mm$ thick edges of the wafer, to which the photo diodes are attached. This in particular allows to use very small photo diodes, which in turn results in a less noisy measurement.

A pre-amplifier is mounted onto the back of each crystal module. Its signals are passed through shapers and are then transmitted to a ADC readout system, which was newly developed for CB/ELSA.

Energy calibration is done via the decays of neutral pions into two photons. Calibration constants are determined in an iterative procedure separately for each crystal, such that the π^0 peak in the invariant mass spectrum of the two photons is at a nominal mass of 134.98 MeV/c^2. More then 10^5 π^0 decays are needed to achieve a good calibration.

6.1.3 TOF and Gamma Veto Detector

The **TOF** ("Time of Flight Spectrometer") helps to detect charged particles, that pass in forward direction through the (non-instrumented) end of the inner detector and the hole in the crystal matrix. It consists of 4 walls, each equipped with 15 scintillator bars. Each scintillator bar has a length of $3\,m$ and a width of $20\,cm$. Each wall thus covers an area of $3\,\mathrm{x}\,3\,m^2$. In order to detect where a particle has hit the TOF, the bars are mounted in turn in horizontal and vertical direction.

The Gamma Veto Detector, as seen from CB/ELSA, is mounted behind the TOF. Its purpose is to detect photons that didn't lead to a reaction in the LH_2 target. It is made of lead disks, mineral oil (being used as a cherenkov radiator, and photomultipliers.

With its fine-grained electromagnetic calorimeter, as discussed in section 6.1.2, the CB/ELSA experiment is especially well suited to the measurement of neutral decays. Nevertheless charged particles, such as a proton emanating from a reaction $p\gamma \to p\pi^0\eta$ can be accounted for, using the inner detector[2].

6.2 Analyzing the $p\pi^0\eta$ final state at CB/ELSA

In order to analyze the reaction $p\gamma \to p\pi^0\eta$ for center of mass system energies in the range $2.35\,GeV \leq E_{cms} \leq 2.4\,GeV$, a Dalitz plot analysis was done in cooperation with a member of the CB/ELSA collaboration [KOPF03], using the EVA library. The optimization

[2]The special strengths of CB/ELSA could also help in the reconstruction of the recently discovered penta quark Θ^+, a particle involving 5 quarks.

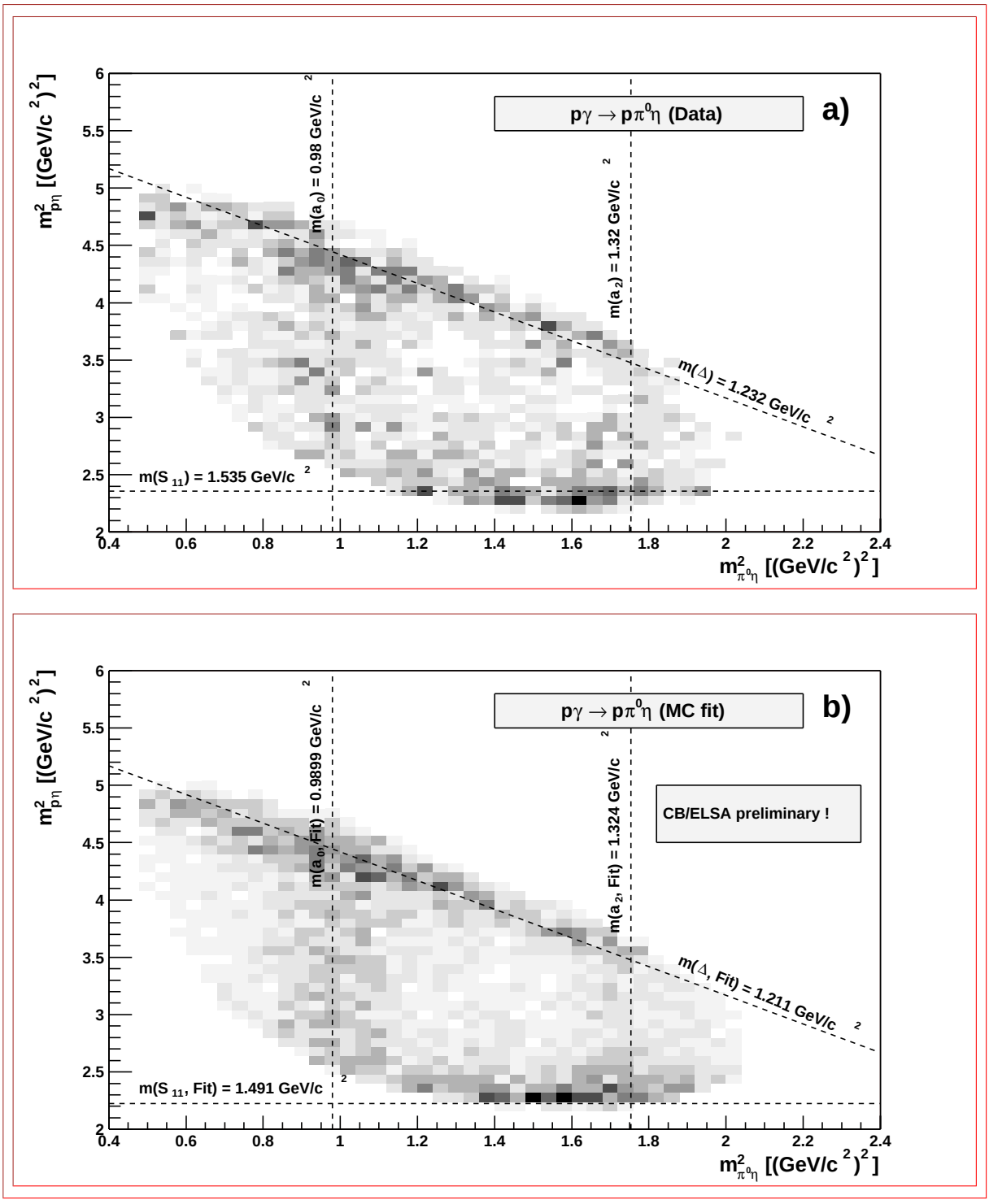

Figure 6.3: *Dalitz plots for the reaction $p\gamma \to p\pi^0\eta$ are shown in the case of measured data (a) and for the fit of phase-space distributed Monte Carlo events to this data (b). $m^2_{p\eta}$ is in both cases plotted against $m^2_{\pi^0\eta}$. Resonances are indicated by dotted lines.*

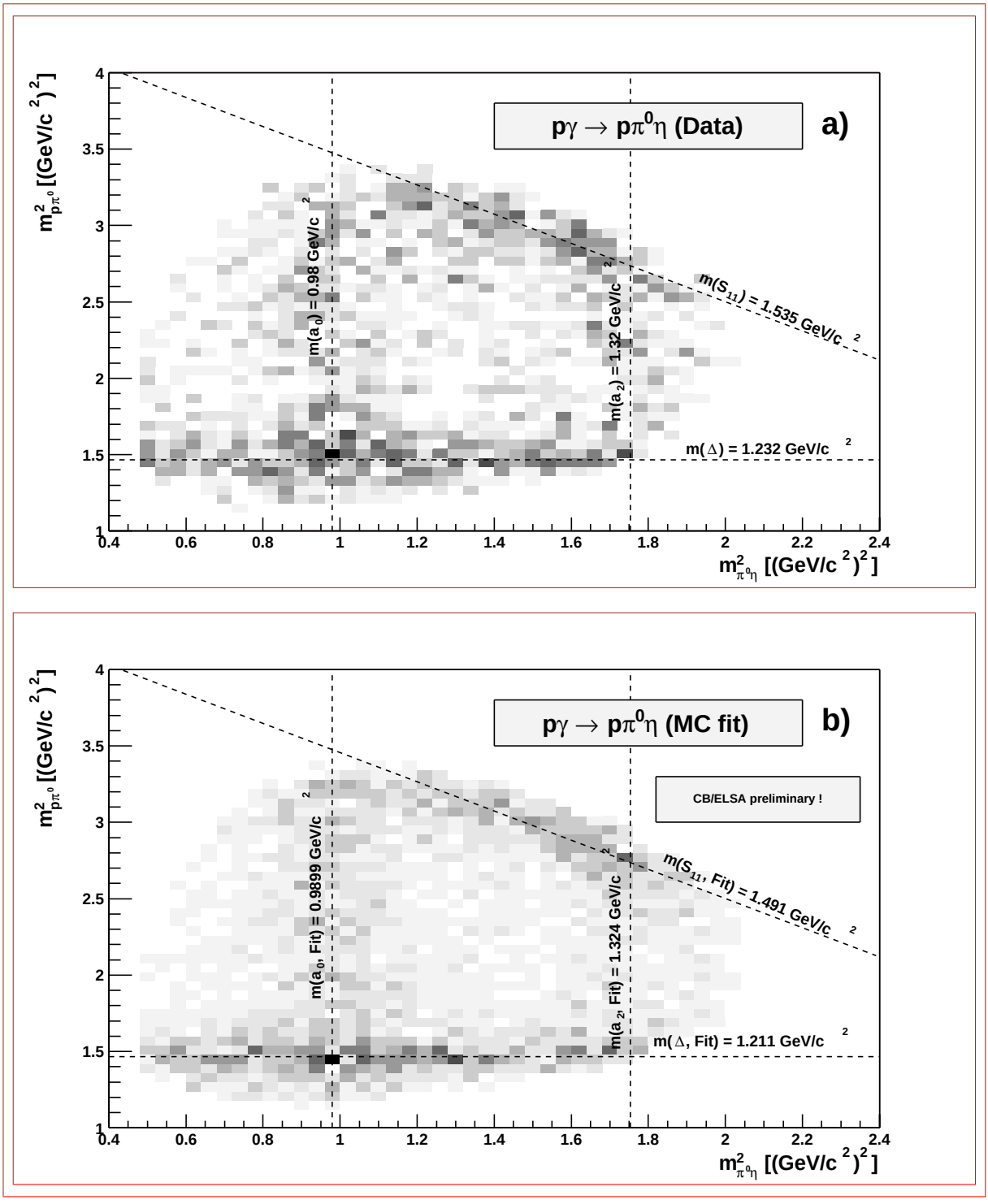

Figure 6.4: *Like in the case of figure 6.3, Dalitz plots for the reaction* $p\gamma \rightarrow p\pi^0\eta$ *are shown in the case of measured data (a) and for the fit of phase-space distributed Monte Carlo events to this data (b).* $m^2_{p\pi^0}$ *is plotted against* $m^2_{\pi^0\eta}$*. Resonances are indicated by dotted lines.*

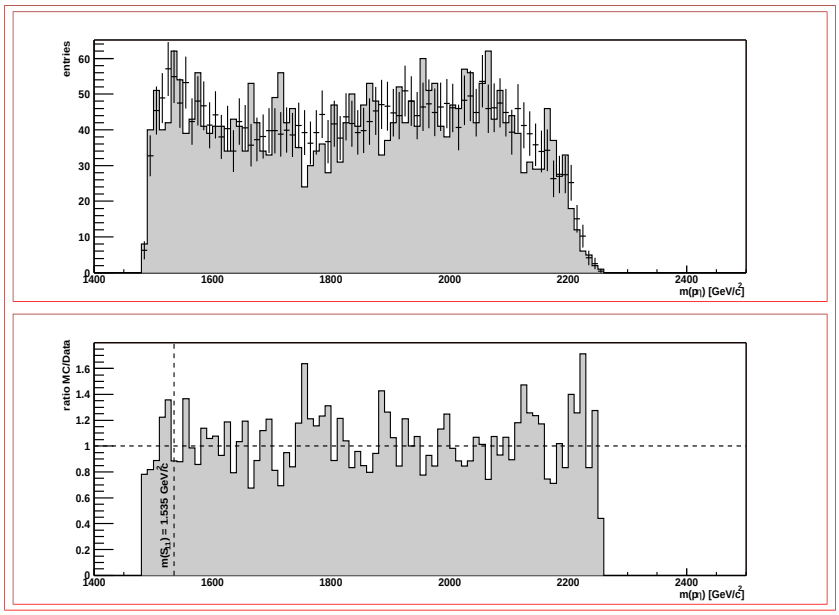

Figure 6.5: *a) The plot shows the projection of Dalitz plot 6.3a on the $p\eta$ mass scale (grey histogram). Plotted on top of it is the projection of the corresponding MC Dalitz plot (histogram with error bars). Both histograms agree well within the error boundaries. b) shows the ratio of both histograms.*

framework, including the "individual" class needed for the optimization (compare A.3), was created by the author of this thesis, using an evaluation function provided by CB/ELSA [KOPF03].

As discussed in section 2.2, doing a Dalitz plot analysis means to assign weights to phase-space distributed MC events, which are subsequently filled into a two-dimensional histogram. By adapting the paramters of these weights, a χ^2 representing the difference between this histogram and the corresponding histogram created from data is minimized.

The latter plots, created using measured data, are shown in figures 6.3a and 6.4a. Figures 6.3b and 6.4b show the corresponding Monte Carlo plots after the adaptation of the weights.

