INTERSHIP REPORT

<u>EUROPEAN SYNCHROTRON RADIATION FACILITY</u>

Christian Böddicker

Report of my intership at the ESRF in Grenoble

Period: 01.03.94 - 31.08.94
Subject: Global Optimization of Optical Systems

CONTENTS

I. THE SYNCHROTRON

I.1. Some Smalltalk ...

In the middle of France, surrounded by the mighty mountain-ranges of the Vercores, the Chartreuse and the Belledonne and flanked by the two large streams Isere and Drac lies the European Synchrotron Radiation Facility (ESRF), about 2 km away from the city centre of Grenoble.
The ESRF is one of the newest and most modern synchrotron radiation facilities in the world. Its main construction (starting in the year 1988) lasted about six years and the first beam was injected in 1992. The first beam lines (the places where the actual work is done) are now starting their work. In 1998, 30 beam lines are expected to operate, maintained and financed by different countries and companies all over Europe.
The ESRF is a cooperation of twelve european countries which sponsoring the whole project (2.6 billion French Francs just for the construction and about 1.0 billion Francs for the first four years of operation).
The facility employs about 430 scientists and workers and 2000 visiting scientists are expected per year, to work at the different beam lines. Many researchers in physics, chemistry, biology, engineering, medical science and others find in ‘Synchrotron Radiation’ (see below) an essential tool for their investigation and research. Even commercial applications are foreseen for future days. The ESRF was built as a ‘facility’, meaning that researchers from all over the world may come and perform their experiments on the different beam lines.
The Experiments Programming Group, for which I worked during my stay in France, is responsible to support the scientists on the different beamlines. So the scientists needn’t bother about computer problems or spend their time with programming of software, but the 15 members of the EXPG take care for that. This division of tasks makes live much easier for the single scientist, because mostly they are not very keen in programming and haven’t the time to spend their time to learn a programming language instead of working on their experiment.

I.2. What is a synchrotron ?

The first observation of synchrotron radiation was done in 1947 in the USA. Since, a lot of synchrotrons were built around the world (The largest and probably most famous is the LEP synchrotron at CERN in Geneva). All synchrotrons following the same principles: Some particles are accelerated in a ring near to the speed of light. Controlled and lead by strong magnets they are forced on a circular path. When they reach this incredible speed, the particles emit different kinds of radiation, which can be used for experiments.
For many years, the ‘synchrotron radiation’ was considered a ‘technical nuisance’ by the first high energy physicists, because it hinders acceleration of particles to very high energies (emitted radiation means always a loss of energy). It was then in the 50’s and 60’s when the first pioneering experiments were done to demonstrate the practical use of this radiation.
The first generation of SR sources were just parasites who used the waste products of the big particle accelerators for their own purposses.Yet modern X-ray synchrotrons (we are now in the third generation) today are built for the purpose only to produce x-ray radiation and all technical

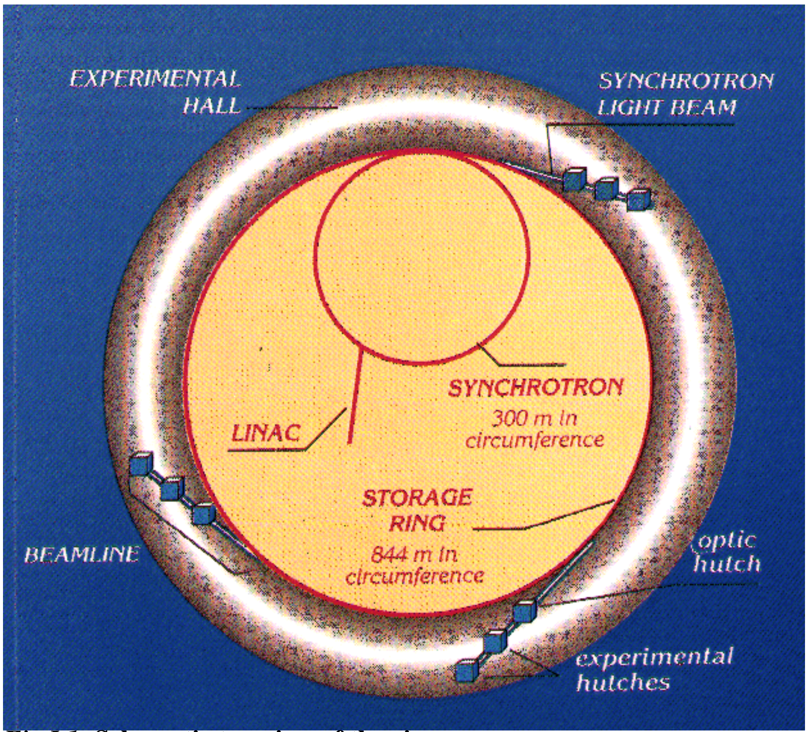


Fig I.1: **Schematic top view of the ring**

knowledge is used to enhance the quality and amount of the radiation output.
The ESRF is the most intensive x-ray facility in the world and is built exclusive to produce this kind of radiation. Here is the principle how the ESRF synchrotron is working:

The ESRF radiation source consist of a storage ring for electrons with a circumference of 850 m. The electrons, grouped in bunches, circulate in the evacuated tube at about 350 000 revolutions per second (Energy = 6 GeV). The velocity is near the speed of light. The electrons are focused and guided with several hundreds of magnets which are placed all around the ring. The diameter of a beam is less than the thickness of a human hair, about 1/10 mm. The particles loses energy in each revolution because they suffer centripetal acceleration, and any accelerated particle emits radiation (the synchrotron radiation). New energy has to be re-inserted into the system to replace this losses. An 'accelerating radio frequency system' is used to keeps the energy of the beam constant.
The beam losses particles because of collision between the electrons and the residual gas in the vacuum tubes and due to this would die after about one day time. Therefore, every more or less 8 hours the ring has to be 'refilled', meaning, a new beam has to be stored (injected) into the main ring.
To give it the needed energy of 6 GeV, the new electrons have to pass several steps: first they are accelerated up to 200 MeV by a Linear Accelerator (LINAC), just to be blown into a smaller accelerator ring (300 m circumference) called Booster. The Booster is responsible to speed the electrons up to nearly the speed of light. When this point is reached they get released to the main storage ring where they may circeling for the rest of their short life.

The storage ring of the ESRF consists of 32 strait and 32 bended segments. The bended segments are the sources of the x-ray radiation, while the strait segments hold some devices to increase the radiation of the beam. There are two different forms of insertion devices, called 'Wigglers' and

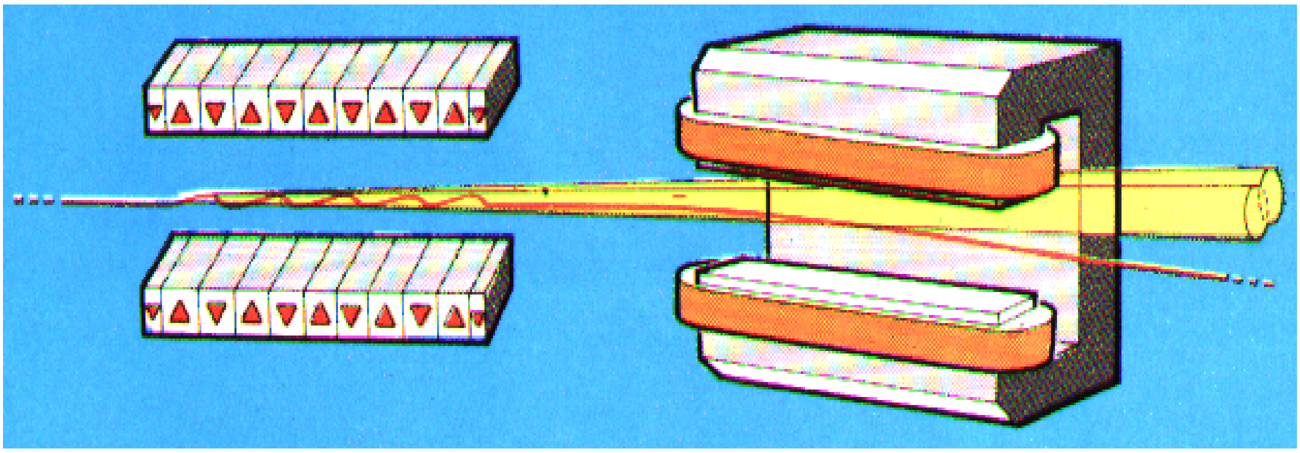


***Fig I.2*: The beam passes the 'wigglers' and 'undulators' before leaving the ring in a curve-segment**

'Undulators', which both consists of arrays of magnets acting on the electrons trajectories. This disturbs the strait path of the beam, forcing it to a slightly undulated course, causing him to increases enormously the intensity of the emitted radiation (see figure I.2).
In the bended segments, the electrons get bent by the force of large electro-magnets. The radiation yet isn't influenced by this force and continues its straight path parting from the beam in a tangential direction. The rays then are forced and guided to their experimental setups in the beamlines, by slits and optical systems (see Fig I.3).

I.3. The Beamlines

The radiation emitted by bending magnet and specially by ID's opens up a wide and sometimes unexplored spectral region for research applications. The photon source, however, cannot be used as it is, and must be complemented by sophisticated optical instrumentation necessary to process the x-ray beam, thus making it suitable for practical experiments. This is done in the beamline. The beamline consist in ultra high vacuum tubes and cameras of a total length from 10 to 100m, containing elements to transport, focus, reduce, monochromatize, etc. the radiation. It also contain the experimental camera where the samples to be analysed are placed, and series of goniometers, detectors, monitors etc. to control and record all the important parameters for the experiment.
The optical components of the beamline can be passive or active. Passive components such slits or filters just reduce the divergence of the beam or its intensity by cutting parts of the rays. Active

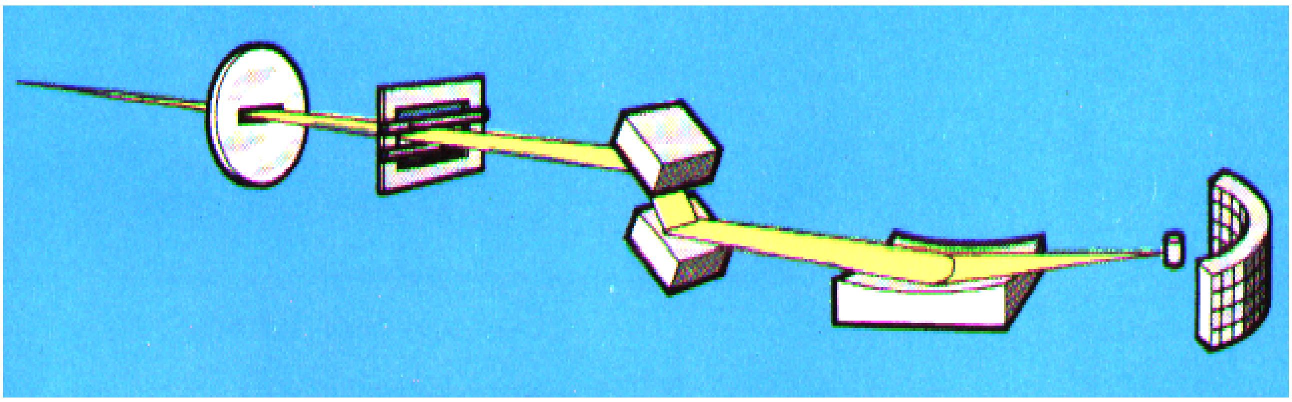


***Fig I.3*: Schematic of a beamline: The x-ray is guided and focused by slits and mirrors to its destination point**

elements are mirrors and diffracting elements. Mirrors are intensively used in SR beamlines to focus the x-ray beam. Lenses cannot be used for focusing the x-ray beam because the refraction index of the x-rays is almost the same in vacuum and in glass, so no deviation is produced. Even with mirrors, the beam penetrate into them unless we choose very grazing incidence angles, in the range where total reflection is produced. This handicap of using grazing incident forces to build very long mirrors (usually from 20cm to 2m) and have to be of a extremely high quality, because surface imperfections affect enormously the focal image. They usually have to be cooled otherwise they are destroyed by the force of the beam and the experiment is lost (imagine that the x-ray, when leaving the ring, has enough energy to melt a steal plate). The mirrors are also responsible for focusing the ray to a needle fine ray to increase its intensity and therefore the quality for the experiments.