In figures 6.3a and 6.3b, $m^2_{p\eta}$ is in both cases plotted against $m^2_{\pi^0\eta}$. 6.4a and 6.4b show $m^2_{p\pi^0}$ being plotted against $m^2_{\pi^0\eta}$. Both forms of the Dalitz plot for the reaction $p\gamma \rightarrow p\pi^0\eta$ are equivalent.

As can be seen from these plots, the $p\pi^0\eta$ final state shows resonances in each two of the three final state particles. Most notably the $\Delta(1232)$, the $S_{11}(1535)$, the $a_0(980)$ and, with a comparatively weak signal, the $a_2(1320)$ are visible.

The weight function used by CB/ELSA, following the procedure described in section 2.2 and omitting interference terms, is shown in equation 6.1[3]. It takes into account the resonances described above.

In the context of this optimization, 16 parameters of equation 6.1 were allowed to be varied, most notably the relative contribution of each resonance (R_x) as well as the phase space ($R_{ph.sp.}$), the masses of each resonance (m) and their widths (Γ). "BW" represents the

[3]Please note that this represents ongoing work of the CB/ELSA collaboration and that all plots and formulas shown must be considered to be **preliminary**!

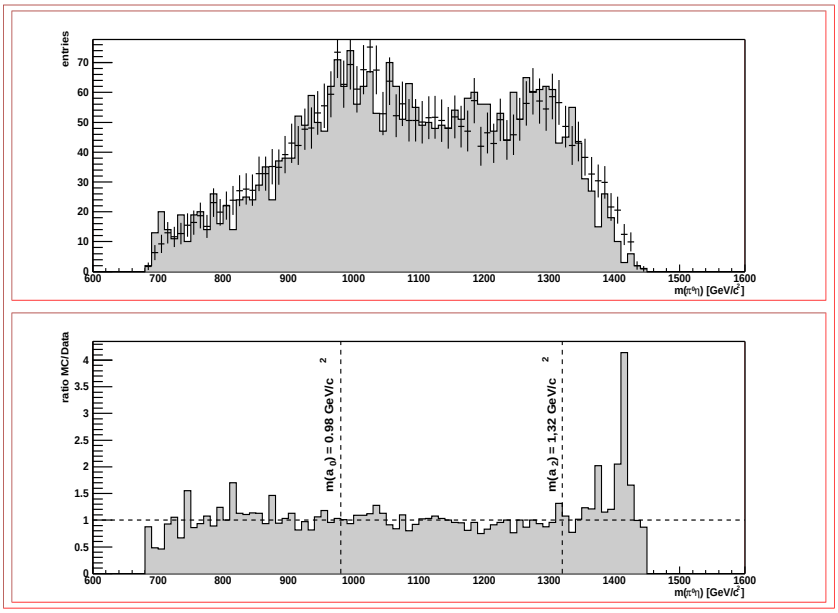

Figure 6.6: *a) The plot shows the projection of Dalitz plot 6.4a on the $\pi^0\eta$ mass scale (grey histogram). Plotted on top of it is the projection of the corresponding MC Dalitz plot (histogram with error bars). Both histograms agree well within the error boundaries, especially in the signal region of the a_0 and a_2. b) shows the ratio of both histograms.*

Breit-Wigner function, Y are spherical harmonics.

$$\begin{aligned} W &= R^2_{ph.sp.} + R^2_{\Delta_{10}} \left| BW\left(m_\Delta, \Gamma_\Delta\right) Y^0_1\left(\Theta_{p_\Delta}, \Phi_{p_\Delta}\right)\right|^2 \\ &+ R^2_{\Delta_{11}} \left| BW\left(m_\Delta, \Gamma_\Delta\right) Y^1_1\left(\Theta_{p_\Delta}, \Phi_{p_\Delta}\right)\right|^2 + R^2_{S_{11}} \left| BW\left(m_{S_{11}}, \Gamma_{S_{11}}\right)\right|^2 \\ &+ R^2_{a_0} \left| BW\left(m_{a_0}, \Gamma_{a_0}\right)\right|^2 + R^2_{a^{20}_2} \left| BW\left(m_{a_2}, \Gamma_{a_2}\right) Y^0_2\left(\Theta_{\eta_{a2}}, \Phi_{\eta_{a2}}\right)\right|^2 \\ &+ R^2_{a^{21}_2} \left| BW\left(m_{a_2}, \Gamma_{a_2}\right) Y^1_2\left(\Theta_{\eta_{a2}}, \Phi_{\eta_{a2}}\right)\right|^2 + R^2_{a^{22}_2} \left| BW\left(m_{a_2}, \Gamma_{a_2}\right) Y^2_2\left(\Theta_{\eta_{a2}}, \Phi_{\eta_{a2}}\right)\right|^2 \end{aligned} \tag{6.1}$$

Figures 6.5, 6.6 and 6.7 show the three possible projections of the Dalitz plots 6.3 and 6.4. Monte Carlo projections are plotted on top of data (grey plots). The lower plot in each figure shows the ratio of both histograms.

Table 6.1 shows the masses of the resonances used in the weight function 6.1 as determined in the fit, as well as the physical values, taken from [PF96]. It is obvious that even the preliminary weight function 6.1 used in this Dalitz plot analysis already yields usable results, despite the fact that interference terms were omitted. The ratios of data- and MC histograms are in very good agreement, especially in the areas of resonances. This indicates that the fit has converged and can also serve as a "proof of concept" for the suitability of Evolutionary Strategies to perform a Dalitz plot analysis. Please note, though, that these are very preliminary results of the CB/ELSA collaboration, shown solely to demonstrate the suitability of the EVA library for doing partial wave analysis.

Even better results might be obtainable when using a weight function that includes interference terms. Indeed table 6.1 shows a deviation between measured and real mass especially for the S_{11}, despite its strong signal. But it should also be noted that this could

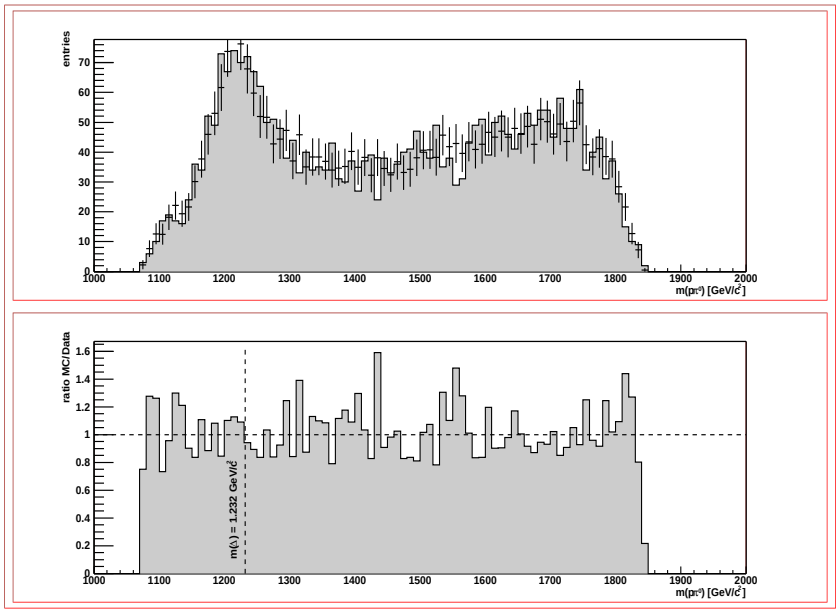

Figure 6.7: *a) The plot shows the projection of Dalitz plot 6.4a on the $p\pi^0$ mass scale (grey histogram). Plotted on top of it is the projection of the corresponding MC Dalitz plot (histogram with error bars). Again both histograms agree well within the error boundaries. b) shows the ratio of both histograms.*

also be related to the fact that this particle is close to the phase space boundaries. In comparison the mass of the a_2, which is located further away from the phase space boundary, is reproduced much better despite its weaker signal.

6.2.1 Achievements of the Optimization Procedure

Apart from the reduction in processing time through parallel execution (compare section 6.3 and especially figure 6.8 for a quantitative comparison), another motivation for using EVA as the basis for the analysis of the $p\gamma \rightarrow p\pi^0\eta$ reaction was to get an independent confirmation of EVA's alleged ease of use – performing a Dalitz plot analysis was not foreseen as as a use case for the deployment of Evolutionary Strategies during the design phase of the EVA library.

The results shown in the last section allow the conclusion, that the deployment of the EVA library in the context of Dalitz plot analyses can be considered to be successful. The development of the "individual" class, not taking into account the time needed to write

Particle	physical mass $[GeV/c^2]$	fitted mass $[GeV/c^2]$
Δ	1.232	1.211
S_{11}	1.535	1.490
a_0	0.983	0.989
a_2	1.318	1.324

Table 6.1: Fitted and physical masses of particles in figures 6.3 and 6.4

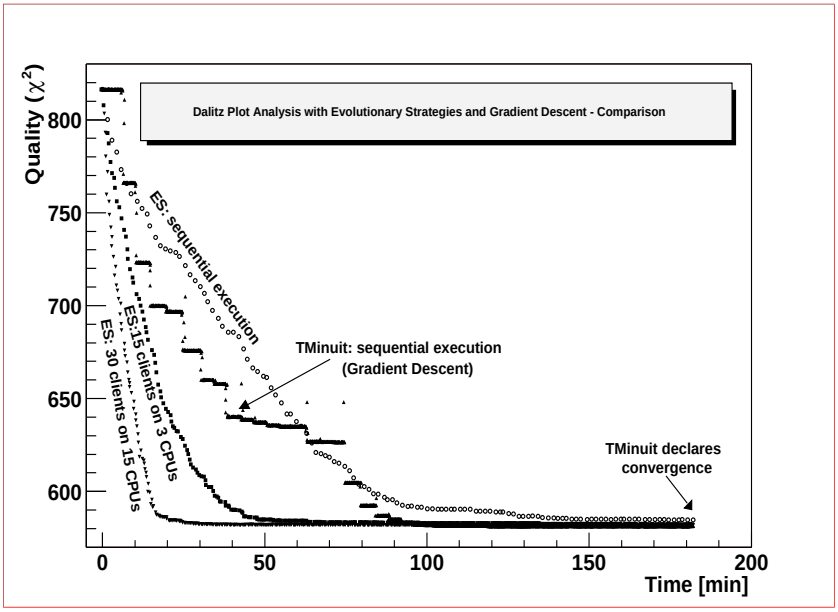

Figure 6.8: *The plot shows four different optimization runs in a Dalitz plot analysis. The quality of the fit is shown as a function of the time needed. Sequential execution of Evolutionary Strategies (circles) is slowest, followed by the gradient descent with TMinuit (curve with "steps"). Execution of 30 clients on 15 CPUs is fastest. All four optimization runs achieve almost identical results.*

the evaluation function, took around an hour, and no major difficulties were encountered.

Even less development effort would have been necessary if EVA's interface to the ROOT library would have been used, as in this case only the evaluation function (i.e the weight function 6.1) needs to be supplied. No knowledge of the inner workings of EVA are required in this case. A certain overhead in compute time must be accepted, however, as ROOT is built around an interpreter. See section A.5 for further information on the EVA library's interface to ROOT.

The large number of parameters of the weight function again demonstrates the need for minimization algorithms. Even when just testing 10 values for each of the 16 parameters, if the evaluation of one parameter set takes a second, the total compute time would have amounted to 10^{16} seconds, equivalent to $3.17 * 10^8$ years. Even though both gradient descents and, to a lesser extent, Evolutionary Strategies exhibit the danger of not finding the global optimum of a fit function, the usage of such minimization algorithms thus can not be avoided.

6.3 Comparison with Standard Procedures

The standard algorithm used for minimization in a Dalitz plot analysis is a gradient descent (see section 3.3.1). Unfortunately the standard tools for gradient descents in particle physics do not use parallel execution to speed up the minimization.

Figure 6.8 shows four different optimization runs in a Dalitz plot analysis. Three use

Evolutionary Strategies, running on between 1 and 15 CPUs in parallel, a fourth run uses a gradient descent implemented by the ROOT framework's `TMinuit` class. The quality of the fit is shown as a function of time. The three fits involving Evolutionary Strategies were stopped when they exceeded 182 minutes, the time needed by TMinuit to declare convergence. All four optimization runs had achieved almost identical values by then and had converged. Sequential execution of an EVA-based fit turned out to be slightly slower than the gradient descent-based fit. EVA was also executed with 15 child individuals on 3 machines of different specifications[4], a technical environment that should be available in most Universities. A significant speedup both over sequential execution of EVA and over TMinuit can be observed as a consequence of parallel execution. The fastest optimization run was done with 30 child individuals running on 15 CPUs. The parameter space can be checked more thoroughly due to the larger number of children and more CPU time is avaible to each child individiual. It is therefore not surprising that this optimization needed the least time of all four runs. An additional factor of 2 in the speedup could have been achieved by executing each child individual on its own CPU.

It is clear that parallel Evolutionary Strategies lead to significant time savings compared to sequential execution of a gradient descent-based fit. And the fact that both the gradient descent and the optimization based on Evolutionary Strategies (ES) find the same optimum reiterates that both algorithms can be treated as equal in their optimization power. With noisy input data ES might even have an additional advantage over a gradient descent due to the ability to bypass local optima of an evaluation function.

It must be said that there are still unanswered questions regarding a regular deployment of EVA in doing Dalitz plot analysis. It is, as an example, not clear how to determine an error on the measured masses shown in table 6.1. But at the very least, as the weight function 6.1 is independent of the minimization method used, EVA's superior ability to perform the fit in parallel can be used to obtain good start values close to the global optimum, which can then be used in more standard, sequential minimization routines such as MINUIT or TMinuit. It should thus be possible to reduce the overall execution time of a Dalitz plot analysis by a large amount in this way.

Evolutionary Strategies might also allow for an almost interactive work style in the design phase of the weight function 6.1. Test and design of a Dalitz plot analysis take up significantly more time than the final fit, which can then be done with a gradient-descent based method.

The "stairs", representing the check of the function values of all partial derivates, in the plot for the TMinuit-based optimization in figure 6.8 also make it clear that TMinuit itself might benefit from parallelisation using similar techniques as those deployed in the EVA library.

[4] 1 Athlon 2500+, a Celeron 2 GHz and a Dual PIII-1GHz, all with 1 GByte of main memory

Chapter 7

Summary and Outlook

Most particle physics analyses implicitly comprise a number of optimization tasks – be it the reconstruction of a charged track in a drift chamber or the "fitting" of a mathematical function to a histogram. Some optimization tasks however, while having the potential to yield significant improvements of an analysis, exhibit a larger scale than can be realistically handled by a single computer in acceptable time. Parallel execution of Evolutionary Algorithms thus provides a cost-effective and easy-to-use alternative to more standard algorithms running on a single machine, such as a gradient descent.

The work presented in this thesis deals with automated parametric optimization in particle physics analysis. As an example the quality of a particle signal in a histogram, measured with a figure of merit such as the squared significance, can be increased by a significant factor, despite the fact that the original analysis was done carefully and included the manual optimization of the particle signals. The optimization potential in part stems from the fact that, in the presence of a large number of parameters, it can become very difficult to manually choose the optimal set of cuts, as quality criteria for a particle signal other than the signal-to-noise ratio can relate to the best set in non-obvious ways. As an example, the squared significance can be increased when doubling the number of background events while increasing the number of entries in the signal by a factor of 1.5 only. In such environments, automated parametric optimization studies can yield very good results.

The danger of creating fake contributions to particle signals through the usage of automated paremetric optimization procedures is being investigated in this thesis and is found to be small, albeit present. Techniques to circumvent this problem are discussed.

In an idealized benchmark the EVA library – an implementation of parallel Evolutionary Algorithms developed as part of this thesis and used in the optimizations – is able to reduce execution time of an automated optimization from hours to minutes, involving 128 child individuals running on the GridKa cluster of Forschungszentrum Karlsruhe. Similarly, in a real-world optimization of the $D_s^+ \to \Phi\pi^+$ decay channel of the D_s^+, using measured data taken at *BABAR* and involving 20 child individuals, execution time goes down from

38 hours for the sequential execution to under 2 hours. This optimization yields a 19.4% improvement of the chosen figure of merit for the signal, the squared significance $\mathcal{S}^2$. An improvement of 45% can be seen in the decay channel $D_s^+ \to \bar{K}^{*0}K^+$ of the D_s.

Other minimization and maximization tasks, such as Dalitz plot analyses, also benefit from the parallel execution, allowing for a more interactive work style through shorter execution times. EVA proves, in this context, to be able to achieve similar results to a gradient descent, albeit in much shorter time.

Depending on the paradigm at hand parallel computation can be a difficult task. Parallel programs are not easy to debug and may exhibit complex error sources such as dead-locks and synchronization problems. Care has been taken in the design of the EVA library to provide users with easy means to debug their code. More specifically, apart from the ability to perform optimizations in parallel using either POSIX threads or the Message Passing Interface ("MPI"), EVA also comprises the possibility to peform optimizations sequentially, helping users to debug their code in a conventional environment. This way error sources coming from parallel execution can be excluded, and errors due to dead-locks and other difficulties related to parallel execution are confined to the EVA library itself. This should overall lead to more robust code.

An intuitive interface to a standard tool of particle physics analysis – the ROOT framework – has been developed for EVA, such that the evaluation of individuals of an Evolutionary Strategy can be done inside the ROOT C++ interpreter. This also means that existing code used for the manual optimization of a particle signal can be used as part of the automated parametric optimization.

The EVA library is one of the first toolkits to combine the ability to perform parallel parametric optimizations with an intuitive interface to the ROOT framework, targeted at the particle physics community, while providing features like the ability for easy debugging and a common interface to both Genetic Algorithms and Evolutionary Strategies.

Future plans for the EVA library include the development of an interface to the Parallel Virtual Machine (PVM), a library similar in scope to MPI, as well as the development of a Web services interface. As EVA will be made available as Open Source, covered by the GNU General Public license, public contributions can be easily incorporated into the library, leading to an even more robust and feature-rich environment.

Appendix A

Using the EVA library

A.1 Installation

A.1.1 How to obtain EVA

EVA is available as source code from *http://www.ep1.rub.de/~ruediger* or via eMail from the author (send an email to *ruediger@berlich.de*).

A.1.2 Requirements

In order to compile and use the EVA library, you need at least one Linux machine equipped with a relatively recent C++ compiler. So far the library has been tested with g++ version 3.2 and 3.3. In order to benefit from EVA's POSIX thread personality, a cheap dual-processor machine is sufficient. A cluster or at least a set of networked Linux machines is recommended in order to benefit from EVA's MPI personality. The EVA library has so far only been tested with the MPICH implementation of the MPI API. Memory requirements depend on your application, as the main memory consumption stems from the individuals being used in a population.

A.1.3 Compilation and Installation

EVA is equipped with a `configure` script generated using the GNU autoconf and automake utilities. All that should be necessary to compile EVA is to issue the commands in listing A.1 in the EVA directory. The last command should be issued as user root. You will then find the library in the directory specified via the `--prefix` switch. In order to use shared libraries you then need to add EVA's `lib` directory to the file `/etc/ld.so.conf`[1] and to

[1]The directory might be different for Linux distributions other than SuSE Linux

Listing A.1: *Commands needed to compile and install the EVA library*

```
evadir> ./configure --prefix=/where/EVA/should/be/installed
evadir> make
evadir> su
evadir# make install
```

Listing A.2: *Setting up shared libraries*

```
/root # cd /etc
/etc # vi ld.so.conf
/etc # ldconfig -v
```

run the command `ldconfig -v` as user root. See listing A.2.

A.2 Documentation

The EVA library's classes and member functions have been extensively documented with the help of the *Doxygen* framework. In addition to the information contained in this chapter it is thus possible to obtain a reference manual by running the `doxygen` command in the EVA library's root directory. Please make sure that the `dot` program out of the `graphviz` package is installed, so class diagrams can be included in the reference documentation. `graphviz` is part of most modern Linux distributions.

A copy of the reference manual should also already be available in the EVA distribution.

A.3 Usage

You have now learned the basic structure of the EVA library. The following section contains a "hands-on" introduction to creating individuals and setting up populations. Users should then be able to write their own code and individuals based on the EVA library.

A.3.1 Overview

EVA is designed as a class library rather than a monolithic application. This means that a user has to assemble and compile a program, based on the means provided by this library. While this puts some additional burden on the user, the added flexibility and extensibility makes this approach worthwhile. It is assumed that the target group for this library - physicists with the need for long lasting optimization studies - has sufficient knowledge of the C++ programming language, so that the two main tasks to be performed by the user don't represent a major hurdle.

In the context of Evolutionary Algorithms, optimization means that a quality is assigned to a set of conditions (usually represented by numerical or binary parameters). In other words, an optimization problem is represented as a function $f(x_i)$[2], where x_i is a set of parameters. It is up to the EVA library to vary the parameter set in a way that $f(x_i)$ is minimized or maximized (depending on the problem). It is up to a user to specify $f(x_i)$ and to provide EVA with information about the number and characteristics of the parameters x_i. Currently EVA "understands" numerical parameters (represented as the `evaDouble` class) and bit parameters (represented by the `evaBit`, `evaBitAP` and `evaBitset` classes).

In order to define an optimization problem, a user must derive a class[3] from the `evaIndividual<T>` class (in the case of parameters specified as `evaDouble`'s or `evaBit`'s) or from the `evaBitset<N>` class (for a predefined collection of bits)[4]. `T` is a C++ template parameter and stands for the type of a parameter. These two classes contain a purely virtual C++ function (henceforth called "evaluation function") `virtual double customValue()=0;` that must be overloaded[5] by the user in his derived class. This function is used to assign a quality (or fitness) to a given parameter set. Setup of parameters can be done in the derived classes' constructor. As `evaIndividual<T>` is derived from the STL `vector<T>` class, it can also be set up using the standard STL means for filling vectors. `evaBitset<N>` is an extension of the STL `bitset<N>`, with N being the size of the bitset.

In addition to basic "value" classes an `evaIndividual<T>` can also hold other `evaIndividual<T>`'s, simply by using the latter as a template parameter.

Once the user has set up these individuals, he needs to initialize a population of them. This is usually done in `main()`. It is at least necessary to specify the type of the individuals used in a population, the size of the population, and the number of parents. Finally the population's `optimize()` function is called, which tries to search for a minimum or maximum of `customValue()` by varying the parameters.

These two steps will now be explained in more detail with a few examples, introducing more options of the library along the way.

A.3.2 Searching the minimum of a parabola

Probably one of the easiest examples of a minimization problem[6] is the search for the minimum of a parabola $f(x) = x^2$. This function has just a single input parameter and a minimum at $x = 0$. It is therefore possible to check that EVA returns the correct minimal

[2] Usually called fitness

[3] henceforth called an "individual"

[4] Alternatively, a predefined individual, "`evaRootIndividual`", provides a convenient interface to the ROOT framework, such that researchers can use ROOT scripts to perform the evaluation of parameters and do not have to create their own individuals. See section A.5 for further information

[5] i.e. defined

[6] "Hello World", the EVA way ...

Listing A.3: *Class declaration of the basic "parabola" individual*

```
class parabola
  :public evaIndividual<evaDouble>(1)
{
  public:
    parabola();
    double customValue(void);
};
```

Listing A.4: *The* `customValue()` *member function of the "parabola" individual*

```
double parabola::customValue(void)
{
  // initialize the result
  double dummy, result=0.;

  // parabola is an extension to the STL vector class and can use its
  // member functions. At a later time, we might want to use a parabola
  // with a higher dimension, so we will do the calculation of the
  // parabola in such a way that we do not depend on a fixed dimension.
  int i, size_this = size();

  for(i=0; i<size_this; i++)
  {
    // We can ask for the evaDouble's value using the value() function.
    // We have to use the -> operator because our STL vector contains
    // pointers to evaDouble's. The at() function is one of two possibilities
    // to gain access to a component of an STL container class. In our
    // example, we could alternatively use the construct
    // dummy=(*this)[i]->value(); But at() also does some error checking
    // and is easier to write, so we use it instead.
    dummy=at(i)->value();

    // e.g. f(x,y)=x^2 + y^2
    result += dummy*dummy;
  }

  return result;
}
```

value for x. This minimization problem will now be implemented in the EVA framework. The first step is to derive a class from `evaIndividual<T>` (see listing A.3).

Obviously the `evaIndividual<T>` will contain objects of the type `evaDouble`. The `evaDouble` class behaves mostly like a standard double value, with a few exceptions that will be explained further below. So far, our parabola class is effectively a STL vector containing one or more pointers to an evaDouble. `evaIndividual<T>` of course provides some additional functionality over the STL vector class that a user will rarely have to care about, but which is needed internally for the optimization process.

What remains to be done is to declare and define a few member functions. First and foremost, a suitable `customValue()` function mus be created (see listing A.4).

Listing A.5: *The default constructor of the "parabola" individual*

```
parabola::parabola()
  :evaIndividual<evaDouble>(1)
{
  at(0)->reInitialize(-100.,100.);
}
```

Listing A.6: *The header file for the parabola individual (parabola.hh)*

```
#ifndef PARABOLA_HH
#define PARABOLA_HH

#include "evaindividual.hh"
#include "evadouble.hh"

class parabola
  :public evaIndividual<evaDouble>
{
  public:
    parabola();
    double customValue(void);
};
#endif /* PARABOLA_HH */
```

Next, a constructor is needed. In the current example it only has to preload the `evaIndividual<T>` with a single `evaDouble` and initialize it with a suitable value. Listing A.5 shows how this can be done.

The "1" means, that there will be just one pointer to an evaDouble stored in the `evaIndividual<evaDouble>` class. The evaDouble will be initialized using its default constructor. Unfortunately this also means that it will be initialized with a random number somewhere in the range of $[-1 : 1]$, which makes this example less interesting. The evaDouble that was created by the constructor of `evaIndividual<evaDouble>` can be accessed as the first element of the STL vector, i.e. with `at(0)`. It can be re-initialized randomly in another value range using evaDouble's `reInitialize(double mn, double mx)` function.