Classical diffracting elements used in SR beamlines are gratings, crystals and multilayers. Gratings are like mirrors where a periodic ruling has been grooved to diffract the x-rays. Their application is in the range of Ultraviolet and soft x-rays, so they are almost not used at ESRF. Crystals are natural or synthetic materials with a periodicity in the crystalline planes that can be used for diffracting and therefore monochromatizing the x-ray beam. Almost all the beamlines have at least some few crystals. The crystals can be curved to focus the beam at the same time in which it is monochromatized (a single wavelength is selected). Multilayers are synthetical structures with periodicity larger than crystals. Recently new optical elements are used like Bragg-Fresnel lenses and capillary tubes.

My work at the ESRF is to find and test optimization methods for just these optical systems. [INFO]

II. THE PROBLEM

II.1. Computer Tools for Beamline Optics

A scientist or engineering in charge of the conceptual design of a beamline has to play with many possible devices and match them in order to arrive to the best practical solution for his particular needs. In a first stage, the experience and the present status-of-the-art in the field are the most important basis for a searching strategy. Then, very basic calculations can be done either analytically or by using simple methods to obtain a rough idea of the beamline performances for a given optical configuration. In many cases, the difference in performances given by different options are very small, and often are dependent on technical parameters that cannot be considered in a simple calculation. In other cases, the required quality of the optical devices are limited by the existing technology, and may not be manufactured, or the costs will increase exponential when approximating to these technological limits. In these cases, accurate calculations have to be done to know very precisely the performance of the beamline in terms of quality of the beam (transmitted flux, energy resolution, focal spot sizes, etc.) before the construction of the beamline. These calculation are generally done with the so called “ray-tracing” technique, which actually is the most powerful computation technique for such simulations. Ray tracing is a powerful tool which allows “optics experiments” with the computer. The philosophy consists in sampling the x-ray beam in a finite number of individual rays, and calculate the trajectory of each ray along the defined optical system. The exact calculation of the trajectory and absorption of each individual ray is done iteratively by the computer, and the macroscopical quantities are the statistical quantities of the ray ensemble.
The ray tracing technique is well known in the optics domain, and there are many commercial packages to perform ray tracing in the visible region, suitable for calculations of camera objective, microscopes, telescopes, etc. In x-ray, specially with synchrotron radiation, the choice is Shadow [see appendix]. This computer package can ray-trace any x-ray optical system and predict image shapes and intensity with great accuracy. It can simulate the SR sources as insertion devices (wigglers and undulators) as well as bending magnets, and allows the simulation of the beamline optical system including reflecting elements (mirrors), diffractive elements (gratings, crystals and multilayers), filters, slits etc. It also allows calculations of the detriment of the beam quality due to the surface errors in the mirrors, which must be considered for a realistic simulation.

II.2. Why do we need global optimization

Ray tracing calculations sometimes becomes a tedious task, because the developer has to analyse his system under many assumptions of, for example, photon energy, slit aperture, incidence angles, etc. Each set of parameters need a definition of the system, a run of the code, and an analysis of the results using suitable post-processors. In particular, if one wants to optimise one beamline parameter, an adequate sampling of this parameters must be provided, then the individual calculations have to be done, and a suitable comparison of the resulting parameters gives an idea of the best value for such parameter. This can be done manually only for few points of one parameter, but if one wants to optimise simultaneously several parameters of the beamline, an exhaustive search of the best value is impractical, and at the present the only solution is the heuristic intuition of the developer.

In this work, we analyse the feasibility of the automatization of such search of the best set of parameters that optimises a beamline configuration, by using modern global optimization tools in connection with the Shadow ray tracing program. Such methods are necessary because the computation time for calculating a gridding of the parameter space is impractical, and the different nature of the parameters (distances, curvature radii, angles) that can be mixed together does not guarantees the existence of a single maximum in the function to optimise. In addition, the definition of this Figure of Merit function (FM) is very critical to the success in the optimization process, and can also be source of local maxima. Also statistical fluctuations in the calculated figure of merit value, due to the nature of the ray tracing method, may create a local roughness in the FM surface. All these arguments drove us to do not consider the standard local optimization methods for searching the best value of the beamline design parameters. Two different approach to global optimization are considered here: genetic algorithms and simulated annealing. Both methods have successfully been applied to different fields in science and engineering and are, in principle, suitable for our problem. In addition these methods have been used in optimization of optical systems in the visible range, and are available as an integrated part in several commercial packages. For a good review see [Schafer,94], [Zeyher,94]. The nature of our problem is not the same, but the demonstrated interdisciplinarity of these methods are good points to consider them.

III. PRINCIPLES OF THE USED METHODS

III.1. Genetic Algorithms(GA)

Like the name is indicating GA have something to do with genetic, actually they are a form of artificial genetics. It is an attempt to bring biological evolution down to machine level and try to use the same principles nature has used for so many centuries since the dawn of life. Just to emphasise this fact, the developers of GA have often used the original biological names and descriptions of genetic to describe the procedures and methods of GA.

In biological terms genetic means:

- A creature is represented by its chromosomes
- Evolution is done by changing and mixing these chromosomes and testing the outcome
- If it's doing well it will survive and reproduce
- If it fails the test (life itself), it dies
- The better you pass the test, the better is your chance to reproduce more often
- While reproducing the chromosomes of different beings are mixing
- Sometimes mutation inflicts some new unforeseeable results

In computer terms 'genetic' means:

- A solution is represented as a string (called chromosome)
- Different solution-strings of a problem are called a population
- A fitness function tests all chromosomes
- The better a chromosome passes this test the higher is its chance to reproduce
- During reproduction the chromosomes will mix with other chromosomes using crossover
- A mutation function helps to avoid local minima by adding a random momentum

You can see that the computer approach isn't so far away from the biological reality. Here is a small description how these methods work:
The possible solutions of the given problem are decoded into a bit string, the chromosome. At the start these chromosomes are chosen randomly, that means every chromosome represents one guess for the solution of the problem. Now a fitness function is calculating the actual fitness for every chromosome. A (more or less) random selection function (e.g. Roulette Wheel Selection) is selecting the chromosomes for reproduction, the higher the fitness the higher its chance to be chosen for reproduction. Crossover is done. If mutation occurs (depending of a random function, normally set to about 2 - 10%), it occurs now. The child chromosome is inserted into the new population (the new population may be totally new one or one which consists of a number of chromosomes of the old population). The fitness is calculated for the new chromosomes and the circle close.
This procedure loops until a given stop criterion is reached. A graphical representation of this

procedure with its single steps, is given in Fig III.1

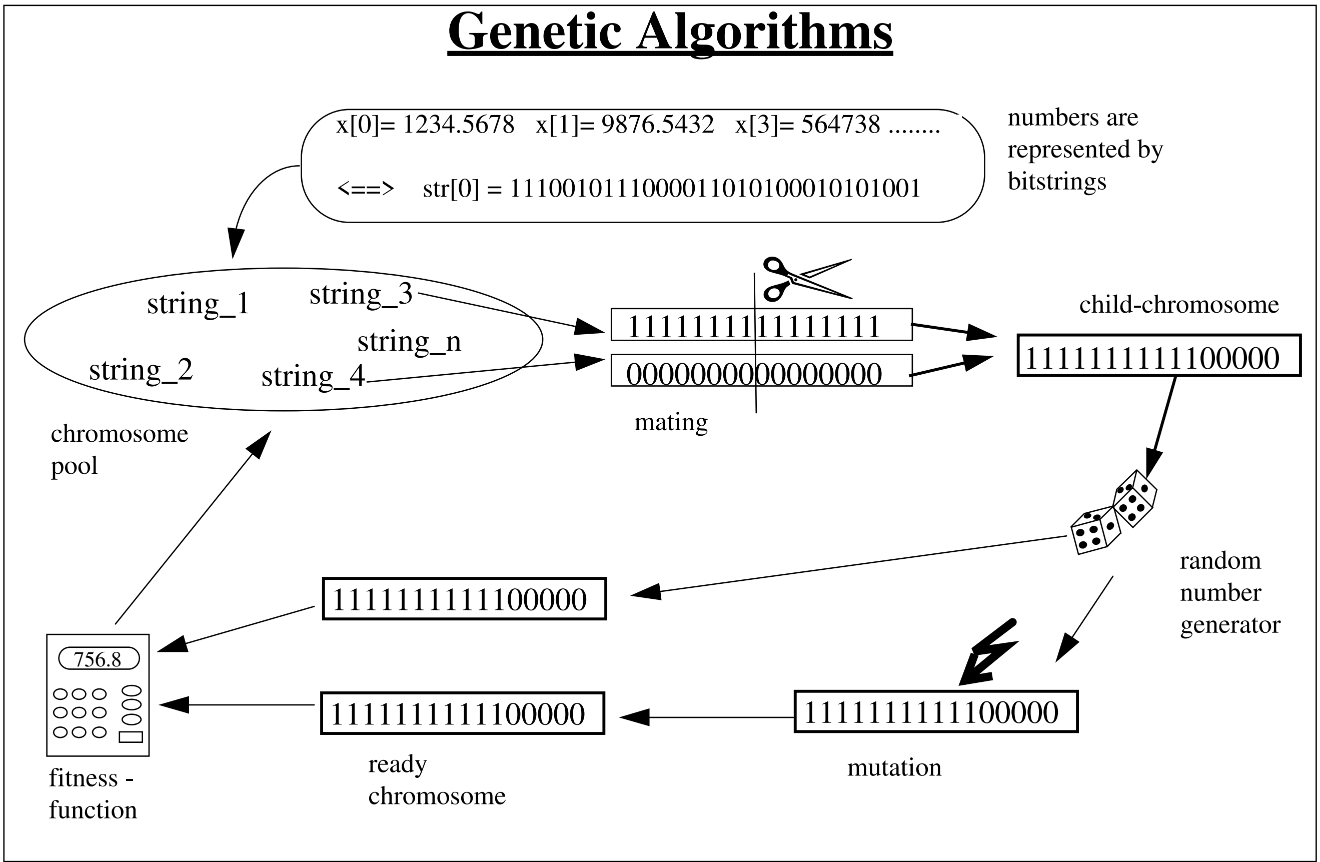


***Fig III.1*:** **Illustration of the Genetic Algorithm work cycle**

For a better understanding an explanation for the most important procedures will follow here:

Crossover: Two chromosome strings get ‘cut’ at the same string position and recombined, by putting the head of the first string and the tail of the second string together. Depending on the used method the parent strings are kept and with the child returned to the chromosome pool or they are discarded.

Roulette wheel selection: A very important function is the selection of the chromosomes which are allowed to reproduce their code in the crossover procedure. It is important for a efficient working of the Algorithm that two criteria are matched by the selection function:

- The higher the fitness of a chromosome the higher should be the probability that it is chosen for reproduction.
- Even a bad chromosomes should have the chance to reproduce. Sometimes important information is hidden in worse chromosomes which isn’t stored in the better ones. By mating a good and a bad chromosome sometimes a much better chromosome evolve.

The roulette wheel method engages this problem by giving every chromosome a special number of slots on an imaginary roulette wheel. The number of slots (in practise it is a percentage) is proportional to the fitness of the chromosomes. Now a fictive roulette ball is thrown into the wheel (a random number generator) to decide which chromosomes will win the game of reproduction. There are of course much more methods of making a fair selection, but for the general understanding of GA they are not important, therefore they are not considered here.

Mutation: The mutation procedure is a possibility to avoid, that your algorithm is trapped in a local extrema, where it can't escape any more. After a time the chromosome pool will converge, that means all bad chromosomes were replaced by their better children. When this stadium is reached it's not longer guaranteed that the procedure of crossing different chromosomes from the pool will yield new children. Nearly all chromosomes will have a similar structure, because they all evolved bye mating with their ancestors. Now, for a example, consider that all the remaining chromosomes have a '1' bit in position four of their gene-string and that for a exact solution of the problem a '0' is needed in just this position. It isn't possible to achieve this goal by the crossover technic, because no chromosome of the pool has this missing part in its string. Mutation (and a good portion of luck) although, have the possibility to switch this bit randomly to the desired value.
Of course, mostly mutation won't yield good results at all, the most chromosomes changed by mutation will become worse instead of better, but due to the fitness test which every chromosome has to pass, this is not a problem. All the chromosomes which have been altered by mutation in a bad direction, are thrown out by the merciless struggle of the fittest, which goes on into the chromosome pool. On the other hand is the rate of mutation such slow (recommended: 0.05 - 3 %) that it seldom changes a chromosome in this rude manner.

I hope this short introduction into GA is sufficient to understand the principles we followed in Optimir. If you feel now excited enough to learn more about this, in my eyes, fascinating combination of biological evolution and computing. The following references should lead you to interesting and important resources of information about this subject: [Goldberg,89], [FAQ], [Rawlins], [Schoeneburg,93], [Stein,92]

III.2. Simulated Annealing (SA)

The second method we used is Simulated Annealing. Although it is, like Genetic Algorithms, a method for global optimization, its principle is total different. Like in GA, the name indicates from were the developers have borrowed their ideas: Annealing is a physical process used for controlled cooling in generating perfect crystals. The right decreasing of the temperature in the cooling process, is an important factor in determining the outcome of the crystal growth. If the temperature is decreased to fast, the crystal will not be able to find its termal equilibrium and it will stop in a metha stable chemical state. Those crystals are not useful for the most purposes, because they lack a great amount of the typical characteristics that a perfect crystal obtains.
The thermal equilibrium is characterised by the Boltzmann Distribution (BD). The BD gives the probability of the crystal being in a state **i** with energy $\mathbf{E_i}$ at a temperature T:

$$P_T(X) = \frac{1}{Z(T)} \exp\left(\frac{-E_i}{k_B T}\right)$$

where **X** is a stochastic variable denoting the current state of the solid, **Z(T)** a normalisation function dependent on the temperature and k_B the Boltzmann constant.

The process of annealing is very similar to the search of global minimas in a wide spreading and uneven function-plane. By making the following two assumptions the annealing process can be used for combinatorial optimization problems:

- the states of a physical system are equivalent to one valid solutions of a optimization problem.
- The energy of a physical state is equivalent to the cost of a solution in the optimization.

The crucial function, which makes SA so different from the most other algorithms, is the function determining if the next point on the search path is accepted or not. The most important different to the most common standard methods for optimization (like e.g. the Gradient method or the Downhill Simplex method [NUMREC,92]) is that SA is ***not*** like its colleagues a greedy search mechanism. Most standard mechanisms walk always down the slope of a function graph, greedy to find the minima of the function they examine as fast as possible. This greedy behaviour is the cause why these optimization procedures (as good however they may be in their search in a small area of view) aren't able to find a global minima.
The SA algorithm, is exploring its surroundings, instead of throwing itself in the first best minimum which is offered to him. Instead of walking down a function slope all the time until a (local ?) valley is reached, the SA algorithm will especially in the beginning of the search, when the 'temperature' is high, allow sometimes also moves which go upwards the slope of the graph. With the ongoing of the search (and therefore the decreasing of the 'temperature') the algorithm allows less and less moves in the undesired upward direction, but concentrates its search downwards (this guarantees the convergence of the search). The mechanism behaves like a wanderer who walks in a hilly mountain area climbing from time to time on a near hill to get a better survey of the whole area and to find the best way to go back to the valley.
The following function, called Metropolis criterion[Metropolis,53], is responsible to determine if a new value is accepted or discarded:

$$P_T\{\text{accept } j\} = \begin{cases} 1 & \text{if } f(j) < f(i) \\ \exp\left(\frac{f(i) - f(j)}{T}\right) & \text{if } f(j) > f(i) \end{cases}$$

where $\mathbf{T} \in \mathbf{R}^+$ is a control parameter equivalent to the temperature in a natural annealing process. We (and so does the rest of the SA world) refer to this parameter as 'temperature' (why make it more complicated).
To select the new search-point following procedure is used:

1. Select randomly a new point. The range of possible selection is strongly influenced by the actual temperature of the system (In the beginning of the search, the range should span the whole search area).
2. Applying the new point to the Metropolis function:
   - If the cost of the new selected point is smaller than the old one => accept it.
   - If the cost of the new point is greater than the old one => accept it with the probability P_T.
3. Repeat steps one & two until termal equilibrium is reached
4. Decrease the temperature, following a given temperature scale (The selection of the right temperature scale is the most important and the most difficult operation, because this scale is responsible for the accuracy and the performance of the SA).
5. Restart the procedure with the new temperature.
6. Do this until a stopcriterion is reached.

Fig III.2 shows a pseudo C code illustrating the SA algorithm[Aarts,93].

The choice of the starting values is very important for the SA algorithm. Depending on your problem, you have to chose the proper values or otherwise the result may be not sufficiently accurate. If you, for example, chose the starting Temperature to low the algorithm may not have enough freedom to explore the whole area, in the beginning of the optimization, but on the other hand, if you choose the value to large, it may take decades (at least it will feel so) before the algorithm will finish the optimization. If you work on a multiuser terminal other people will, for sure, be annoyed by the CPU time you waste.

Fig III.3 is an attempt to visualise the search strategy of the SA algorithm. The search function can be imagined as a rubber ball, which is thrown onto a hilly plain. In the beginning the ball has a lot of kinetic energy and every time it bumps into a wall he will be rebounced losing a little bit of it. With the high energy in the beginning, the ball has no problem to leave the different valleys it visits during his journey, because it has enough power to jump over even the highest peeks. After a time, when he has lost more and more of its energy, it will find it harder and harder to leave especially the deeper valleys. Its movement will finally stop in the deepest valley, which it can't leave due to a lack of energy.
This, of course, is only a ***very*** simplified image of what really happens in a SA procedures, but it may help to see the main idea.

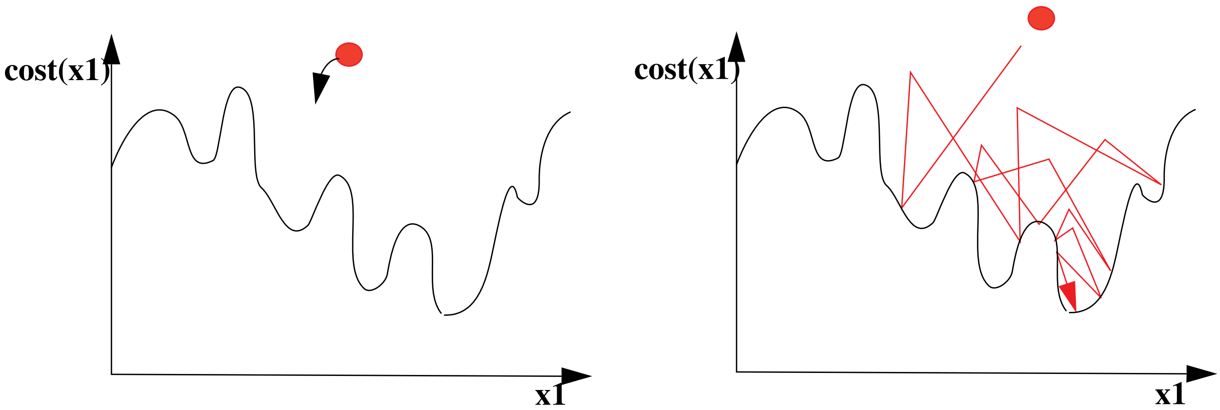
SA have been used successfully in many different global optimization problems (see e.g.[Kirkpatrick,83],[ASAPAP],[NUMREP]). The method gives a statistical guarantee to find a

```
void Simulated_Annealing ()
{
    INITIALISE(i, T, Loops) ;
    do{

        for (l=1; l<Loops; l++){

            Generate( j from Si) ;

            if ( f(j) ≤ f(i) )      i = j ;

            else  if (  exp( (f(i) - f(j)) / T )   > random[0, 1) )    i = j ;
        }

        k = k+1 ;
        Calculate_Number_of_Loops(Loops) ;
        Calculate_Temperature(T) ;

    }while(stopcriterium_not_reached) ;
}
```

Fig III.2**: Pseudo C program of the SA algorithm**

global solution. The big disadvantage is that some optimization problems will consume an incredible amount of computational time. For those cases SA is mostly not very practical (unfortunately our own case belongs to this group).
Many attempts have been made to improve time relevant routines of the basic algorithm to boost the performance of the SA. Some solutions look very promising and are worth a further investigation (see e.g. [Vanderbilt,84], [ASA,93], [ASAPAP]).
SA is a method to consider were the general standard methods fail. It is obviously not a method to use in a strait function with only one optima, therefore each of the ‘old fashioned’ algorithms will perform much better. SA will find its place in the world of optimization in those parts, where standard methods won’t archive good results, because they run in local minimas were they are stuck forever.
SA is not the new glamorous ‘general problem solver’ for every problem one may encounter in his work. SA is best used in cases were you know that local optima may disturb your search or were you don’t have the slightest idea what your function is looking like. In those cases SA may be a mighty and useful tool, making your live much easier.

Simulated Annealing



1. Select a new configuration - the higher the 'temperature' the larger the range of the new parameters.
2. Calculate the cost of the new config. (call the function to analyse with the current config.)
3. Decide if you wanna accept this new config. by following rules:
   * if cost < last_accepted_cost => ACCEPT
   * if cost > last_accepted_cost => ACCEPT with a probability prob(tmp)
5. Repeat procedure (either with the new or the old config.) until thermal equilibrium is reached
6. If temp > 0 : Decrease temperature and restart loop

The decreasing of the temperature is the most influential part of the system. It also is responsible for the speed and the accuracy of the system:

$\boxed{\text{accuracy} \sim \text{speed}^{-1}}$

Fig III.3: Principle of the Simulated Annealing method

IV. THE PROGRAMS

IV.1. Optimir

The following description of the program is actually a compressed reprint of the official documentation I wrote for the program:

What's that stuff about

Optimir is an optimization program using Genetic Algorithms (GA), I developed during my time at the ESRF (1.3.94 - 31.8.94). To avoid the reinventing of the wheel, Optimir has evolved from another GA program called PGA (Parallel Genetic Algorithms testbed Ver.2.4) [PGA,93]. PGA was developed at the University of Edinburgh in the department of Artificial Intelligence. It was built with the purpose to create a educational program to understand the principles of GAs. Optimir takes advantage of the well tested and efficient subfunctions that PGA offers. The body of the program was rewritten in its main parts to fit its new purpose, the optimization of optical systems (the name OPTIMIR means: OPTimization of MIRror). The program, however, has now reached a level of abstraction, that it can be applied to nearly any optimization problem.
The idea which enables this generality is to treat the fitness-function as an extern black-box-function, stored in a separate, independent running program. The communication between these two programs is done via two files, which pass all the important information. The first file passes the parameters to evaluate to the fitness-function program, the second file returns the calculated fitness to Optimir. The names of this two files are passed by optimir to the fitness function as command line parameters using a system call.