At this point all necessary steps needed to create an individual suitable for deployment in the EVA framework have already been done. The resulting header file is shown in listing A.6.

In the next step a basic population will be created. This is done in `main` (see listing A.7).

The comments in the source code should provide a sufficient description of what is done there. One additional detail is the fact that the parabolas created in the beginning are not deleted in the end. This is done by the `evaPopulation<parabola>` destructor instead, so users do not have to care for this.

Listing A.7: *The* `main() function for the parabola example`

```
#include "parabola.hh"
#include "evapopulation.hh"

int main(int argc, char **argv)
{
  int i;

  // Create a population of parabola classes ...
  evaPopulation<parabola> par;

  // ... and preload it with a number of parabolas.
  // Please note that we are using the STL function
  // push_back() here ! An evaPopulation<parabola>
  // is basically a vector<parabola *> with some
  // additional code.
  for(i=0; i<10; i++) par.push_back(new parabola());

  // Specify the number of parents ...
  par.setNParents(2);
  // ... and the maximum number of generations
  par.setMaxGeneration(100);
  // We want success reports after each generation
  par.setReportGen(1);

  // Now start the optimization cycle
  par.optimize();

  return SUCCESS;
}
```

With a suitable `Makefile` the parabola example can now be compiled. A sample Makefile is shown in listing A.8. It assumes that the EVA library and its header files are stored below `/opt/eva`.

The output of the compiled program will look similar to listing A.9.

Please note that some parts of the printout were left out in order to save space, so the output will look slightly different in most cases. What you see is information about the best individual of each generation. The printout is initiated by the population class `evaPopulation<parabola>`, which calls the `doInfo()` function of the first (i.e. best) individual in the population in each generation.

Unfortunately, the output looks far from optimal. For example, we would like the parabola to emit information about its input parameter, and the algorithm has used the default name "unknown" to denote the individual. So in our next step we'll add an overloaded version of `evaIndividual<T>`'s `doInfo()` function to the `parabola` class so it prints out more suitable information. By the same time the parabola will be given a name, which should be done in the constructor. See listing A.10 for further information.

Now the output will look similar to listing A.11.

It is normal that the values differ from the first run, as the evaDouble is initialized randomly, using a different seed for the random number generator for every new run. Being an

Listing A.8: `Makefile` *for the parabola example*

```
TOPDIR=$(PWD)
EVADIR=/opt/eva
INCDIR=$(EVADIR)/include
LIBFLAGS=-L$(EVADIR)/lib

CPP = g++

CPPFLAGS = -O2 -Wall -fexceptions
INCLUDE = -I$(INCDIR) -I$(PWD)
LIBS = -lc -lm  -lxml2 -lz -leva

FILES = $(TOPDIR)/main.o \
        $(TOPDIR)/parabola.o

.cc.o:
        $(CPP) $(CPPFLAGS) $(INCLUDE) -c $<

parabola: $(FILES)
          $(CPP) $(FILES) $(LIBFLAGS) $(LIBS) -o parabola

clean:
          rm -f *.o *~ parabola
```

Listing A.9: *Output of the parabola program*

```
In generation : 1
Individual unknown has value 46.3541
In generation : 2
Individual unknown has value 24.3591
[...]
In generation : 100
Individual unknown has value 2.53011e-06
```

Listing A.10: *Overloaded version of the* `doInfo()` *function and new constructor*

```
parabola::parabola()
  :evaIndividual<evaDouble>(1)
{
  at(0)->reInitialize(-100.,100.);
  setName("best");
}

void parabola::doInfo(void)
{
  cout
  << "parabola " << getName() << " has value " << value() << endl
  << "The input parameter has the value " << at(0)->value() << endl;
}
```

Listing A.11: *Output when using the new* `doInfo()` *function*

```
[...]
In generation : 100
parabola best has value 7.89781e-07
The input parameter has the value 0.000888696
```

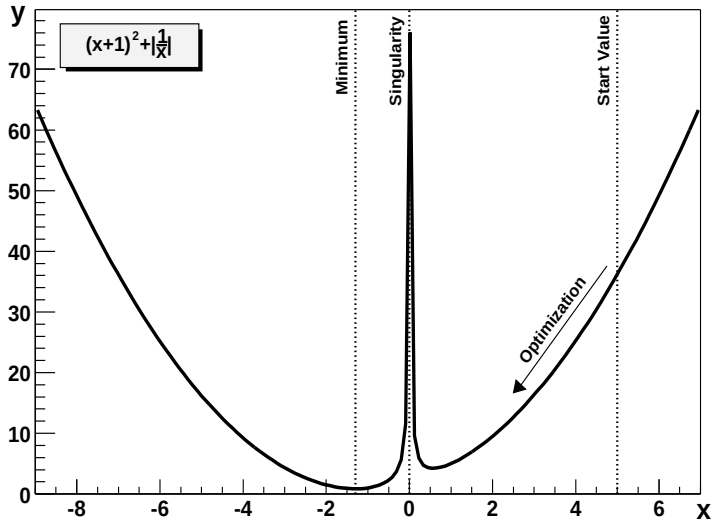

Figure A.1: *The plot shows a function with a singularity at 0 and a minimum close to −1.3, left of the singularity. Minimization startes on the right side of the singularity. The function is designed to demonstrate the Evolutionary Strategies remarkable ability to bybass even the most cumbersome local optima.*

optimization package, EVA will converge towards an absolute or local optimum, but will quite possibly never really reach it. So it is normal that the input parameter is still not exactly 0. after a hundred generations.

A.3.3 Restricting the value range

In the next step a local parameter will be added to the parabola class in order to allow a meaningful discussion of the conversion of individuals to and from XML. Furthermore some additional functionality of the evaDouble class will be shown. To do this, a 5 dimensional function $f(\vec{x}) = \sum_{i=1}^{5}((x_i + 1)^2 + |\frac{1}{x_i}|)$ will be used, with a minimum close to -1.3 and a singularity at 0 in each dimension, instead of a simple, one-dimensional parabola. This means that 5 input parameters are needed instead of 1. Picture A.1 shows the function for the one-dimensional case.

The joint start value for all 5 input parameters will be stored in a private `double` variable. Start values should be adjustable via the `setStartValue()` function that must be implemented. The new class declaration can be found in listing A.12. You will find that it has now been changed to include the "usual" functions of a class (e.g. a copy constructor and an `operator=()`) plus some functions particular to the EVA library. The new functions and functionality will be introduced along the way.

We will begin with the constructor (see listing A.13). By default. the optimization will use a start value of $(5., 5., 5., 5., 5.)$ in our case, which means that the minimization procedure has to "jump" over the singularity in order to get close to the absolute minimum. It would be good to be able to avoid potential problems related to the singularity[7], so a small value

[7] such as a division by 0

Listing A.12: *Class declaration for parabola individual with an added singularity*

```
class parabola
   :public evaIndividual<evaDouble>
 {
 public:
   parabola();
   parabola(const parabola&);
   ~parabola();

   void clearall(void);

   void operator=(const parabola&);

   double customValue(void);
   void doInfo(void);

   string convertToXMLCore(void);
   void loadFromXMLCore(xmlNodePtr);

   void setStartValue(double);

 private:
   double startValue;
 };
```

Listing A.13: *The constructor of the new parabola individual*

```
const int parSize=5;

parabola::parabola()
  :evaIndividual<evaDouble>(parSize)
{
  int i;

  // Initialize the start value, so
  // we know where to start.
  startValue=5.;

  for(i=0; i<parSize; i++){
    // Restrict the search area to [-8,8.]
    at(i)->setClosedBoundaries(-8.,8.);
    // Add a gap around 0 so we don't tap
    // into the singularity
    at(i)->setGap(-0.001,0.001);
    // Set the step width and its adaptation
    at(i)->setSigma(0.8);
    at(i)->setSigmaSigma(0.008);
    // And provide a start value
    *at(i)=startValue;
  }

  setName("best");
}
```

range around the singularity will be excluded from the calculations. This is done by adding a gap to the value range of the `evaDouble` using the `evaDouble::setGap()` function. Crucial for the success of the optimization is the choice of the sigma parameter (set using `evaDouble::setSigma()`) used in the mutation of all input variables. If it is too small, variations of the evaDouble parameters won't succeed in getting beyond the singularity, whose effect is limited to a range of roughly ± 1 around its middle value 0. If sigma is too large (e.g. : larger than the value range of each evaDouble), variations of each parameter resemble random jumps in the input space. In our experience, sigma shouldn't be larger than about 10% of the search area (i.e. $sigma <= 1.6$ in our case) and should be the smaller the finer the structures are that are being explored. The EVA library implements a dynamic adaptation of sigma (steered by the size of the "sigmasigma" parameter, set using `evaDouble::setSigmaSigma()`). However, in some cases this algorithm won't be able to adapt sigma quickly enough, especially if the topology of the quality surface of `customValue()` changes quickly. The effect of an inappropriate sigma is a stagnation in the quality improvement of individuals[8]. Unfortunately if - unlike our example - the exact location of the optimum isn't known, it is not always possible to distinguish this from the case where the absolute minimum has been found. See further below for a suggestion to overcome this problem.

The explanation of the fitness calculation and the creation of the `doInfo()` function will be skipped here, as it has already been explained in detail for the simple parabola above. It should also be clear how to write a `setStartValue()` function[9] so it won't be discussed in detail here either.

The `operator=()` function should be mentioned, however, as it shows how to make sure the parent classes' data is copied the way it is meant to be copied. This function can be found in listing A.14. The copy constructor is shown alongside `operator=()` in order to show their similarity. Similarly, the same listing shows how to clear or reset the internal data structures, either using the destructor or the `clearall()` function. Its services can be used by the destructor. In order to make sure that all data structures are cleared, `clearall()` needs to call the parent classes' `clearall()` function.

A.3.4 XML generation

Much more important than these, however, are the two functions `loadFromXMLCore()` (listing A.16) and `convertToXMLCore()` (listing A.15). They are part of the framework which is used to parallelize Evolutionary Algorithms in the EVA library and are also used to store and load intermediate results.

[8] This is only true for the most common selection mechanism, where new parents are selected from both parents and children. Compare figure 3.1

[9] Please note that this function needs to re-initialize the parabola's `evaDouble` parameters with `startValue`.

Listing A.14: *Copying and clearing data structures*

```
/*************************************************************/
// The copy constructor
parabola::parabola(const parabola& cp)
   :evaIndividual<evaDouble>(cp)
{
  startValue=cp.startValue;
}

/*************************************************************/
// The destructor makes in an EVA individual usually uses
// the services of the clearall() function.
parabola::~parabola()
{
  parabola::clearall();
}

/*************************************************************/
// Make sure startValue and evaIndividual are copied the
// way we want it.
void parabola::operator=(const parabola& cp)
{
  startValue=cp.startValue;

  // let evaIndividual handle the rest
  evaIndividual<evaDouble>::operator=(cp);
}

/*************************************************************/
// Reset/clear internal data structures
void parabola::clearall(void)
{
  // restore the original value
  startValue=5.;
  // clear evaIndividual<evaDouble> data structures
  evaIndividual<evaDouble>::clearall();
}
```

Listing A.15: *Converting a parabola object to XML*

```
string parabola::convertToXMLCore(int offset, int addOffset)
{
  ostringstream result;

  result
  << blanks(offset) << convertVarToXML(startValue,"startValue","")
  << blanks(offset) << "<evaIndividual>" << endl
  <<               evaIndividual<evaDouble>::convertToXMLCore(offset,addOffset)
  << blanks(offset) << "</evaIndividual>" << endl;

  return result.str();
}
```

Listing A.16: *Loading a parabola object from XML*

```
void parabola::loadFromXMLCore(xmlNodePtr xp)
{
  xmlNodePtr node = xp;

  while(node != NULL){
    if(nodeHasName(node,"startValue"))
      startValue=dGetProperty(node,"");
    else if(nodeHasName(node,"evaIndividual"))
      evaIndividual<evaDouble>::loadFromXMLCore(node->children);

    node=node->next;
  }
}
```

In EVA terminology, a "core" XML description is XML code without the outermost envelope. As it is not clear who initiates the XML generation (quite often this is done by the `evaPopulation` class or a class derived from the current class), it is not clear what the name of the envelope will be. So it is up to the calling class to create the envelope, while the class whose `convertToXMLCore()` function is called contributes the content. The calling class also determines how many blanks should be inserted in front of the XML description in order to create formatted output.

With this information it should now be possible to understand listing A.15. There is one local parameter, and users need to care for its conversion to XML themselves. For basic types like `double`, `long`, etc., this is done with the `convertVarToXML()` family of functions. They are defined in the evaMember class. More information about them can be found in the reference documentation. An excerpt from the XML code created by `convertToXMLCore()` can be found in listing A.18.