To adapt a new problem to Optimir you don't have to recompile or change anything in the program itself. You only have to write a program, that contains your function to optimise (the fitness-function) and which is able to read and write two files, all the rest of the optimisation is done by Optimir.
Not even death is free today and you have to pay a price for this generality of the program. The price is that the file-handle-operations slow down the speed of the optimization procedure. We have decided to pay this price anyway by considering the following points:

- you don't want people to change, editing and rewrite parts of a standing, tested and well working program. Our approach allows errors only in their own program (= the fitness function). (This may prevent you from being woken in the middle of the night by people calling to complaining about your program)
- especially in our case the fitness function is such time consuming (we have to pass the parameters into a ray-tracing program which needs about one minute to trace all the rays) that the extra time spend for file handling isn't significant.
- by letting people changing the core program for their purposes, after a short time the code of the program would probably be so messed up that not even the authors or maintainers, could repair it or find the bucks.
- you can treat Optimir as a black box. You don't have to maintain it. Users don't have to know more than the action of the parameters.

```
xterm
    USAGE: optimir [ -arg1 -arg2 ....]
    -a        Flip adaptive mutation flag (only for 'one') (FALSE)
    -A<nr>    Automatic mode, stop after max. <nr> populations.(0=stop only if stable)
    -A-<nr>   Automatic mode, stop after <nr> populations.(doesn't stop if stable)
    -b<n>     Set selection bias. (1.5)
    -c<n>     Set crossover rate (only for 'gen'). (0.6)
    -C<op>    Set crossover operator. (one)
    -e<fn>    Set evaluation function. (extern)
    -g        Use graycoding instead of binary. (FALSE)
    -h        Display this information.
    -i<n>     Set reporting interval in generations. (10)
    -l<n>     Set # of generations per stage. (100)
    -m<n>     Set mutation rate. (0.05)
    -M<n>     Interval between migrations. (10)
    -n<n>     Set chromosome length. (32)
    -p<n>     Set number of chromosomes per population. (50)
    -P<n>     Set number of populations. (3)
    -r<op>    Set reproduction operator. (one)
    -R<nr>    Set seed for random number generator. (time(NULL))
    -s<op>    Set selection operator. (rank)
    -S<nr>    Set convergence limit. (1e-9)

    -FC<file> Read params from <file> instead of command line (FALSE)(config.dat)
    -Fd<nr>   Dump chromos after <nr> stages(only auto-mode , 0=no dump).(1)
    -FEF<file>Set new name for Extern Fitness function(ga_fm)
    -FEI<file>Set new name for Extern Fitness function input file(radii.dat)
    -FEO<file>Set new name for Extern Fitness function output file(result.dat)
    -FF<file> Initialise chromos from file(saved with -FI opt.) (geninit.dat)
    -FG<file> Set filename for gen output. (bestgen.dat)
    -FI<file> Initialise population,save & exit program (FALSE)(geninit.dat)
    -FM<file> Set filename to read the limits of the axis (minmax.dat)
    -FO<file> Also log output in <file>. (FALSE)(output.dat)
    -FR<file> Restore saved program from <file> (savepop.dat)
    -FS<file> Set filename to save program (savepop.dat)
    -Fs<nr>   Save program after <nr> generations (500)

    Crossover operators ... one, two, uniform.
    Selection operators ... rank, fitprop.
    Reproduction operators ... one, gen.
    Evaluation functions ... extern
expgf{boeddi}94:
```

Fig 1II.1: **screenshot of the help message (`optimir -h`)**

The program offers a lot of different switches, which are able to influence and control the outcome of the optimization. Different GA-approaches can be chosen to solve a problem. All these options are shown by calling the program with the **-h** option. The values in brackets are the default parameters. The default parameters depend strongly on your specific problem and they are ***not*** necessarily suitable for your case.
Fig.1 shows a screenshot of the '`optimir -h`' screen display. Beneath is a short explanation, what the single parameters are doing. If you don't have an idea what GAs are about at all, you might find it difficult to understand even the explanation. For those interested I would recommend the book from David E. Goldberg (at least my favoured). Otherwise a "quick and dirty" explanation can be found in the Appendix, but that covers only the most essential idea of GA (there is a lot more).

Options

The most of the options are self-explaining, but some of them are a little foggy. Therefore you will find here a short explanation. If you have still problems understanding their usage, feel free to contact me (boeddi@ai-lab.fh-furtwangen.de) or Manuel Sanchez del Rio (srio@esrf.fr).

We can divide the options in two groups, one related to general peculiarities of GAs and the other related to file management.

General: The most important parameter for the beginning are the ones defining the 'outlook' of the chromosomes and their populations, as well as the interaction between them.
The (**-P**) option determines the number of populations. Populations are separated chromosome pools, which work independently.
The only way of connecting the different populations is by setting the migration rate (**-M**) greater zero. This rate indicates after how many cycles a chromosome will be chosen, according to it's fitness, from one of the pools and migrate to the other pools (destroying the worst chromosome in each of these pools). The pool where the chromosome is taken from depends on the number of the migration-call (the first migration takes the first population, the second migration the second population,...). The number of members of each generation can be changed by using the **-p** option.

Depending on the number of parameters you wish to represent in a chromosome, the accuracy you need and the range of the parameters, you have to choose the optimal size of the chromosome string with the **-n** option. Because we store the parameter values in bitstrings and every parameter may have a different range in which its values may lie, we have to represent the real values in a form which takes account of this problem. The following two functions are used to do this job. A single real parameter value is converted by function (2) to an integer number, which represents the real number in its particular interval [min, max]. The resulting integer number lies in an interval [0, 2^{bpp}], where bpp is the number of bits, each parameter may use in the chromosome. The resulting integer values (one value for each parameter) are then converted into bit strings and concatenated, resulting in the chromosome. (Function 2 is used only if you give starting parameters (see below) to Optimir, otherwise only function 1 is needed by Optimir).

$$\text{param} = \left((\text{max-min}) * \frac{\text{intval}}{2^{\text{bpp}}}\right) + \text{min} \tag{1}$$

or vice versa:

$$\text{intval} = \left(\frac{\text{param - min}}{(\text{max-min})}\right) * 2^{\text{bpp}} \tag{2}$$

where $bpp = \frac{\text{size of chromosome}}{\text{number of parameter}}$ = number of bits per parameter , *min* and *max* the boundaries of the parameter range, *intval* the resulting integer which is stored in the string and *param* the real value of the parameter.

Therefore the length of a chromosome is: **n * bpp**, where n is the number of parameters.The number of parameters should be a divider of the length specified with **-n**, otherwise the last bits of the chromosome are ignored by the calculations but carried around anyway, reducing the speed of the single algorithms and increasing the needless usage of memory.
(***Warning***: The *value* is represented as an integer, the rounding may cause undesired results if the

string length isn't sufficient). For example: A parameter with the value 1.5 in the range [0,10], is stored in a bitstring that allows 8 bits for each parameter, as value= 10 (9.6 rounded up). If this is recalculated using function (1) to give a parameter real value of param=1.5625. For bits=16 <-> param=1.499938965; for bits=32 <-> param=1.499999999; ....
Conclusion: 8 bits / parameter is normally not sufficient for such a large interval, the more bits you allow per parameter for representing a number the more accurate is the re-decoded number. For getting better results you have to either decrease the range of the parameter or increase the bits per parameter (**-n**) to allow a smaller granulation in calculating the numbers. But remember: The larger you make the chromosomes the more time need the GA algorithm for handling the single chromosomes.

Optimir allows you to choose between two different running modes. The default is interactive, which means that after a certain number of loops, specified with the **-l** option, the program stops to ask for the next action. Type one of the characters in the brackets to select the action. (A)gain will restart the optimization procedure, (C)ontinue will continue for another (**-l**)-loops and then ask again, (G)ene-print writes the actual best genes of each population to the gene-file (**-FG**), (D)ump dumps the actual chromosomes into a file <outfile>.chr (this option only appears if the **-FO<outfile>** parameter was chosen at the command line; the dumped chromosomes are stored into a file with the name of the outputfile plus the suffix '**.chr**') and (Q)uit finally will quit the program by giving you the choice to save the chromosomes for a later run before leaving (restart with **-FR**).
The second mode is the automatic mode which can be selected by the **-A** option. In this case the program won't stop until it finished its optimization. You will see the output, but you have no chance to influence its run with interactive actions. The number following the **-A** option indicates the number of generations done before stopping. If this number is positive it means it will run <nr> loops or until it converge. If the number is negative it will run exactly <nr> loops and if <nr>=0 it will run until it converges. Convergence is reached if:

$$|\text{average fitness} - \text{best fitness}| < \text{stable\_factor}$$

The stable_factor can be set by the **-S** option. If an population seems to be stable a '=' sign will be printed to the right of the population number (pop). During interactive runs the stable_factor has only the function to visualise the convergence in this manner, it won't influence the flow of the program itself.

The screen and the output files are ***not*** updated in every loop, this is for two reasons. First it is time consuming to do so and second relevant changes won't show up in one or two loops anyway, a reasonable time to check the results is in a range of 10 - 50 loops. This update frequency can be set with the **-i** option. In an interactive run, the update frequency must be smaller or equal the **-l** stage frequency and should be a divider of it. If the **-i** value is larger than the **-l** value it is set to the **-l** value (otherwise the program would stop without showing the new results).
The following options selecting different functions which are used in the GA process. The **-C** option selects the crossover procedure:

- **one:** one-point crossover. Cut the chromosomes at one random point. The child bears the first part of the first parent and the second part of the second parent.

- **two:** two-point crossover. Cut the chromosome at two points. The child contains the middle piece of one parent and the start and end pieces of the second parent.
- **uniform:** uniform crossover. For ever gene of the chrome is decided which one of the parents spend it's chromosome.

The number of different permutations is 2(L+1) for one-point, L(L+1) for two-point and 2^L for uniform crossover (L= Length of chromosome).
The selection function which determines which chromosomes will be used for mating, is chosen by the **-s** option. The choices are:

- **fitprop:** Chromosomes are selected in proportion to their fitness.
- **rank:** Putting the chromosomes in rank order and selecting in proportion to their rank (the higher the rank the better the chance of selection).

The bias **-b**, influences the weight of the rank position. If bias=1.0 then the probability is a linear function of the ranking, a value of 2.0 is favouring the first member much stronger and you get a parabolic probability function. Reasonable values for **-b** are normally between 1.0 and 2.0.
The **-r** option is determining the kind of reproduction. There are two kinds of reproduction:

- **one:** Two chromosomes are selected from the population and crossed. The child may then be mutated depending on the mutation rate (see below) and is re-insert into the population by replacing the worst chromosome.
- **gen:** The whole population is replaced by a new one. Two chromosomes are selected and crossed with a probability governed by the **-c** option. The new chromosome (it may be a crossed chromosome or just one of the parents) is then mutated with a probability determined by the **-m** option and then insert in the new population. This procedure is repeated until the new population has the same size than the old population.

The mutation is determined by the **-m** option which gives the probability that a chromosome passing the mating procedure is mutated by flipping one of its bits. Normally the mutation rate should be kept small, reasonable values for -m are 0.02 - 0.1 (2 -10%), but this may depend on the problem to solve.
If you have selected **one**-reproduction than you can select adaptive mutation (**-a**) instead of the standard mutation routine. In this case the mutation occurs with a probability, which is calculated depending on the similarity of the parents. If the parents are identical the probability is 1.0, if they are totally different (every bit) the probability is 0.0, the other probabilities are calculated by comparing bit by bit, returning a higher probability for every bit equal in both chromosomes.
The next three general options are of a more or less experimental nature and their usefulness may be questionable. However, feel free to use and exploit them, in some cases they can be extremely useful.
The **-g** option is using graycoding instead of normal bitcoding (converting a number to base 2) to represent the values in the chromosome. Graycoding is another form of number representation in a bit string. If you are interested in graycoding, I recommend [Goldberg,89] for an introduction. I haven't got better results by using the graycode mechanism, but this may depend on the problem to solve.
The **-R<seed>** option grants the possibility to set a seed for the pseudo random function, this may be useful for exploring the optimization algorithm with it's different parameters. The random

numbers are dependent on the seed and in runs with the same seed the same sequence of random numbers will be generated This makes the program deterministic. The default is an initialisation with the actual system time, supplying a different seed for every new run of Optimir.
Finally, the **-e** option is a relict from PGA (the original program, which was used to construct Optimir), which enables you to select different built in fitness functions. This may be useful anyway ! If you decide to run the program for mostly one specific problem and you want to avoid the time consuming file handling, than you may think of writing a function that has the form ***void eval_new(CHROMOSOME *pop,int number).***
The function accepts a pointer to some chromosomes to test and an integer value specifying the number of chromosomes to test. The function must write the calculated fitness for every chromosome in ***pop[idx].fitness*** where idx $\in$ [0,number]. Look at the ***eval_extern()*** function to get a better impression.

File-options: The following parameters specifying the files where information is retrieved from or written to.
The format which is used in the files that read config-information is explained in the header of the files `*.tmp`, which are stored in the 'TEMPLATE' subdirectory (Don't change these files, they are supposed to be examples to explain the formates of the different input files. Make a copy instead and leave the `*.tmp` files untouched.).
The option **-FC[<name>]** indicates that the information for the current run is **not** taken from the command line, but stored in the file, named after the option (if no file is named the default name is taken). If **-FC** is set, no other parameters are allowed !
The option **-FM<name>** specifies the name of the file containing the range of the single parameters and additional starting parameters (***Warning:*** Normality, it's **not** a good idea to give starting values to the optimization procedure, because that is in strong contradiction to the idea and principles of the GA method. It even may prevent you from finding a global solution).
The **-FO[<name>]** option lets you specify an output file where the results of the optimization are logged during the run. This files may become very large during an extensive optimization. The file specified by the <name> following the **-FO** option contains information of the average and the best fitness in a particular state.The interval for this is set by the **-i** option. Another file with the name `<name>.chr` is also created containing all chromosomes of a particular state. The interval for this updating is set by the **-Fd** option. (*This works only in the automatic (**-A**) mode* ! If the program is running in interactive mode the ***(D)ump*** command is responsible for dumping the chromosomes to the file). Due to the fact that the `'*.chr'`-file contains every chromosome of every population, the file may become of gigantic size, if the **-Fd** specifies a too small number of dump intervals. If you don't need the chromosomes for your analyses of the optimisation it is normally a good idea to just switch off the dump option with `-Fd0`. When running Optimir in automatic mode, the **-FO** option is strongly recommended to get information about the average fitness.
If you don't need this information about the average fitness, the gene-file, which is specified with **-FG<name>** may be sufficient (this file is always created and updated in a automatic run). The **-FG** option exists only to enables the change of the default name, the gene-file will always be opened no matter if you set **-FG** in the command line or not (In interactive mode the (G)ene-print will put the best genes into the file.).

The options **-FS<name>**,**-FR[<name>]** and **-Fs** are responsible for saving and restarting a

optimization run. This may be important especially for long optimisation sessions (imagine a run of 4 days, with a shutdown 10 minutes before your programme finished).
The **-FS[<name>]** option is specifying the file name for the save file and the **-Fs** option is specifying after how many loops a backup is made.
To restore the optimisation, run optimir with the **-FR[<name>]** option as the only parameter (specify the name of the save-file if it has not the default name). If you feel brave enough you can even modify the save-file and change some parameters during the continued run. This is a good method to continue a run where you have set the **-A** option with too small a number (increase the number after the **-A** option in the second line of the save file).

The options **-FI[<name>]** and **-FF[<name>]** can be used to speed up the initialisation at the beginning of an optimisation.Normally, Optimir generates random bitstrings to use them as start population. If you have a very time consuming fitness-function the initialisation alone may need days.
The **-FI** options grants the possibility to initialise a pool of chromosomes, write them into a file and exit optimir. These chromosomes are just a collection of random generated bitstrings and their fitness, which is calculated using the specified fitness function. You can use these chromosomes in *every* optimisation, which uses the same fitness function and the same chromosome length (**-n**), as those selected in the initialisation run. The only parameters showing an effect in a **-FI** run are: **-p, -P, -n, -R, -FE***. Every other parameter is ignored or may even causing problems.
To use these chromosomes add the **-FF** option to the Optimir call. This will order the program to read the chromosomes from the specified file instead of generating and testing the start population a new.The product of the values specified with **-P** and **-p must not** exceed the number of stored lines in the init-file. In the case that more lines stored in the file than needed for a particular run, the exceeding lines are ignored.
The chromosomes in the init-file are ordered by their fitness, so it is a good idea to run the utility program `'mixgenes'`(Call it without parameters to get a help screen) to shuffle the chromosomes before using them. Otherwise the first population has all the good genes and the last population all the worse genes.

Extern Fitness Function: Three parameters are responsible to specify the external fitness function. With **-FEF<name>** you can change the name of the external function. **-FEI<name>** specifies the name of the file containing the parameters to test and finally **-FEO<name>** names the file where the results are expected. You don't have to use this parameters at all as long your fitness function is using the default names.

The External Fitness Function

The external Fitness-Function(FF) is a separate stand-alone program, which is called by Optimir using a system call. The command string of the system call has following form:

`" program_name in_file out_file " ,`

where `'program_name'` is the name of the program containing the fitness function, `'in_file'` is the file optimir wrote the parameters to analyse and `'out_file'` is he file

where Optimir expects the results of the analysis.
The FF-program must be able to do following things:

- read the file 'in_file' and recognise how many lines this file contains. Each line contains a set of parameters to test. The parameters are separated by blanks and of the type double.(***Warning:*** The number of parameter-lines are **not** constant. They vary from time to time).
- calculate for each line of parameters the fitness and write the results to the file 'out_file'. Use a new line for each result. If there are less result-lines in the file 'out_file' than were parameter-lines in the file 'in_file', Optimir will complain and exit with an error, after the FF-program returned the control back to Optimir. In other words: You must write for every parameter-line one result-line.

That is all that you have to consider, by writing the program. Everything else is handled by Optimir.

Utilities

In the sub-directory UTILITYS you will find some utility programs which may be useful. Each program will display a help screen when called without or wrong parameters.Here is a **short** description of them.

mixgenes: Will shuffle the genes of the initialisation file created with the -FI option of Optimir. It will read exactly the number of chromosomes specified with number from the file '`file.in`', shuffle them and write the result into the file '`file.out`'.

Syntax: `file.in file.out number`

str2flt: converts the chromosome stored in the file '`genefile`' into its real values. The range of the variables are taken from the file '`limitfile`'. If '`limitfile`' is not specified, '`minmax.dat`' is assumed. The format for the '`limitfile`' is the same as in Optimir. Only the first chromosome of the file is considered, if more chromosomes are stored in '`genefile`', the surplus ones will be ignored. The output will be send to stdout (normally the screen).

Syntax: `str2flt genefile {limitfile}`

gray2flt: same as str2flt, but decodes a graycode coded chromosome instead of a binary decoded.

Syntax: `gray2flt genefile {limitfile}`

flt2str: converts the real values stored into the file '`file.in`' into an chromosome string which is then written into the file '`file.out`'. The format of the '`file.in`' file is shown into the file '`flt2str.tmp`'. This program may be used to convert real numbers into a string and then back again to see if the string length is sufficient for the wanted precision. (see chapter 2 for a short discussion about this problem).

Syntax: `flt2str file.in file.out`

Examples

Here some examples of how to run Optimir:

optimir -P2 -p100 -n80 -A1000 -i50 -FMlimits.1 -FGbestgenes.1 -FOoutfile.1-FSsave.1

This will run a case with 2 populations each containing 100 chromosomes with a chromosome length of 80 bit. The optimisation would stop after either 1000 loops or after it converged. The screen and the output files would be updated after every 50 loops. The range of the parameters are read from the file '`limit.1`', the default name of the gene dump file is changed to '*`bestgenes.1`*', the output is logged into the file '*`outfile.1`*' and finally the default name for the save file is changed into '*`save.1`*'.

optimir -m0.2 -Ctwo -rgen -sfitprop -l200 -FO

This runs the defaults for -p and -P, changing the mutation rate to 20%, selecting two-point crossover, the gene reproduction method and the fitprop selection scheme. The run is interactive, but logging the output also to the default file '`output.dat`'. The best genes are stored in the default file '`bestgen.dat`' and a backup is made in the default file '`savepop.dat`'.

optimir -p200 -n128 -FI -R1234567

This initialise 200 chromosomes of length 128 bit, calculate their fitness, write them into the file '`geninit.dat`' and then exit the program. The random number generator is initialised with '1234567', so the outcome (the chromosomes) is not really random. These are the only parameters that give sense with the **-FI** option, any other option would be ignored by Optimir. (Ok, it's a lie! You can also use the -P option to specify the number of chromosomes. In this case the number of chromosomes is calculated by: -P * -p. The difference is that the chromosomes are ordered within their population. But you should anyway run '`mixgenes`'(an utility program) before using this init-chromosomes and in this case the output is **absolutely** the same.)

optimir -FFchromos.dat -p100 -P2 -n128 -A1000

This runs a session with the init-chromosomes stored in the file '`chromos.dat`'. This file **must** contain at least (if there are more the rest is ignored) **-p** * **-P** chromosomes (in this case 200) and their fitness. The chromosomes **must** have a length of 128 bit. The optimisation will run for 1000 loops or until the populations converge.

The Example Problem

The example problem is a function of several Gausian curves which are combined to one 3 dimensional plane.
The C code for this function is in the file '`fm.c`'. The function was taken from a program for Simulated Annealing written by A.Bosin at the University of Cagliari[Bosin,93], but it fits perfectly as an example for GAs too.This function has nothing to do with the optical problems we had to work with, but it's added here only for the cause of giving an easy example for a fitness function.
The function has 3 peaks (the minimas of the original function were switched to maximas by returning the negative value of the function) with the values $f_1(9.94,-54.3) = 5.2$, $f_2(-31.7,87.2) = 8.0$ and $f_3(-85.4,0.60) = 9.2$. The global maxima is very narrow and the next best (local) maxima is a very wide one which often leads the search in a wrong direction. It has proven useful to make instead one very large optimisation more smaller ones to get a better statistical distribution. The most time intensive optimisations weren't necceseryly the best one (see Table 1).
The function reads data from the file '`gaus.dat`' to specify the Gausian curves. Fig. 2 shows the graph of this function

Table 1 shows the results gained with following parameter:

1. optimir -P1 -p100 -A1000 -FOout.1 -FGgen.1
2. optimir -P3 -p500 -A1000 -FOout.2 -FGgen.2
3. optimir -P5 -p500 -A1000 -FOout.3 -FGgen.3a -rgen
4. optimir -P1 -p1000 -A1000 -FOout.4 -FGgen.4 -a -Ctwo
5. optimir -P3 -p500 -A1000 -FOout.5 -FGgen.5 -rgen -Cuniform
6. optimir -P3 -p500 -A1000 -FOout.6 -FGgen.6 -sfitprop -m0.1

NR	para1	para2	FoM	Evals
1	-30.87	87.87	7.998	955
2	-86.3	-0.35	9.070	2988
3	-30.92	87.72	7.998	1551169
4	-84.99	-0.04	9.146	1000
5	-87.50	-0.36	8.682	931506
6	-85.15	-0.38	9.166	2890

Table 1: Results of different optimisations

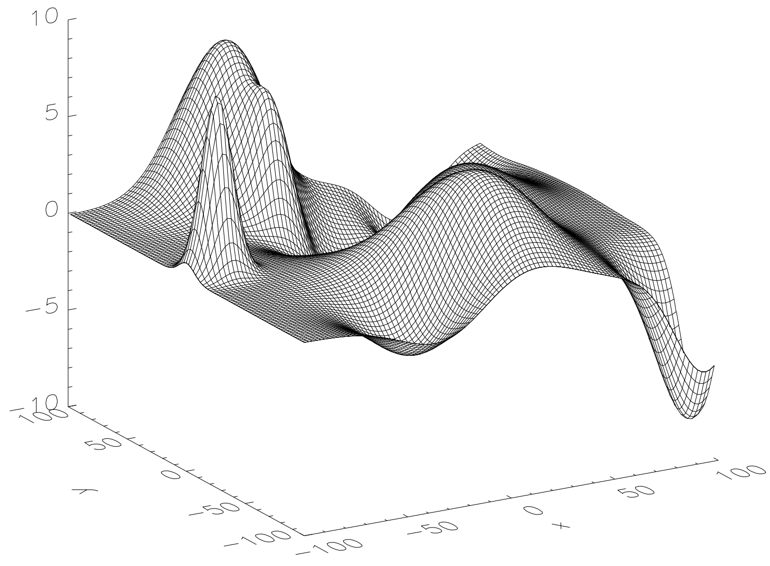


Fig2: **Graph of the function used in fm.c**

IV.2. ASA

ASA is a general Simulated Annealing program [ASA,93], it was developed in 1987 and is since under steady maintenance and improvement. I haven't changed anything in the main code of the program (asa.c), but restricted my changes to the file 'user.c'. For gaining more information about the ASA part, which contains all the important SA routines, I refer to the manual, which is included in the ASA-package.
I changed the cost-function procedure in the way that it behaves like the fitness-functions written for Optimir. ASA calls a extern program which contains the cost-function. This function reads the data from a file and writes the results in another file. The file names of these two files are passes to the cost-function as command line parameters. This is absolutely the same procedure than in the Optimir program, therefore every fitness function written for Optimir can as well be used as cost function by ASA.
To run the optimization you have to type:

asa_run -opt1 -opt2 ...