Users need to take care to initiate the XML conversion of a parent class. It has its own copy of the `convertToXMLCore()` function and, in turn, takes care of the conversion of its parent class (`evaMember`) to XML. However, an envelope around the core XML description of the `evaIndividual<evaDouble>` class must be added. Again see listing A.15 for information on how this is done. The recursive way in XML description of classes are created makes sure that the user has to invest but minimal effort into the creation of the XML code. It should be pointed out that the user does not have to care for the creation of XML code for the evaDouble objects stored in the parabola object. This is done automatically by the `evaIndividual<evaDouble>` class. In `convertToXMLCore()`, a string containing the whole core XML code of the class (including that of its parent classes) is returned and can be processed further by the calling class.

Just like `convertToXMLCore()`, `loadFromXMLCore()` (see listing A.16) uses its parent classes' version of the `loadFromXMLCore()` function to do most of the job of loading an XML description. Starting with a node (a data structure from the GNOME foundations libxml2 package) of an XML tree, this function checks for those XML tags that were created in `convertToXMLCore()` (see listing A.15). If found, it either reloads the parameter

associated with this name from XML (like in the case of "startValue") or it hands over an `xmlNodePtr` to the `loadFromXMLCore()` function of its parent class (or the objects stored in a vector in the case of a container class). Please note that loading of the `evaDouble`'s XML code is already handled by the `evaIndividual<evaDouble>` class.

An `xmlNodePtr` is really a linked list, and so it is possible to get access to the next node using the `node=node->next` construct. Two functions help with the scanning of the XML tree. `nodeHasName()` helps to check whether a given node has the desired name. The `[x]GetProperty()` (with [x]=d for `double`, l for `long`, etc.) family of functions helps to convert XML descriptions of basic types to their real-world counterparts[10]. These functions implement the reverse action to the `convertVarToXML()` family of functions. Again, see the detailed description in the evaManual chapter of the reference manual for further information.

A.3.5 Loading and saving a population

Now that it was shown how to convert individuals to XML it is also possible to make disk copies of our population during the optimization. This also allows to reload an XML description, should it be necessary to continue searching for optima after a first run was finished. In order to do this it is necessary to slightly modify the `main()` function (see listing A.17). Safety copies can be initiated using the function `evaPopulation<S>::setSaveTurn(long st)`. The parameter `st` specifies the number of generations, after which a copy of the population should be made. Internally this is done using the `evaMember::saveFile(const char *fn)` function, which is also accessible to the user to save an XML description of arbitrary classes derived from the evaMember class to a file with filename `fn`. Please note, however, that in order to use this function your class needs to implement a local version of `convertToXMLCore()`. By default, EVA uses the name `evadefault.xml` for the safety copies. You can set the name using the function `evaMember::setSaveFile(const char *)`. You can reload the file using the `evaMember::loadFile(const char *)` call. In listing A.17 this mechanism is used to commence training after a first run was finished.

In order to do this, with the `main()` function of A.17, a user only has to specify the name of a safety copy generated using evaMember::saveFile() on the command line. Please note that only minimal error checking is implemented in this example in the `main()` function. The `loadFile()` function itself implements some error checking, though, and returns a boolean value "`true`" if loading was successful, "`false`" otherwise.

Listing A.18 shows an excerpt of the XML code that was generated for our parabola class. The whole XML description has a size of below 150 kilobytes. In the XML code, you will find two tags with the name `evaIndividual`. This is just a consequence of the fact that both the population and the `parabola` objects stored in it are derived from

[10]Both `nodeHasName()` and `[x]GetProperty()` are implemented in the `evaMember` class.

Listing A.17: *Initiating safety copies from* `main()`

```
int main(int argc, char **argv)
{
  // [...] Code prior to this line erased to save space

  // How often do we want to make safety copies of our
  // population ? Every 100 generation !
  par.setSaveTurn(100);

  // If the filename of a safety copy
  // was provided by the user, load it
  // into the population.
  if(argc==2){
    if(!par.loadFile(argv[1])){
      cout << "Error : could not load file " << argv[1] << endl;
      exit(1);
    }
  }

  par.optimize();

  return SUCCESS;
}
```

the `evaIndividual<T>` class. As each layer's nodes are processed by a different classes' `loadFromXMLCore()` function, there are no clashes with regards to the names being used.

A.3.6 Changing parameters of a population

Of course this mechanism can be used to selectively change parameters, after the XML description of a population was loaded. All that needs to be done for this is to call the corresponding functions after the XML description was loaded. In our example, this could be done by modifying the section next to `loadFile()` in listing A.17. Listing A.19 shows how this can be done.

Again, this example makes extensive use of the STL features of the EVA library. Both the `parabola` class and the `evaPopulation<S>` class are "just" extensions of the STL `vector<T>` class, so its functions can be used. For example, it is possible to ask for the size of a population using the `vector<T>::size()` function and the same can be done for the parabola, whose size is needed to determine the number of `evaDouble` object pointers stored in it. Furthermore, access to the `parabola` objects stored in the population is possible using the STL (and array) `par[i]` construct. For practical reasons, however, the STL function `at(j)` is used to do the same for the `parabola` objects in our example.

Listing A.18: *Excerpt from the XML code generated by* `evaMember::saveFile()`

```
<saveFile>
  <generation val="1000"/>
  <maxGeneration val="1000"/>
  <reportgen val="10"/>
  <saveturn val="100"/>
  <selectMethod val="1"/>
  <recombineMethod val="0"/>
  <maximize val="false"/>
  <trackReelection val="false"/>
  <hname val="dh-152-03"/>
  <dname val="ep1.ruhr-uni-bochum.de"/>
  <evaIndividual>
    <component pos="0">
      <startValue val="5"/>
      <evaIndividual>
      <component pos="0">
        <iValue val="-1.31615"/>
        <sigma val="0.0798139"/>
        <sigmasigma val="0.0008"/>
        <sigmaMutateMethod val="1"/>
[...]
```

Listing A.19: *Modifying parameters after loading a safety copy in* `main()`

```
  //[...]
  // If the filename of a safety copy
  // was provided by the user, load it
  // into the population.
  if(argc==2){
    // Can it be loaded ? Emit an error if not
    if(!par.loadFile(argv[1])){
      cout << "Error: could not load file " << argv[1] << endl;
      exit(1);
    }
    else{
      // Find out about the size of the population
      // Please note that we're using plain STL
      // functions for this.
      int popSize=par.size(), parSize=0;

      // Let's try a smaller sigma and sigmasigma
      // for each evaDouble
      for(i=0; i<popSize; i++){
        parSize=par[i]->size();

        for(j=0; j<parSize; j++){
          par[i]->at(j)->setSigma(0.08);
          par[i]->at(j)->setSigmaSigma(0.0008);
        }
      }
    }
  }
  //[...]
```

A.3.7 Multi-populations

In the EVA library, `evaPopulation<T>`s are a superset of the `evaIndividual<T>` class. They can thus be used as individuals and compete against each other within a super-population. Apart from memory and computing time restrictions, there are no limitations to the depth of such "multi-populations", i.e., the competing populations could themself be super-populations, and so on.

Most of the framework needed to build multi-populations is already in place. All remaining modifications of our example program can be done in `main()`. Listing A.20 shows in a few simple steps how a multi-population is set up. As the code is extensively documented and should be self-explanatory, only a few special features will be discussed below.

The super-population is created by using a population as a template parameter. If the population is an `evaPopulation<parabola>`, the super-population is of type `evaPopulation<evaPopulation<parabola> >`. C++ requires that, if a template parameter is itself a templatized object, there must be a space between the two concluding "`>`" signs. If it is omitted, the parser will emit an error message. A super-super population could accordingly be set up as `evaPopulation<evaPopulation<evaPopulation<parabola > > >`. Please note that up until this point the populations at all levels have no members. Internally, e.g. the `evaPopulation<parabola>` is just a `vector<parabola *>` of size 0 with some additional code. So the populations need to be filled with members, which is done in the for-loops (see e.g. listing A.20).

The destructor of the `evaPopulation` class takes care of deleting the individuals stored in it. As a direct consequence, memory for these individuals needs to be allocated using the C++ `new` call. For this reason the member populations of the super-population are created as pointers rather than values.

Just like in the case of a basic population, optimization of multi-populations is started using the call `superpar.optimize()`.

A.3.8 Parallelizing populations

Parallelization in the EVA framework is done on the level of populations. Any population can be parallelized, be it a basic population, a super-population or a population at an even higher level. All that needs to be done is to derive one of currently two parallel "personalities" from a population. The population that should be parallelized is specified as a template parameter to the parallel population. Most of the functionality is thus provided by the parent population, the "parallel" population only adds the parallelization framework. As a direct consequence, users, whose duty it is only to provide suitable individuals, do not need to change their individual's code in oder to benefit from parallel execution. This greatly simplifies debugging of user code, as it can be done using a serial population, without the complications usually associated with the debugging of parallel

Listing A.20: *Setting up multi-populations in* `main()`

```
#include "parabola.hh"
#include "evapopulation.hh"

int main(int argc, char **argv)
{
  int i,j;

  // An evaPopulation<parabola> has the same interface as an individual.
  // So we can make it an individual in the same way we've done it for
  // the parabola class itself. Note the blank between the two > > . It
  // is needed if two template classes are nested. Otherwise you'll get
  // errors from the parser.
  evaPopulation<evaPopulation<parabola> > superpar;

  for(i=0; i<10; i++){
    // This part is similar to a basic population. The only difference
    // is that we use pointers. They are needed because the super populations
    // destructor will delete them when the super population is destroyed.
    evaPopulation<parabola> *par = new evaPopulation<parabola>();

    // Add parabolas to the sub-population.
    for(j=0; j<10; j++) par->push_back(new parabola());

    // We'll use a smaller number of generations
    par->setNParents(2);
    par->setMaxGeneration(100);
    // We want the super-population to do the talking
    par->setReportGen(0);

    // Add the sub-populations to super-population
    superpar.push_back(par);
  }

  superpar.setNParents(2);
  // Only few generations are needed
  superpar.setMaxGeneration(10);
  // Emit information after every generation
  superpar.setReportGen(1);

  // Loading and savomg can be done in the same way as for
  // basic populations
  superpar.setSaveTurn(1);

  // If the filename of a safety copy
  // was provided by the user, load it
  // into the population.
  if(argc==2){
    // Can it be loaded ? Emit an error if not
    if(!superpar.loadFile(argv[1])){
      cout << "Error : could not load file " << argv[1] << endl;
      exit(1);
    }
  }

  // superpar has an optimize() function just
  // like any other population. In fact superpar
  // has the same code as a basic population.
  superpar.optimize();

  return SUCCESS;
}
```

Listing A.21: *Setting up multi-populations in* `main()`

```
#include "parabola.hh"
#include "evapopulation.hh"
#include "evapthreadpopulation.hh"

int main(int argc, char **argv){
  // The basic individuals and the population to be parallelized
  // are specified as template parameters
  evaPthreadPopulation<parabola, evaPopulation<evaPopulation<parabola> > > superpar;

  // [...] The rest of the code remains unchanged, so we do not show it here.
}
```

code.

If parallelization is done for a basic population, users should take into account the overhead created by generating and parsing the XML representations of individuals and by sending them over a network. Where the time needed for this is small compared to mutation and value calculation, significant speed-ups can be achieved over sequential execution. For this reason, the parallelization is done on the example of a multi-population. The procedure, however, is just the same for basic population. Simply provide the population class to be parallelized as a template parameter. In addition to this the parallel personality needs to know about the type of the individuals that are stored as the lowest level of our optimization framework. In our example this is the `parabola` class. Again a template parameter is used to tell the parallel population personality about it. Listing A.21 shows how to parallelism a population using the POSIX thread personality. As **the rest of the main function remains unchanged** compared to listing A.20, only the **one relevant line** is shown. It can be seen that, from the users perspective, setting up a parallel population is almost trivial.

A.3.9 Using different parallel personalities

Apart from the POSIX thread personality `evaPthreadPopulation<S,T>` there is also a version that makes use of the MPI library for parallelization. It relies heavily on the generation and parsing of XML representations of classes derived from `evaMember`. Setup of the MPI personality is very similar to the POSIX thread personality, so it will not be explained in detail here. From a users' perspective, the only real difference between `evaPthreadPopulation<S,T>` and `evaMPIPopulation<S,T>` is that the latter needs to receive the `main()` functions command line parameters `int argc` and `char **argv`, as the MPI implementation MPICH relies on them. Usage of MPI programs is explained e.g. in the MPICH manual [MPICH].

Instead of going into detail about the `evaMPIPopulation` class, a `main()` function (listing A.22) and a `Makefile` (listing A.23) are shown that allow to switch between the serial, multi-threaded and MPI personality simply by uncommenting the appropriate lines in the

Listing A.22: *Switching between parallel personalities in* `main()`

```
[...]
#ifdef USEPTHREAD
#include "evapthreadpopulation.hh"
#endif

#ifdef USEMPI
#include "evampipopulation.hh"
#endif

int main(int argc, char **argv)
{
  int i,j;

  #ifdef USESERIAL
  evaPopulation<evaPopulation<parabola> > superpar;
  #endif

  #ifdef USEPTHREAD
  evaPthreadPopulation<parabola, evaPopulation<evaPopulation<parabola> > > superpar;
  #endif

  #ifdef USEMPI
  evaMPIPopulation<parabola,
        evaPopulation<evaPopulation<parabola> > > superpar(argc,argv);
  #endif

  // [...] The rest of the code remains unchanged, so we do not show it here.
}
```

`Makefile`. Again, the `main()` function is unchanged compared to listing A.20, so only those lines are shown that have been changed.

As is obvious from the `Makefile` shown in listing A.23, if the lines associated with the POSIX thread personality are uncommented, the `-DUSEPTHREAD` definition is activated and the `evaPthreadPopulation<S,T>` section of the `main()` function becomes active. The serial and MPI personalities are handled similarly.

A.3.10 Restrictions of the parallelization

There are only relatively few restrictions to the individuals being stored in a parallelized population. First, they must be independent of each other, i.e., during the course of the value calculation and mutation there may be no exchange of data between them. This is usually no restriction, as value calculation and mutation are atomic actions in most cases. However, global parameters must be avoided. Where there is an exchange of data between individuals or global variables are being used, a user has to take care of synchronization of the `customValue()` and `customMutate()` functions himself. However, again, this is not a restriction for most applications.

Please note, that this is also no restriction for (user- or pre-defined) recombination and duplication schemes, which of course require "interaction" between individuals, as this

Listing A.23: *Switching between parallel personalities (Makefile)*

```
TOPDIR=$(PWD)
INCDIR=/opt/eva/include

CPP = g++

INCLUDE = -I$(INCDIR) -I$(PWD)

# Use these settings for sequential execution
# CPPFLAGS = -O2 -Wall -fexceptions -DUSESERIAL
# LIBS = -lc -lm -lxml2 -lz -leva
# LIBFLAGS=-L/opt/eva/lib

# Use these settings for the MPI personality
# CPPFLAGS = -O2 -Wall -fexceptions -I/where/mpich/resides/include -DUSEMPI
# LIBS = -lc -lm -lxml2 -lz -leva -lmpich
# LIBFLAGS=-L/opt/eva/lib -L/where/mpich/resides/lib

# Use these settings for the pthread personality
CPPFLAGS = -O2 -Wall -fexceptions -DUSEPTHREAD
LIBS = -lc -lpthread -lm -lxml2 -lz -leva
LIBFLAGS=-L/opt/eva/lib

FILES = $(TOPDIR)/main.o \
        $(TOPDIR)/parabola.o

.cc.o:
        $(CPP) $(CPPFLAGS) $(INCLUDE) -c $<

parabola: $(FILES)
        $(CPP) $(FILES) $(LIBFLAGS) $(LIBS) -o parabola

clean:
        rm -f *.o *~ parabola
```

functionality is handled by the server process or thread rather than the clients.

The XML code equivalent to the super population `superpar` from listing A.20 has of course grown in size, to over 1.7 megabytes. While this is no real restriction, users that want to use super-populations as sub-populations of a "super-super" population should be aware of the size of the code that is generated. One potential problem is the size of the receive buffer in an MPI population, which is predefined to be 1 megabyte. So in the case of a super-super population similar to our example you would have to modify the size of the buffer to benefit from parallelization. You can change the size of the buffer by using the define switch `-DBUFSIZE=<buffersize in bytes>` during compilation. This can be done by modifying the `CPPFLAGS` section of listing A.23.

A.4 Consistency Checks

A few of the consistency checks used during the development of the EVA library will be shown below.

A.4.1 Random Numbers

Evolutionary Algorithms heavily depend on the generation of random numbers. The EVA library currently uses the *Mersenne Twister* algorithm, that was implemented in the `MTRand` class by Richard Wagner, based on code by Makoto Matsumoto, Takuji Nishimura, and Shawn Cokus [MAT98]. `MTRand`'s period, $2^{19937} - 1$, and its order of equidistribution, 623 dimensions, are far greater than is common for other random number generators.

The `MTRand` class is distributed under the *GNU Lesser General Public License* (see [LGPL]) and can thus be used freely in the EVA library[11].

EVA implements random numbers with a gaussian probability density function (PDF)[12] on top of `MTRand`, using an algorithm described in detail in section A.4.2 below.

In order to show their characteristics, gaussian random numbers with two different sets of mean and sigma (0./0.5 and 6/2.) were filled into a *ROOT* histogram. The function

$$f(\vec{p}, x) = p_0 * e^{-0.5*((x-p_1)/p_2)^2} + p_3 * e^{-0.5*((x-p_4)/p_5)^2} \tag{A.1}$$

was fitted to the random distribution using the means provided by the `ROOT` object oriented analysis framework (see [ROOT] for further information). Picture A.2 shows, that already for a relatively low statistics the original values of $(mean/sigma)$ are reproduced by the fit reasonably well.

[11] Future versions of MTRand are likely to be distributed under an even more open license, imposing almost no restrictions on users at all

[12] Henceforth these will be called "gaussian random numbers"

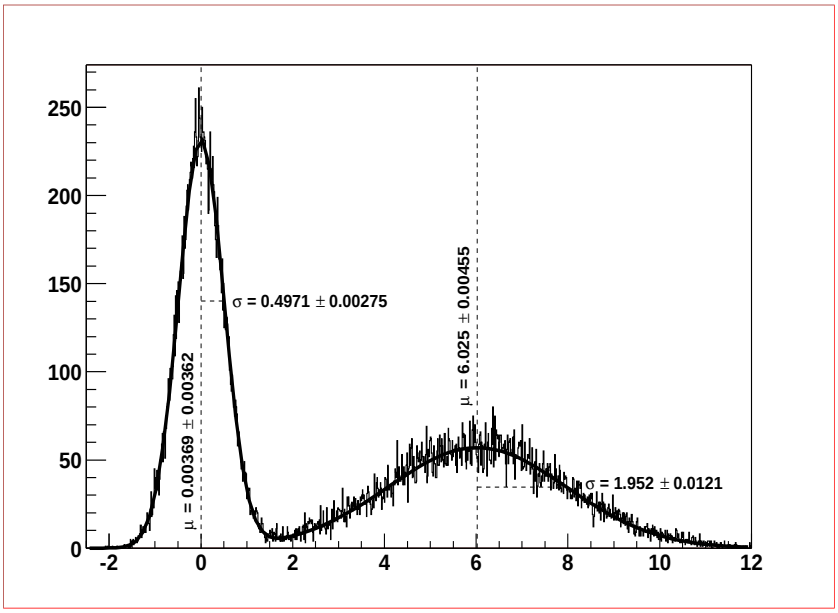

Figure A.2: *Function A.1 was fitted to gaussian random numbers with two different sets of mean/sigma (0./0.5 and 6./2.). Their values, as reproduced by the fit, are shown in the picture.*

A.4.2 Generating random numbers with a gaussian Probability Density Funtion (PDF)

A crucial point in Evolutionary Algorithms is the generation of random numbers. Evolutionary Strategies, in particular, require random numbers with a gaussian PDF, such as the ones shown in picture A.2. They can be obtained from random numbers with a uniform distribution on $[0, 1[$ using the following procedure:

One can obtain random numbers with a known distribution $f(x)$ by calculating the inverse of the function

$$F(x) = \int_{-\infty}^{x} f(x')dx' \tag{A.2}$$

The application of $F^{-1}(x)$ to random numbers with a uniform distribution on $[0, 1[$ yields random numbers distributed according to $f(x)$. Unfortunately, for the gaussian distribution, $F(x)$ cannot be calculated analytically.

A little "trick" helps to find a way out of this dilemma. Let's assume that a set of random numbers with a gaussian PDF already exists. It will have the density

$$f(x)dx = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx \tag{A.3}$$

This distribution is now "smeared" along the y-coordinate with a second gaussian distribution, resulting in the 2-dimensional density

$$g(x, y) = \frac{1}{2\pi} e^{-\frac{x^2+y^2}{2}} dxdy \tag{A.4}$$

or, in polar coordinates

$$g(r, \Phi) = \frac{1}{2\pi} r e^{-\frac{r^2}{2}} drd\Phi \tag{A.5}$$

A two dimensional random vector can be obtained using the assumption that Φ is distributed uniformly on $[0, 2\pi[$, while r is distributed according to

$$h(r)dr = re^{-\frac{r^2}{2}}dr \tag{A.6}$$

in $[0, \infty[$. In contrast to equation A.3, however, it is possible to calculate function A.2 for $h(r)$:

$$H(r) = \int_0^r h(r')dr' = 1 - e^{-\frac{r^2}{2}} \tag{A.7}$$

with the inverse

$$H^{-1}(r) = \sqrt{-2ln(1-x)} \tag{A.8}$$

Two independent, gaussian random numbers can then be obtained with the help of two random numbers X_1 and X_2 with a uniform distribution on $[0, 1[$ by projecting equation A.8 onto the x- and y-axis.

$$Y_1 = \sqrt{-2ln(1-X_1)}\ cos(2\pi X_2) \quad \text{and} \quad Y_2 = \sqrt{-2ln(1-X_1)}\ sin(2\pi X_2) \tag{A.9}$$

Gaussian random numbers with any desired mean μ and width σ can then be obtained by stretching and scaling equations A.9 :

$$Y = \sigma\sqrt{-2ln(1-X_1)}sin(2\pi X_2) + \mu \tag{A.10}$$

A.4.3 External vs. Internal representation of an evaDouble

As was described in section 4.4.2, the EVA library's `evaDouble` class implements the transition from the internal to an external representation of a double value, which can optionally be restricted to a certain value range and have gaps in this range. The following examples try to reproduce the transition described in picture 4.2.

With boundaries

In the first example a lower boundary at -0.5 and an upper boundary at 1.5 was added to the value range of an `evaDouble`. Picture A.3 shows the external representation of an `evaDouble`, as a function of the internal, continuous, representation.

Adding gaps

In the next step, two gaps at $[-0.1 : 0.3]$ and $[0.7 : 0.9]$ where added to the (external) value range of picture A.3. Picture A.4 shows the resulting mapping from internal to external representation.

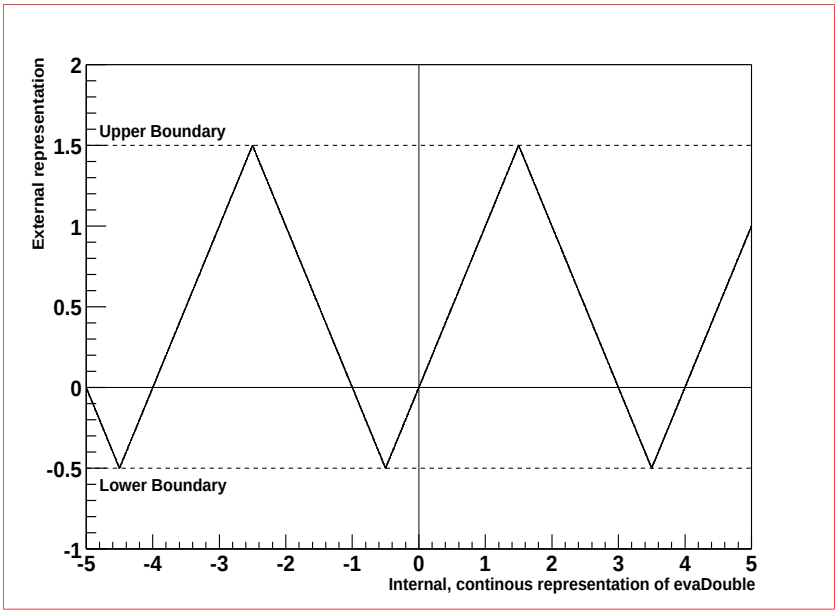

Figure A.3: *The internal, continuous representation of an* `evaDouble` *is mapped to the external representation, whose value range is limited to* $[-0.5 : 1.5]$.

A.4.4 Training Feed-forward Neural Networks

An implementation of Feed-forward Neural Networks (FFNN) was created as an extension to the EVA library. Each layer of a network is implemented using an `evaIndividual<evaDouble>`, so a network can be represented as an `evaIndividual<evaIndividual<evaDouble> >`.

Training a FFNN means adapting the weights associated to each node in such a way that, during supervised learning, for a given input sample the deviation of the actual from the desired output becomes minimal. In other words, training a FFNN means minimizing function A.11,

$$E(\vec{w}) = \frac{1}{2} \sum_{\nu=1}^{p} \sum_{k} (y_k^\nu - s_k(x^\nu))^2 \tag{A.11}$$

where $\vec{w}$ are the weights, p is the total number of input/output patterns used in supervised training, k are the output nodes, y_k^ν is the actual output of each output node as a result of input pattern x^ν, and $s_k(x^\nu)$ is the desired output associated to x^ν. As searching for minima and maxima of high-dimensional functions is the prime domain of Evolutionary Strategies, they can also be used to train FFNNs. See also [RBDP95] and [RBNN95] for further information on training feed-forward neural networks using Evolutionary Strategies.