The command line options are just and only to specify some file names. The options for the optimization itself are stored in a separate file, which is read by the ASA module. The meaning and usage of the program options are described in the ASA documentation and won't be repeated here.

Options: The command line options for ASA are exclusively to specify file names. You can't influence the optimization procedure with them. The **-h** option will print a short help screen which explains the usage of the parameters. The names in the brackets '( )' are the default values.
The **-O<name>** option will change the name of the output file where the results are written to, default is '`asa_out`'.
-L<name> is responsible for the name of the limit file, the default: '`minmax.dat`'.
With the **-P<name>** option you are able to set a alternative name for the option file ASA is using. This is the file where all the options are set to influence the annealing process, the place for tuning the algorithm and adapt it to your problem. The documentation will give you the needed information to select the parameters, fitting to your specific problem.
The options **-E*** are responsible for changing the default names of the files in touch with the cost function. **-EF<name>** changes the name of the cost function program itself. You can specify any executable program here.
The **-EI<name>** option is the name of the file where ASA writes the parameters to analyse. The default is '`fm.in`'.
ASA assumes the returned cost value in the file '`fm.out`' or in a file you specified with the option **-EO<name>**

For more information about 'Adaptive Simulated Annealing' refer to: [ASAPAP].

V. RESULTS

The goal of this work was to prepare a set of Global Optimization tools which generally can be applied to the optimization of any optical system. For the calculation of the optical system performances we use Shadow. Therefore the programs developed should be able to optimise any possible case than can be analysed with Shadow (see Appendix). In order to illustrate the use of the presented programs and also to test the efficiency of the methods, we analyse different particular cases. Notice that these are test cases, with the only purpose to illustrate and test the use of Optimir and ASA in connection with Shadow. In general any other optical system can be optimised, and the procedure, from the point of ASA and Optimir, are absolutely the same (like mentioned before, both programs treat Shadow as a Black Box). The test cases have been chosen with increasing complexity to be able to extrapolate the results to any generic optimization case with Shadow.
The presented calculations have been performed on a SUN Sparc 10 workstation using the SunOS operating system. The source codes of the optimization programs can also be compiled on a HP systems (and probably on most other systems too, but we didn't try), but a Shadow version running under HPUX is still not available. We, therefore, were bound to our SUN machine instead of using our faster HPs.
Most of the cases ran over the night and a practical maximum running time of about 2-3 days was imposed (weekends). The workstation run a maximum of two optimization processes simultaneously, but were sometimes used for other CPU intensive work too. That's why we can't make direct comparisons of the running time, instead one column in each table (called Evals) tells the number of calls to the Shadow routine.
Both optimization programs have a lot of different options and free parameters, we used them freely to test the different options and find out which suites best for our cases. We haven't had the time to make an exhaustive statistical investigation of the influence of all the single parameter on the optimization process (especially for ASA). The following results may help anyway by finding the right parameter for your optimization. The user manuals of the programs will give you an additional help here.

V.1. Toroidal Mirror

Mirrors are used to focus the x-ray beam in horizontal or/and vertical directions. When the plane in which the focalization is done contains the source and the focal point, the focalization is called *tangentia*l or done in the *tangential* plane. The plane perpendicular to the tangential plane is called *sagittal*. Therefore, a mirror focusing in both tangential and sagittal plane has to be doubly curved in two perpendicular directions. In the simplest case, these curvatures have circular sections (the shape is a portion of a torus), and the tangential and sagittal curvature radii R_t and R_s, respectively, can be calculated in function of the source-mirror p and mirror-focus q distances and incident angle θ by using the formulas:

$$\frac{1}{p} + \frac{1}{q} = \frac{2}{R_t \cos\theta} \qquad (1)$$

$$\frac{1}{p}+\frac{1}{q}=\frac{2\cos\theta}{R_s} \qquad (2)$$

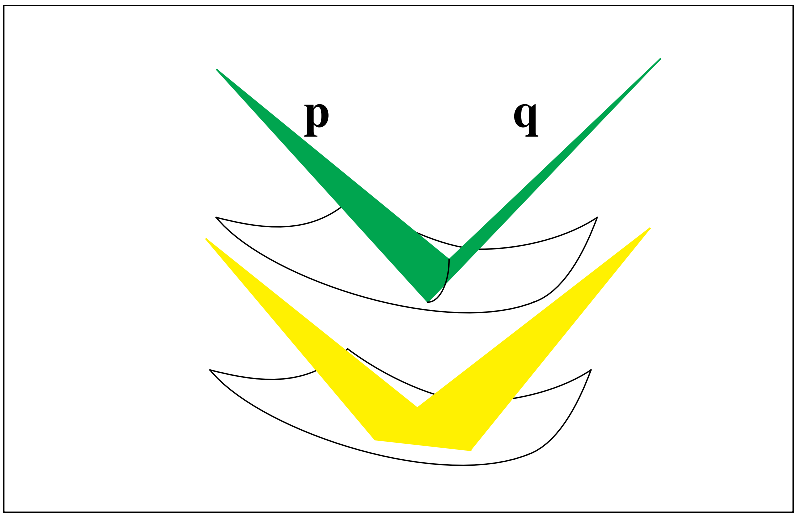


Fig IV.1: **A toroidal mirror focusing in the Sagittal plane (top) and in the tangential plane (bottom)**

In order to test our optimization program we ignored these formulas and left it to the programs to find the curvature radii (2 parameters) for a given configuration. The result must be close to the values calculated with the precedent formulas. With this, we have a test case to confirm if the optimization program works well or not.

The case we studied here was our first test with the Optimir program. As one easily can see, this is not a truly ***global*** optimization case, but it was though as a test to verify the program functions and to test the behaviour of the program, when confronted with its different options.

The case consist of a source of square shape (0.1mm x 0.1mm) with a divergence of 1 mrad in both horizontal and vertical planes. The mirror is placed at 10 cm from the source plane and the image plane is at 10 cm from the mirror. The incident angle on the mirror is 89 degrees (1 degree of grazing angle). With these values the theoretical R_s and R_t values are 0.175 and 573.0 cm respectively.

We run the GA program searching the maximum value of a figure of merit (FM) in the parameter space R_s [0.01,10] and R_t [0,1000]. The FM is computed looking at the image plane in the following steps:

- Calculate the Full Width at Half Maximum (fwhm) values of the histograms representing the horizontal H_x and vertical H_z projections of the image itself.
- Calculate the number of rays inside the rectangle H_x* H_y=I. This is an estimate of the intensity in the centre of the image.
- Evaluate the FM as intensity per unit of area in the centre of the image, or in other words, FM=I / (H_x*H_y).

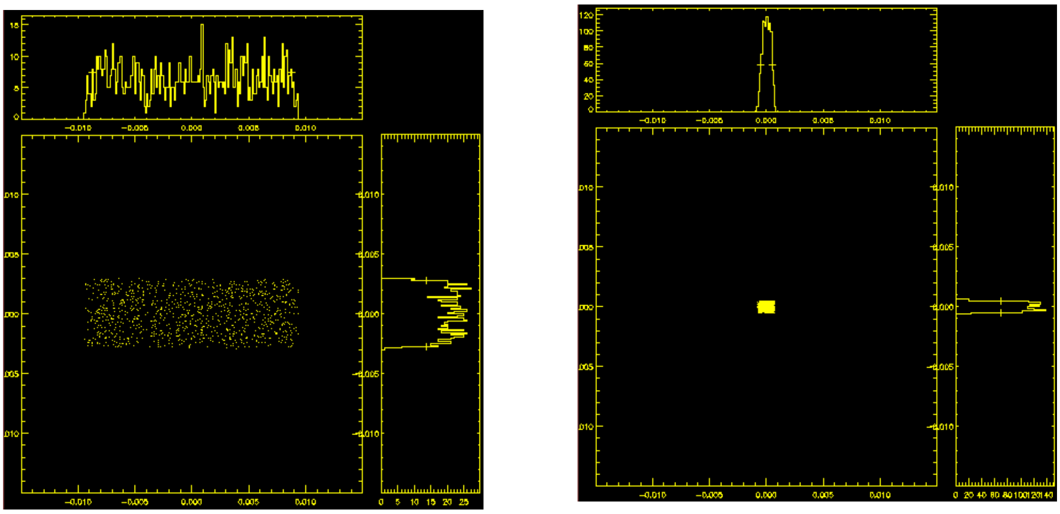


Fig IV.2: **Image produced by the toroidal mirror to be optimized for two cases. Left panel: with R_s= 2 cm and R_t = 800 cm. Right panel: with the optimum R_s=0.17 and R_t=573 cm. he image dimensions are also cm. Note the horizontal and vertical histograms used for calculating the FM value**

This FM increases when the spot is smaller and more intense, so it is a good evaluation function for optimizing the focal properties of the toroidal mirror
.

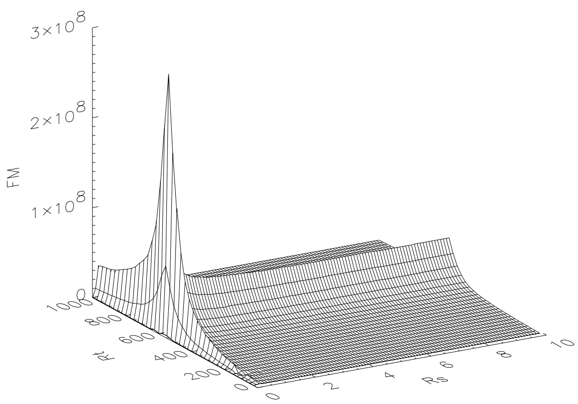


Fig. IV.3: **Figure of Merit for the Toroidal mirror as a function of the tangential and sagittal radii.**

We found out that the number of members in a population is crucial for a good optimization (which, of course, is not very surprising). Most runs were done with just -P2 * -p40 = 80 chromosomes in the whole optimization and allowing only 40 members in each population. This is much to less! The results gained with 100 or even more chromosomes in a population were extremely better. It seems, that a higher number of members in one population improves the results much more than the same amount distributed on more different populations, or in other words: -P1 -p100 yield normally better results than -P5 -p20 (As a consequence, we made most

future runs with one or two populations).
Also the selection of the crossover procedure was very important. While the uniform (-Cuniform) procedure gave even worse results, the choice of the two-point selection (-Ctwo) improved extremely the results.
The increase of the mutation rate didn't yield any improvement. Though we have used it successfully in later runs with other problems, we haven't got any better results in this special case (especially the mutation rate depends strongly on the problem to analyse and other circumstances).