Both FFNN's with sigmoidal and radial-basis transfer functions have been implemented within this library (`libFfn`). Training patterns and network geometries can be loaded using a simple XML based format. When the training is finished, the trained network is stored as a C++ function, which can be easily used in custom projects. Using the means of the EVA library, training can also be done in parallel. Picture A.5 shows the resulting

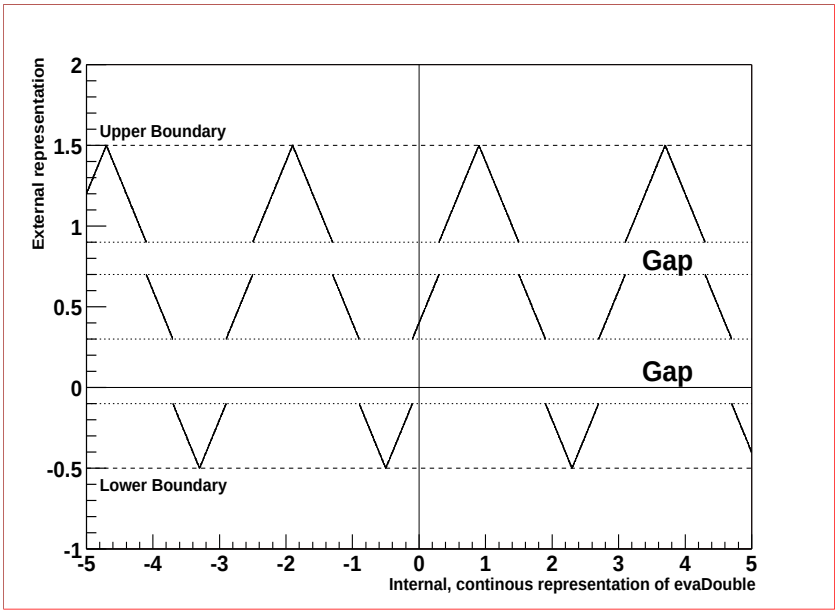

Figure A.4: *Two gaps were added to the value range of picture A.3. Gaps are reflected as a different transfer function mappingthe internal to the external representation of an* `evaDouble`.

class hierarchy.

As `libFfn` is entirely based on the EVA library, trying to use a network trained with this algorithm to recognize a training pattern is a thorough test of the EVA library. In the following example, `libFfn` had to distinguish between two two-dimensional sets of input patterns. One was distributed along the x- and y-axis, the other was distributed evenly in $[0:1]/[0:1]$.

Picture A.6 shows the first set and, plotted on top of it, the output of the trained "$2-2-1$" network[13] for selected points of the input region. It is clearly visible that the network is able to distinguish between both patterns.

A.5 Interfacing the EVA library to ROOT and PAW

Today the most common analysis frameworks in particle physics research are ROOT and PAW, with PAW slowly phasing out. Particle physicists will usually extract only those data portions from the raw events coming from the experiments that are of direct importance to their analysis. It is a common procedure to store this data in PAW or ROOT ntuples or trees[14]. This decreases the amount of data to be analyzed from hundreds of terabytes to several gigabytes and is indeed the only reason why the vast amount of data coming from particle physics experiments is manageable. After all it is usually necessary to loop more

[13] A network with 2 input-, 2 hidden and 1 output node. The output is scaled by a factor of 10 and rounded to the next lower integer.

[14] In ROOT terminology a tree is a collection of datasets, each of them assembled from the same amount of basic types and higher-level objects. An ntuple is a tree that can only hold floating point values

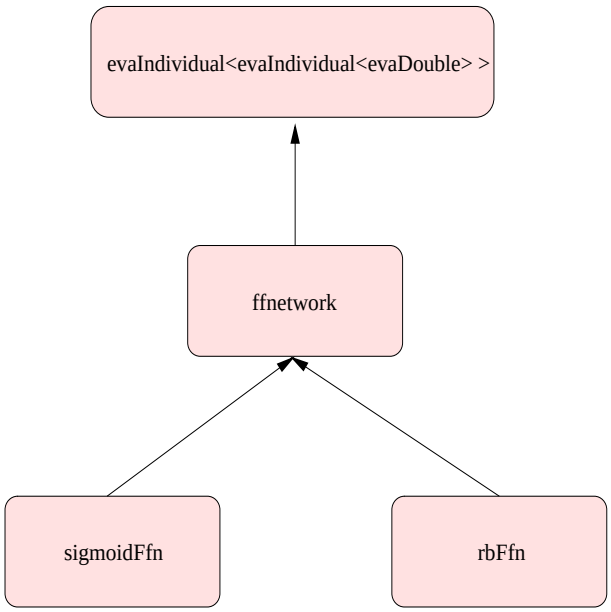

Figure A.5: *The class hierarchy used in* `libFfn`. *The library is built on top of the EVA framework.*

than once over a dataset to perform an analysis.

In order to facilitate the deployment of the EVA library in Particle Physics, it was a key requirement to allow the easy access to the analysis means provided by ROOT and Paw. In the case of the ROOT framework, two preferred possibilities exist :

- The first and easiest method of interaction is through ROOT scripts. A special class `evaRootIndividual` has been developed that integrates a ROOT interpreter into the EVA framework. The population class `evaRootPopulation` takes care that the ROOT interpreter is initialized properly. It is also able to set up the entire optimization scheme using information stored in a simple, XML-based configuration file. Listing A.24 shows such a configuration file. Apart from setting up the "geometry" of the population, it is possible to specify parameters for the parent individuals, including values and boundaries for each `evaDouble`.

  Users write scripts for this interpreter and test it in their native analysis environment. These scripts can then be used almost verbatim in conjunction with the `evaRootIndividual` object. The name of the script can be specified in the XML configuration file. Listing A.25 shows a simple evaluation script. The interpreter is called from the `customValue()` function of `evaRootIndividual`. The ROOT script gets access to the `evaRootIndividual`'s parameters and returns a single `double` value representing the quality or fitness of this parameter set.

  Listing A.26 shows a simplified `main()` function that can be used to set up an `evaRootPopulation`.

  There are two disadvantages to this method :

  – It can only be used in conjunction with the `evaMPIPopulation` and

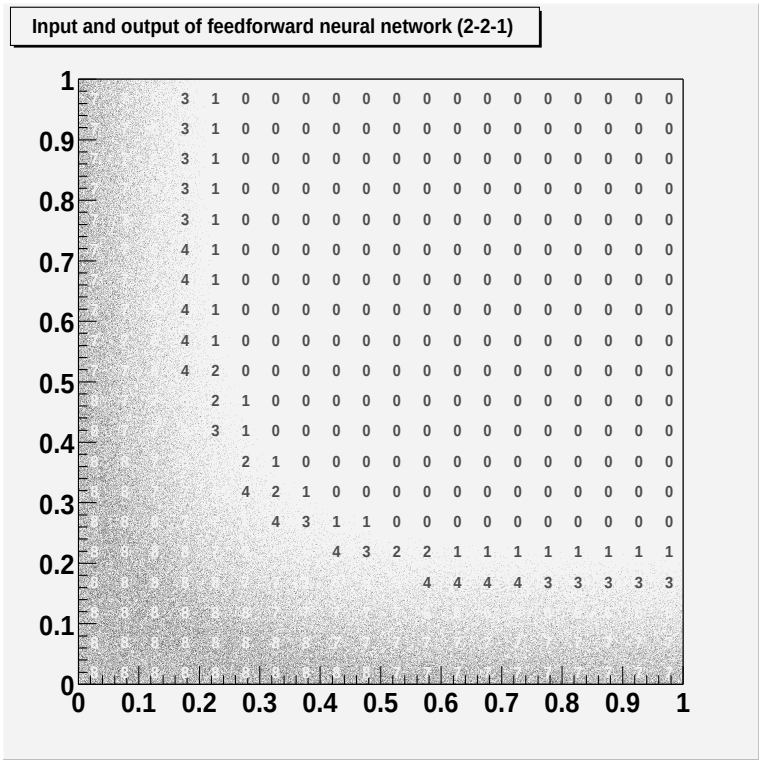

Figure A.6: *Input pattern used to train a 2-2-1 network (an even distribution of input values and a distribution confined to the x- and y-axis) and the output of the network after the training (numbers).*

`evaPopulation`. Usage in multi-threaded environments is not possible due to the usage of global variables in the ROOT interpreter.

– The ROOT interpreter tries to load shared libraries. These must comply with the environment the EVA application was compiled in, as otherwise the symboltable of the executable doesn't fit the dynamic libraries, resulting in errormessages from the ROOT interpreter. Consequently one should run an evaRootPopulation on a cluster with a similar version of Linux/Unix as the one the application was compiled on.

There is only a small performance penalty in using ROOT scripts instead of compiled functions, as ROOT has the ability to compile ROOT scripts on the fly when they are first loaded[15].

- ROOT commands are also accessible as part of ROOT's library of C++ classes and can be integrated as function calls directly into an application without the need to interface with the interpreter. This method has the advantage to yield the fastest execution, but lacks flexibility, as changes in the code require recompilation of the application.

Interaction with PAW is done by calling the PAW interpreter in batch mode using the `C` call `system("paw -b samplescript.kumac")` and Kumac scripts. A similar method

[15]Simply add a "+" after the name of the script in listing A.24.

Listing A.24: *Setting up an* `evaRootPopulation` *with an XML input file*

```
<rootPopulation nPop="10" nPar="1" maxGen="40" saveTurn="1" maximize="true">
   <rootIndividual nVar="4" rootScript="phipi.cc">
     <evaDouble val="1.3" name="pcmscut">
     <lower val="1." isOpen="false" isActive="true"/>
     <upper val="2." isOpen="false" isActive="true"/>
     <sigma val="0.052"/>
     <sigmasigma val="0.0052"/>
     </evaDouble>

     [... additional evaDouble's erased ...]

   </rootIndividual>
</rootPopulation>
```

Listing A.25: *A simple root script*

```
  void phipi(double *values, int nValues, double *result)
  {
    *result=0;
    for(int i=0; i<nValues; i++){
       // do some calculation
       *result += values[i]*values[i];
    }
  }
```

would also work for the ROOT interpreter

A.6 Copyright

The EVA library is distributed under the terms of the *GNU General Public License* (GPL). A copy of this license can be found at [GPL]. The GPL ensures that the EVA library can be freely copied and modified, as long as modifications of the library are themselves placed under the GPL. This approach, commonly referred to as Open Source licensing, is in line with the intention to make the EVA library a leading implementation of Evolutionary Algorithms. There are numerous examples in the particle physics community where Open Source software has proven to be superior both in terms of quality and extensibility to proprietary software. Due to the fact that Open Source software is freely available and its sources are available to any interested party, bugs are discovered and fixed more quickly and suggestions for improvements can be included into the software without delay.

Listing A.26: *The* `main()` *function used in conjunction with the evaRootPopulation*

```
#include "evarootindividual.hh"
#include "evarootpopulation.hh"

#ifdef USEPTHREAD
#include "evapthreadpopulation.hh"
#endif

#ifdef USEMPI
#include "evampipopulation.hh"
#endif

using namespace std;

int main(int argc, char **argv)
{
  // Need to prevent POSIX threads from being used
  #ifdef USEPTHREAD
  cout << "Error : Attempt to use POSIX threads !" << endl;
  exit(1);
  #endif

  #ifdef USEMPI
  evaMPIPopulation<evaRootIndividual, evaRootPopulation> rpop(argc,argv);
  rpop.setSendPolicy(DISTRIBUTEWILLING);
  #endif

  #ifdef USESERIAL
  evaRootPopulation rpop;
  #endif

  // Check that we have a sufficient number of arguments
  if(argc < 2){
    cout << "Error : bad number of arguments : " << argc << endl;
    exit(1);
  }

  // load XML description of population
  xmlDocPtr doc;
  if((doc=xmlParseFile(argv[1]))){
    rpop.loadSimpleXML(doc->children);
    cout << "Loaded XML description" << endl;
  }
  else{
    cout << "Error : could not load XML description " << argv[1] << endl;
    exit(1);
  }

  rpop.optimize();

  exit(SUCCESS);
}
```

Appendix B

Using Neural Networks to Optimize Significance

Feed-forward neural networks can be used to find an ideal cut separating two or more event classes. Section A.4.4 gives an example, where a feed-forward network was trained using the EVA library[1] to distinguish between two event classes, the first distributed along the x- and y- axis, the second distributed evenly over the whole range of the training data. As the value range of both data sets overlaps and the density of the first data set is not constant over the whole area, a non-linear cut had to be found, separating the two datasets with a minimum of mis-identification of input vectors. As shown in figure A.6, the smallest useful network featuring two input nodes for the x- and y- values, two nodes in the hidden layer and one output node is already able to perform a good separation of both event classes[2].

This remarkable ability of feed-forward networks has in the past also been used successfully in scientific analyses in particle physics. Due to a neural networks ability to exploit possible correlations between input parameters it is often possible to achieve better results than when using orthogonal cuts.