V.2. Sagittally focusing crystal

A crystal is a diffracting element used in the beamlines to monochromatize the white or polychromatic x-ray beam. If we think in the Bragg law, which relates the diffracted wavelength with the crystal interplanar distance and the Bragg angle: $\lambda= 2d \sin \theta_B$, we can see that if we change the incident angle on the crystal (therefore, the Bragg angle θ_B) the outcoming wavelength is changed. Normally the crystal is rotated to scan the wavelength, and a second parallel crystal is used to make the outcoming beam parallel to the incident one. This is illustrated in figure IV.4.

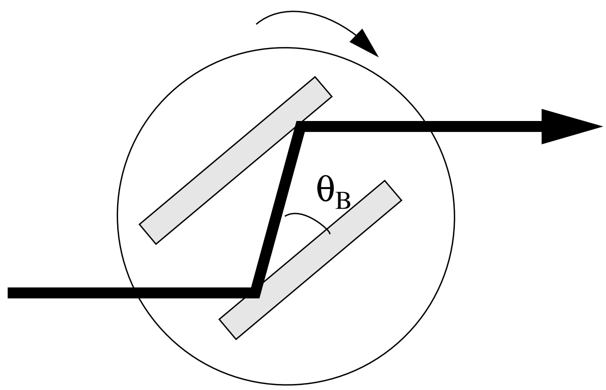


***Fig IV.4*:** **Typical double crystal x-ray monochromator mounting.**

In some cases, it is useful to focus the beam at the same time that is monochromatized. This is generally done by elastically bending one of the crystal. Sagittal focusing is generally done by curving the second crystal with a sagittal radius R_s, calculated with the formula showed before, where now the incident angle $\theta= 90 - \theta_B$.
The sagittal bending of the second crystal produces a significant drop of the total transmitted intensity. This is done because a ray which arrives on the first crystal with an angle θ, arrives on the second one with an angle $\theta + \Delta\theta$, deviated a quantity $\Delta\theta$ from the angle on the first crystal because of the crystal curvature. Therefore the ray will in general not be transmitted because the energy corresponding to this new angle does not correspond to the energy selected by the first crystal. Therefore a good matching between the crystals is essential to improve the transmitted

intensity. This can be done by changing the position of the crystal or accepting a given horizontal divergence of the beam. If the distance source crystal is p, and q is the crystal image distance, we may change p and q but keeping fixed the total length L = p + q. The essential parameter is the ratio M = q / p, which is called *magnification factor*, because is the factor to which the source size is amplified when images on the focal position. Notice that changing M, we must change the curvature radius R_s to keep the focus in the right position (i.e. to verify eqs [2])

In this example we try to optimise the beam horizontal divergence ω and the magnification factor M, a total of two parameters, in order to get the maximum of transmitted light. The source considered is an ESRF bending magnet from where the horizontal divergence is one of the free parameters. Notice that in all the cases the same intensity of the beam is spread in the variable horizontal divergence. The optical system is a silicon crystal with diffracting planes 111. The crystal is set to diffract photons at 8 keV (therefore θ_B= 14.31 degrees) and has dimensions of 20x20 cm. The source image distance is L = p + q = 13 m, and the magnification factor M is the second parameter to optimise. We do not know a priori the optimum values for the divergence ω and M, and theoretical work in similar systems (but not the same) predicts an optimum transmission for M values around 0.33 [Sparks,80]. The FM value is calculated computing the beam intensity at the image position. In Shadow, each ray carries its own intensity, which is initialized to one at the source position. Therefore the total intensity of the beam is just the sum over the number of rays of the ray's intensities.

Our best value from the optimization correspond to ω=5.67 mrad and M=0.19. Other values are shown in table 1. Notice that we reach the same best value with both GA and SA methods. In the case of SA more runs were needed. This is not a general rule, in many cases SA preforms much better than GA, it depends on the problem (see: "GA and VFSR: A comparison" [ASAPAP]).
The FM dependence for this system has been calculated by gridding the parameter space and it is plotted in Fig IV.5.

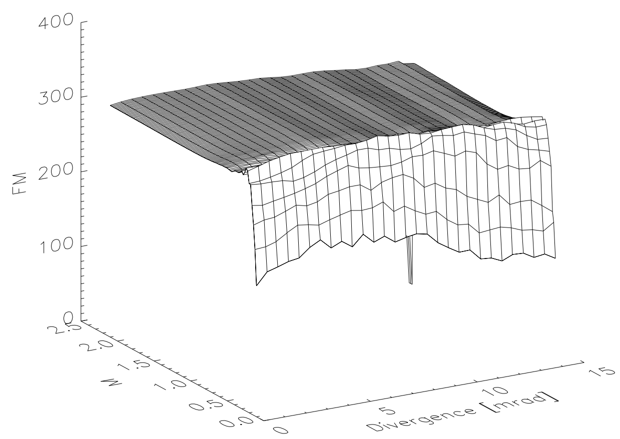


Fig IV.5: **Figure of Merit for the transmitted intensity in the crystal case, in function of the beam divergence and magnification factor. From this graphic, a maximum value is found at ω=6 mrad and M=0.2, giving a FM=339.8.**

NR	M	ω(mrad)	FM	Evals	Method
1	0.19	9.35	337.70	1000	Optimir
2	0.21	6.00	340.01	1000	
3	0.19	5.59	338.95	1000	
4	0.19	5.67	342.398	5644	ASA

Table 2: Results of magnification factor (M) and horizontal beam divergence (ω) obtained with the Optimir and ASA codes.

V.3. Cylindrical adaptive mirror

This was our first attempt to run our global optimization programs with more than two parameters. We try to optimise the profile of a mirror, which is defined by 11 points along the curved directions. The values or heights of these points are left free and is the responsibility of the GA program to find the best or at least good values for focusing the beam. In other words the global optimization program has to adapt the mirror profile to obtain the more intense and smaller focal image. A good solution would be a cylindrical shape with the heights corresponding to a circumference arc of tangential radius R_t=4297.2 m.

The source is a point with both horizontal and vertical divergences of 100 µrad. A mirror is placed at 30 m from the source and the image plane is set at 10 m from the mirror. The incident angle is 89.8 degrees. Mirror dimensions are 60 cm in the beam direction and 0.2 cm in the transversal direction. The mirror is focusing in the vertical (tangential) plane and the curvature is determined by the profile generated by the global optimization program. The FM has been defined as the intensity in a width equal to the fwhm (= full width at half maximum) divided by this fwhm in the vertical plane (focusing plane).

We started to analyse this system with the GA method, but didn't get any good results with this setup. We made many runs with various options and a chromosome length of 176 bits (i.e. 11 bits per parameter).

The variation interval for each parameter was kept quite large [0.0, 0.0015] so the parameters space is quite dense. We did not reach to even approximate the good parameters with the performed runs. We think that in this case, the number of population members has to be increased to cover better the parameters space, and in this case the running time increases to a impractical length.

At this point we realize that with so many parameters is not possible to obtain good results in a reasonable amount of time, starting from a wide range of parameters. We thought the possibility of giving some useful information to the GA program. In other words we try to help the program to get a reasonable good solution and then let him improve that solution. This can be done by giving a set of starting values. This means some few members of the chromosome pool are put there artificially, and then let the system evolve. Therefore, the best value, will at least be as good as the starting parameters, and this could be improved. In the real world, the starting parameter means that we know a useful configuration of the mirror (or, in general, of the optical system)

and we try to improve it. Even if we do not arrive to the “best” solution, a good solution (improving the existing one) may be enough for our purposes. This is often the case of the optical system developer.
Starting values in a GA procedure are a very critical and dangerous thing. The principle of GA is a slowly evolution from many random chromosomes. By giving starting parameters you set a ‘Superhero’ into a population of normally weak (in the beginning) ‘creatures’. The population will after short converge to a population of ‘Superheros’, which almost ever are chosen for reproduction. Therefore the weaker chromosomes may not be able to evolve, and some of their possibly good genes will die with those chromosomes, when they are replaced by the better ones, crossed from the ‘Superhero’ chromosomes. The chromosomes won’t have the change to search the whole function plain, but the ‘Superheros’ will force the search in their direction and this direction hasn’t to be the global one. You may become better results, but then you often could have reached this solution also with a standard optimization.
In any case, we thought it was worth to check if it works anyway, and we decided to implement this option in our Optimir program. After few runs with this new option we confirm that the system improved the FM value a little bit, but not enough to call it a success.
In the case of an adaptive mirror, the spherical solution can be improved when the geometrical aberrations are important, and this depends on the distances and angles. In such a case, an elliptical profile of the mirror improves substantially the focal dimensions. In our particular configuration, the geometrical aberrations are not important so we decide to change the set up a little bit moving the origins of the light source and the destination plane. This is the case analysed in the next chapter and generally called “Elliptical” adaptive mirror.

V.4. Elliptical adaptive mirror

The system considered here is very similar to the one in the previous chapter, but with the following parameters: Source-mirror distance p=4500 cm, mirror-image distance q=150 cm, incident angle θ=89.774 degrees, mirror dimensions: 60 cm along the beam direction and 2 cm in the transversal direction. With this configuration the spot image produced by the elliptical mirror is better than the cylindrical one, as shown in Fig IV.

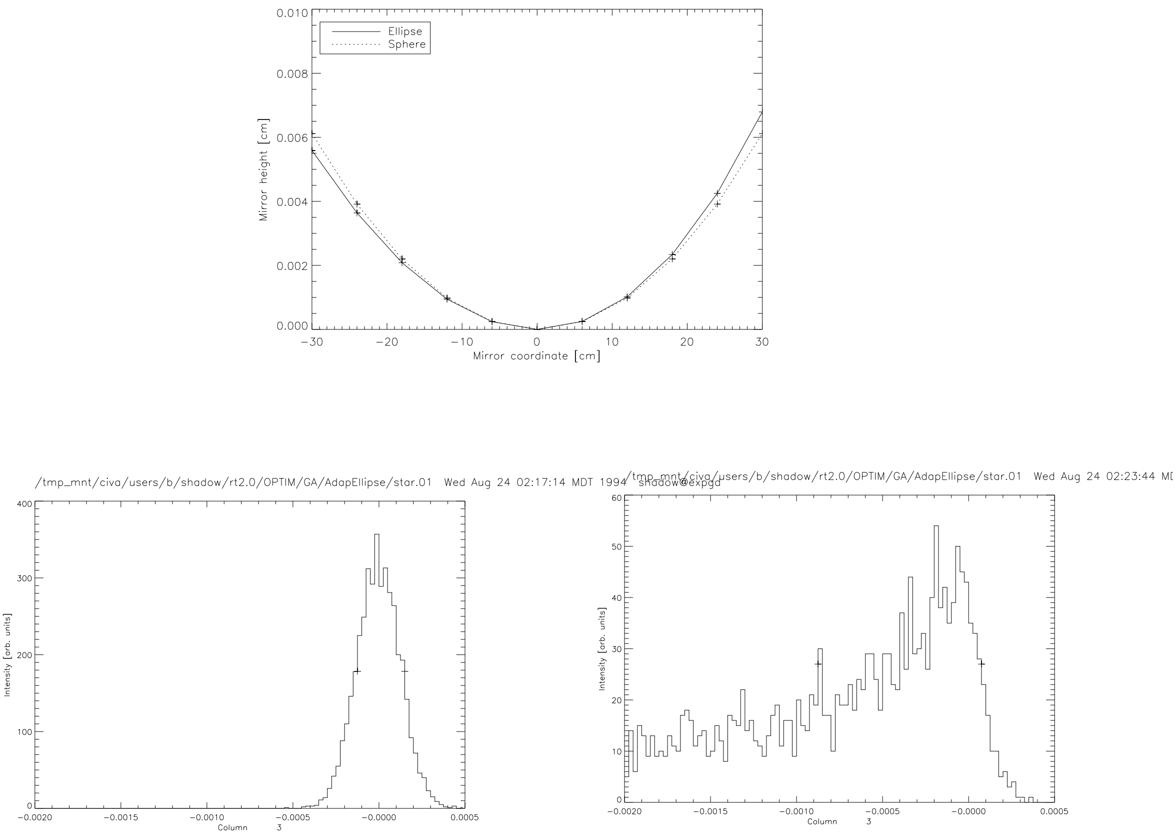


***Fig IV.6*: Theoretically calculated mirror profiles with cylindrical and elliptical shapes (top panel) and the intensity distribution at the image plane (bottom panel) for both elliptical (left) and cylindrical (right) cases. FM values are 9.91 10^6 for the elliptical and 8.33 10^5 for the cylindrical one.**

This setup is in principle the same than that in chapter IV.3, but some important parameters of the setup (different distances from mirror to source and destination and similar things) were changed, The results in this altered case were much more promising than those in the case before.
We managed to get rather good results, but never came to a result as good as the elliptical case with a running time in our limits (the case shown in table 3 was running for about one week).
We focused our attention on the GA without starting parameters and on SA. We reduced the range of each parameter to reduce the extension of the parameter space and to shorten the duration of the different runs.
(REMARK: The last parameter is not influencing the optimisation process, because this part of the mirror is not illuminated by the light-source. This, for example, explains the strong deflection of the last parameter in Fig IV.7.)