The biggest drawback of feed-forward neural networks, however, is at the same time one of the biggest advantages of parametric optimization methods based on Evolutionary Strategies as presented in chapter 5. In supervised training a neural network is given examples of the event classes to be distinguished as input vectors, and the weight set of the network is adapted in a way that the output nodes of the network return the desired value. A clear knowledge of the characteristics of the input data is thus required during the training phase, and in particle physics this can usually only be done using Monte Carlo events. Optimization of significance based on Evolutionary Strategies, on the other hand, requires but little knowledge about the underlying data set. It can indeed be necessary to cal-

[1] As opposed to the standard Back-propagation algorithm that is usually used to train feed-forward networks. Please note that a network trained using the EVA library is in no way different from a network trained using the standard algorithm

[2] Please note that due to the overlap it is impossible to completely separate both event classes

culate the background from Monte Carlo data. Combinatorial background, however, can usually be simulated with high accuracy, as no clear knowledge of underlying physical decay processes[3] is necessary. A big drawback of parametric optimization with Evolutionary Strategies, however, is that this method doesn't exploit correlations between cut variables.

No clear advice can be given where Neural Networks will perform better than the algorithm presented in chapter 5. The choice, however, will clearly depend on the question, how big a systematic error one will incur by having to use Monte Carlo events for training. As particle decays that can only be reconstructed with relatively low statistics are expected to be more vulnerable to the creation of fake peaks (see section 5.2.4) during the optimization of significance using Evolutionary Strategies, it is also important to know how big the data set of reconstructed events is that is used for the optimization.

Unsupervised Training

It appears possible to combine the advantages of feed-forward neural networks with Evolutionary Strategies. The problem of having to use a set of training data with known characteristics could be avoided by using the significance resulting from a given weight set of a network as its figure of merit. This unsupervised training technique would then be able to exploit correlations of input variables while not having to rely on Monte Carlo data for training.

Training of such a network could no longer be done using the back-propagation algorithm as it relies on partial derivatives of the network's error function. Training is however possible using Evolutionary Strategies, as demonstrated in section A.4.4 and also discussed in [RBNN95].

The approach has been tested as part of this thesis. While it still appears to be a workable method, it has turned out that the biggest challenge lies in the determination of good starting values for the weight set. A very large number of training cycles is therefore necessary to reach a satisfactory result. Please also keep in mind that the determination of the quality of each weight set requires processing large amounts of data. So this approach seems to be problematic.

[3] such as lifetime and decay channels

Bibliography

[AMOEBA] The Amoeba home page
http://www.cs.vu.nl/pub/amoeba/amoeba.html

[AUBERT1] B. Aubert et al., *Phys. Rev. Lett.* **86**(2001)2515-2522

[AUBERT2] B. Aubert et al., Observation of a Narrow Meson Decaying to $D_s^+\pi^0$ at a Mass of 2.32 GeV/c^2
BABAR–PUB–03/011; SLAC–PUB–9711

[BABARCP] CP violation "primer" (*BABAR*)
http://www.slac.stanford.edu/pubs/confproc/babar504/babar504-001.html

[BEOWULF] The Beowulf home page
http://www.beowulf.org

[BOCK98] http://rd11.web.cern.ch/RD11/rkb/PH14pp/node39.html
A description of Dalitz plots

[BRANDT92] Datenanalyse
Siegmund Brandt
ISBN 3-411-03200-6

[CACTUS] Homepage of the *Cactus* Grid middleware
http://www.cactuscode.org/

[CARL96] http://computer.org/computer/timeline/
A timeline of computing history

[DAL53] R.H. Dalitz, On the analysis of τ-meson data and the nature of the τ-meson, Phil. Mag. 44 (1953) 1068

[DIRC] I. Adam *et al.*, Nucl. Instr. and Methods A **453** 301 (2000); I. Adam *et al.*, Nucl. Phys. Proc. Suppl. **93** 340 (2001).

[EAFAQ] http://www.faqs.org/faqs/ai-faq/genetic/part1
The Hitch-Hiker's Guide to Evolutionary Computation

[EDG] Homepage of the European Data Grid initiative
http://eu-datagrid.web.cern.ch/eu-datagrid/

[EGEE] Homepage of the EGEE initiative, the successor to the European Datagrid [EDG]: "Enabling Grids for E-science and industry in Europe"
http://egee-ei.web.cern.ch/egee-ei/2003/index.html

[FOST99] Ian Foster and Karl Kesselman
The Grid - Blueprint for a New Computing Infrastructure
Morgan Kaufmann, ISBN 1-55860-475-8

[GANJ02] Sergey Ganjour
Study of $B \to D_s^{(*)+} X$ and $B \to D_s^{(*)+} D^{*-}$ Decays with the *BABAR* Decector
Dissertation, Ruhr-Universität Bochum, March 2002

[GLOBUS] Homepage of *Globus* middleware
http://www.globus.org

[GOETZ03] Klaus Götzen
Untersuchung des Zerfalls $D_s^{\pm} \to K^+K^-\pi^+\pi^-\pi^{\pm}$
Dissertation, Lehrstuhl für Experimentalphysik I, Ruhr-Universität Bochum
Bochum, August 2003

[GPL] The Gnu General Public License
http://www.gnu.org/licenses/gpl.html

[GRIDKA] Homepage of the *GridKa* Grid Computing Centre at Forschungszentrum Karlsruhe
http://grid.fzk.de/

[HOLL75] John Holland
Adaptation in Natural and Artificial Systems
MIT Press, ISBN 0262581116 (reprint)

[IMPA98] John Impagliazzo et. al.
History in the Computing Curriculum

[KOPF02] Bertram Kopf
Untersuchung der photoinduzierten Reaktionen $\gamma p \to p\pi^0\pi^0$ und $\gamma p \to p\pi^0\eta$ an einem Flüssig-Wasserstoff-Target
Dissertation zur Erlangung des Doktorgrades der Naturwissenschaften
Dresden, Mai 2002

[KOPF03] Private communication with Bertram Kopf about the feasibility of using Evolutionary Strategies in a Dalitz plot analyses.

[LCG] Homepage of the LHC Computing Grid
http://lcg.web.cern.ch/LCG/

[LEGION] Homepage of the *Legion* middleware
http://citeseer.nj.nec.com/459307.html

[LGPL] The Gnu Lesser General Public License
http://www.gnu.org/licenses/lgpl.html

[LHCBCP] Explanation of CP violation on the LHC-b web page
http://lhcb-public.web.cern.ch/lhcb-public/html/introduction.htm

[LUED95] Jörg Lüdemann
Beobachtung von Resonanzen in der Proton-Antiproton-Annihiliation im Fluge in drei pseudoskalare Mesonen
Dissertation im Fachbereich Experimentalphysik I der Ruhr-Universität Bochum
Oktober 1995

[MAT98] M. Matsumoto and T. Nishimura
Mersenne Twister: A 623-Dimensionally Equidistributed
Uniform Pseudo-Random Number Generator
ACM Transactions on Modeling and Computer Simulation
Vol. 8, No. 1, January 1998, pp 3-30.

[MFM] Simulation and measurement of the fringe field of the 1.5 T *BABAR* solenoid, E. Antokhin *et al.*, Nucl. Instr. and Methods A **432**, 24 (1999)

[MOSIX] The Mosix home page
http://www.mosix.org

[MPICH] The MPICH home page
http://www-unix.mcs.anl.gov/mpi/mpich/

[MPICH-G2] The MPICH-G2 home page
http://www.hpclab.niu.edu/mpi/

[PACX] The PACX home page
http://www.hlrs.de/organization/pds/projects/pacx-mpi/

[PELI02] Marc Pelizäus
Nachweis und Untersuchung der Reaktion $D_s^{\pm} \to \pi^+\pi^-\pi^+\pi^-\pi^{\pm}$
Diplomarbeit, Lehrstuhl für Experimentalphysik I, Ruhr-Universität Bochum
April 2002

[PVM] The PVM home page
http://www.csm.ornl.gov/pvm/pvm_home.html

[PF96] Physical Review D, Particles and Fields, Part I
The American Physical Society
American Institute of Physics
ISSN 0556-2821

[RBNN95] Rüdiger Berlich
Training Feedforward Neural Networks using Evolutionary Strategies
Proceedings of AIHENP95, pp. 521ff.
World Scientific, ISBN 9810224362

[RBDP95] Rüdiger Berlich
Visualisierung hadronischer Splitoffs und ihre Erkennung
mit neuronalen Netzen
Diploma Thesis, Physics
Ruhr-Universität Bochum, 1995

[RBLU01] Roots of a Revolution - Linux history
Rüdiger Berlich
Linux User UK Magazine, March and April 2001

[RECH73] Ingo Rechenberg
Evolutionsstrategie
Friedrich Frommann Verlag, Stuttgart 1973

[RECH94] Ingo Rechenberg
Evolutionsstrategie '94
Frommann-Holzboog Verlag, ISBN 3-7728-1642-8

[ROB98] Studying CP Violation with the *BABAR* Detector at SLAC
http://hep.ucsb.edu/people/roberts/Colloqium/sld001.htm

[ROOT] The *ROOT* Object Oriented Analysis Framework
http://root.cern.ch .

[SCHO93] Eberhard Schöneburg, Frank Heinzmann, Sven Feddersen
Genetische Algorithmen und Evolutionsstrategien
Addison Wesley, ISBN 3-89319-493-2

[SPAN98] High-statics Dalitz plots created by members of the Crystal Barrel collaboration
http://www.phys.cmu.edu/cb/dalitz_plot.html

[SUNGRID] Homepage of the SUN Grid engine
http://gridengine.sunsource.net/

[SVT] The *BABAR* Silicon Vertex Tracker, C. Bozzi *et al.*, Nucl. Instr. and Methods A **435**, 25 (1999)

[TAN03] Andrew S. Tanenbaum, Marten van Steen
Verteilte Systeme - Grundlagen und Paradigmen
Prentice Hall, ISBN 3-8273-7057-4

[UNICORE] Homepage of the *Unicore* Grid inter-connection framework
http://www.unicore.org/

List of Figures

Curriculum Vitae

Persönliche Daten

Name	Rüdiger Berlich
Geboren	Am 21.01.1968 in Wuppertal
Familienverh.	ledig
Staatsangeh.	deutsch
Adresse	Friedrich-Naumann Str. 109 76187 Karlsruhe
E Mail	ruediger@berlich.de

Berufserfahrung

seit 8/2001	Doktorarbeit an der Ruhr-Universität Bochum
8/1999–7/2001	SuSE Linux Ltd, London; Aufbau und Leitung der britischen Tochterfirma der SuSE Linux AG als Managing Director
2/1999–7/1999	SuSE Inc., Oakland/USA; Aufbau und Leitung einer Supportabteilung für englischsprachige Kunden als Technical Manager Support
2/1998–1/1999	SuSE Linux GmbH, Nürnberg
5/1995–1/1998	Max-Planck-Institut für Hochenergiephysik, München; Mitarbeit am Aleph Experiment am LEP (CERN/Genf)

Studium

10/1987–4/1995	Studium der Physik an der Ruhr-Universität Bochum
	Diplomarbeit zum Thema “Visualisierung hadronischer Splitoffs und ihre Erkennung mit Neuronalen Netzwerken”. Abschluss von Vordiplom und Diplom mit der Note “sehr gut”.

Schulbildung

1974–1987	Grundschule und Gymnasium in Wuppertal
	Leistungskurse Englisch und Französisch
	Abschluss Abitur

Thank You !

I would like to take the opportunity to say a big “Thank you” to all who have contributed to the success of this work. While writing this text I’ve found that there are more peope that I like than that I dislike – which, I guess, is fortunate. It wasn’t easy to select the people I want to mention here – I hope I have succeeded in listing the right ones.

First and foremost, I would like to thank the two people who have initiated this thesis - Prof. Dr. H. Koch and Dr. Marcel Kunze. Their big interest in the ongoing work, their advice and patience were key to its success ! Similarly, Dr. Klaus Peters, Dr. Matthias Steinke, Dr. Bertram Kopf, Dr. Bernd Lewandowsky and Dr. Andrea Wilms were always available for my questions, I’d like to sincerely thank them. All members of EP1 have in one way or another contributed to this work. Thanks to y’all, mates !

The colleagues at Forschungszentrum Karlsruhe have provided me with an exciting work environment. I’d especially like to thank Klaus-Peter Mickel, Dr. Holger Marten and all members of GES and GIS for the opportunity to participate in the exciting field of Grid computing.

Marcus Hardt and Jos van Wezel: thanks for the whisky. You do know that my first large C++ program was named Jhonny Walker, don’t you ?

A very big “Thank You” goes to Ms. Ursula Epting. She is a constant source of inspiration and is always willing to issue another Globus certificate. Have fun ! :-)

Also, Andrea Ballouk and Annette von Czarnowski, despite sometimes being far away, will always remain close to my heart.

Ms. Edith Borie has given me a home away from home.

Horst, who runs the Katzenstube cafe in Bochum, has become a friend over the years.

A big “hi” goes across to the UK to all my former colleagues at SuSE Linux Ltd., but especially to Jasmin Ul-Haque, Sally Donohoe, Roger Whittaker, Justin Davies and Khalid Sanad.

I would like to thank Inge and Jens Hanten, who are part of the family.

Next, I want to thank my brother Peter Berlich for constant support and more SMS than I can count. Keep in mind : “Mut zum Komma” ... :-)

Finally, I would like to dedicate this work to my mother Hanna Berlich, who died in 1982, to my father Karl Berlich, and to my aunt Hildegard Heinzerling.