NR	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	FM	Evals	Method
1	5.2 E-3	3.9 E-3	2.2 E-3	9.5 E-4	2.4 E-4	5E-7	2.4 E-4	9.7 E-4	2.2 E-3	4.1 E-3	6. E-3	2337300.0	3000	Optimir
2	5.1 E-3	5.2 E-3	3.4 E-3	1.2 E-3	2.4 E-4	1.6 E-6	4.8 E-4	1.8 E-3	1.5 E-3	1.9 E-3	4.1 E-3	854489.0	3000	
3	5.3 E-3	3.1 E-3	2 E-3	9.5 E-4	2.4 E-4	1.9 E-6	1.7 E-4	6.7 E-4	2.1 E-3	3.8 E-3	4.6 E-3	1781120.0	7000	ASA
4	4.2 E-3	3.2 E-3	4.6 E-3	8 E-4	2.5 E-4	3.4 E-6	1.8 E-4	9.5 E-4	2.3 E-3	3.8 E-3	4 E-3	1750910.0	10000	
5	5.7 E-3	4.5 E-3	2.4 E-3	1.7 E-4	3.7 E-5	7.4 E-6	1.5 E-4	3.7 E-4	1.6 E-3	5.2 E-3	7.6 E-3	428473.0	5000	
6	6.1 E-3	1.8 E-3	1.2 E-3	2.6 E-4	1.1 E-4	7.5 E-6	2.3 E-5	6.7 E-4	4.1 E-3	1.6 E-3	4.7 E-3	377909.0	5000	

Table 3: Results of optimization with Optimir and ASA by using different options

The following options were used for the optimizations:

Optimir: 2.) -P1 -p250 -a
3.) -P1 -p250 -a

ASA: 4.) CPS=1 TRS=0.01
5.) CPS=0.5 TRS=0.01
6.) CPS=0.5 TRS=0.001
7.) CPS=1.5 TRS=0.01

CPS means Cost_Parameter_Scale and TRS stands for Temperature_Ratio_Scale. The meaning of these parameters for the optimization procedure are explained in the ASA manual. The meaning of the Optimir parameters are explained in section III.1.

In case 1, we gave start values to see if it improves the search when some approximate values are known to the function. Therefore one of the chromosomes got the already good fitness of FM=841000. All other cases ran with randomly chosen starting values, which, of course, is the normal way (you don't need a time consuming global optimization if you already know were the extrema lies).
In this case, the GA procedure performed better than the SA routines. An adjustment of the ASA parameters could maybe yield better results, but up to now we haven't got the time to test ***all*** the

various parameters of ASA.

To give an impression about the change of the FM value during a single run of Optimir, the next table shows you the results of the run after different generations. The options for this run were:

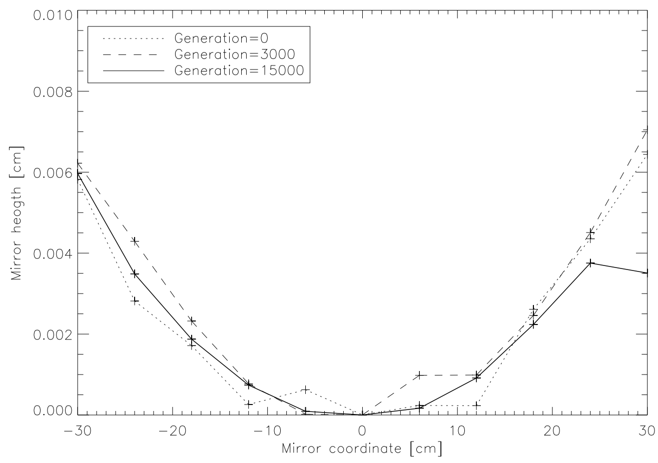
optimir -P1 -p500 -a.

The first row contains the fitness of the best ***randomly*** chosen chromosome in the starting population.

p1	p2	p3	p4	p5	p6	p7
5.8 E-3	2.8 E-3	1.7 E-3	2.6 E-4	6.2 E-4	4.2 E-6	2.4 E-4
7.5 E-3	1.4 E-3	3.5 E-3	2.3 E-4	1.2 E-4	3 E-7	2.2 E-4
6.2 E-3	4.2 E-3	2.3 e-3	7.7 E-4	1.9 E-5	4.6 E-6	9.5 E-4
6.2 E-3	4.3 E-3	2.3 E-3	7.7 E-4	1.9 E-5	4.6 E-6	9.8 E-4
3.2 E-3	2.9 E-3	4.5 E-3	6.7 E-4	1.9 E-4	1.7 E-6	2.2 E-4
6.3 E-3	3.5 E-3	1.9 E-3	7.4 E-4	1.4 E-4	6.0 E-6	1.6 E-4
6.3 E-3	3.5 E-3	1.9 E-3	7.4 E-4	9.2 E-5	1.0 E-6	1.7 E-4

p8	p9	p10	p11	FM	Gener.
2.3 E-4	2.6 E-3	4.4 E-3	6.4 E-3	62437.4	0
8.5 E-4	2.8 E-3	5.5 E-3	6.6 E-3	134912.0	50
9.0 E-5	1.7 E-3	2.5 E-3	4.3 E-3	179061.0	1000
9.9 E-4	2.5 E-3	4.5 E-3	7 E-3	381435.0	3000
9.9 E-4	2.2 E-3	3.8 E-3	7.0 E-3	890179.0	5000
9.1 E-4	2.2 E-3	3.8 E-3	3.5 E-3	1847360.0	10000
9.1 E-4	2.2E-3	3.8 E-3	3.5 E-3	2116120.0	15000

Table 4: Results of one Optimir-optimization after different generations



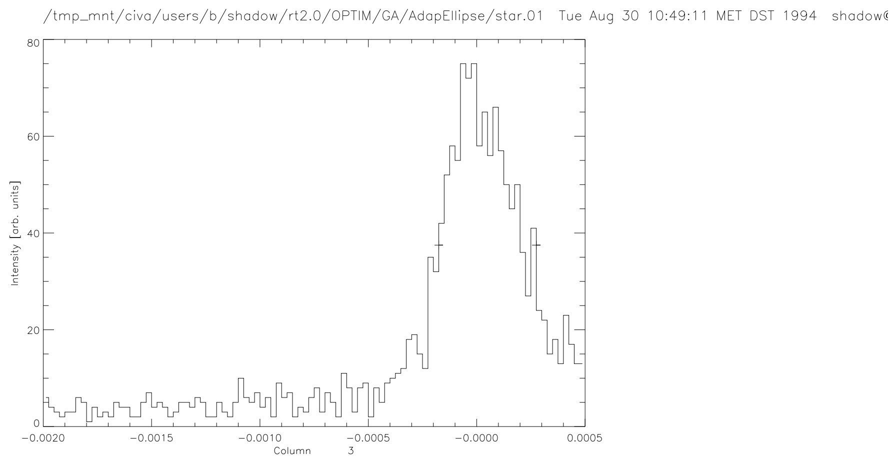


***Fig IV.7*: ** Some mirror profiles of table 4 after different gereations (top) and the intensity distribution of the best chromosome after 15000 generations(bottom).

VI. CONCLUSION

VI.1. Have we reached our aims ?

After the work put into the global optimization methods we now, after about 6 month, come to the following conclusions:

a. It is worth to put more interest in these methods, because they are a interesting way of optimization. Even if there are problems with running time and some of the gained results doesn't seem very exact, one must consider that in many cases these methods are the only ones that are able to find at least a solution *near* the final global minima.

b. It's not worth to following this path without accelerating the speed of the cost function, which in our case is 'Shadow'. One evaluation call needs up to 1 minute to return and with an average amount of 5000 - 10000 calls for a single optimization, which are needed for a reliable result, the optimization will take 3 to 7 days to finish. This long time is mostly not acceptable. (Some ideas to solve this problem are given in the second part of the conclusion.)

Although, we have not reached a level of some "buy-and-be-happy" products which could just be handed to the users at the beamline, we at least have designed two *working* products which can help in the optimization of beamline optics. If the user has the time or enough raw machine power than this programs may be very useful. And not every problem you may encounter has to be one were 100 variables have to be optimised at once, but there are lots of problems with a smaller amount of variables were the local extrema could give you a hard time while disturbing your optimization attempts. And all runs we did with fewer than eleven parameters yield good results and even the results gained with eleven parameter were not to bad if enough time was gained to the process (see chapter IV.4, Table: 2 & 3).
The work with global optimization has not reached its dead end here, but is now at a crossing were different ways are leading in different directions. Here are some of the possible ways:

VI.2. Where to go from here ?

There are different ways to go from here. It is all but a question of time and money you want to spend on the project. The question is not "Is it possible ?" but "How important is it ?" (and perhaps also: "What's the budget ?").

The most obvious way is of course to increase the speed of 'Shadow'. We have used Shadow from an IDL script [IDL,93]. We went this, on the first glance a bit strange looking way, because IDL is a nice and comfortable interface to communicate with Shadow. It's simple to prepare the input data for Shadow and some already existing postprocessors made it easy to calculate the fitness and cost values from the raw data returned by Shadow. It's quite simple with IDL to change the setup of a optimization case or write a totally new one, it normally takes between 15 min. and 1 h to do so. The problem is the execution time ! IDL is an interpreter, compiling every function before using it. It would be much faster to do these 'managing' jobs with a C or even FORTRAN (gee, it's still alive) program, but on the other hand also much more complicated

(with a large amount of programming time). If we were doing the optimization seriously, it would, without question, be worth to replace at least the postprocessing routines, we now doing with IDL, by C programs. We haven't done this now, because our main target was to test a wide field of different functions to see if the GA/SA approaches are worth to invest more time and work for a deeper going research. In this phase of our research it would be a waste of (programming) time to write a new program for each new setup and it would keep us away from the really important work.
Another thing is to speed up the 'batch' running mode of Shadow. For our evaluations we normal need only a few of all the values Shadow calculates. The additional overhead is only consuming computational time and has no sense for our calculations. One way could be to extract the important functions of Shadow and place them in a C program to speed up the evaluation. This could also reduce the extensive file handling because Shadow is reading and writing a lot of files which is always time consuming.
This approach, however, results in losing the generality of the optimization procedures. If you go this particular way you probably have to write a new ray-tracing algorithm for every new problem. This is just what we tried to avoid, because it needs a lot of programming time, a special knowledge of the problem itself and every change in the experimental setup will result in the complete change of the program code.

Apart from Shadow, we have to look for other solutions: One is to buy faster machines or even parallel processor machines, which really could speed up the programmes enormously. At least GA are perfectly suitable for parallel processing. The internal speed could be increased by 50% and more [Schoeneburg,94]. Similar it is with ASA. Just now, while I'm writing this report, efforts are done in the US to implement ASA on a parallel computer. I actually don't know how much this will boost up the program and if it's worth doing, but otherwise, not even the US FORCE is wasting money senseless (I may be wrong with this statement anyway).
Some minor time can be saved by changing the structure of the optimization. Both programs (Optimir and ASA) where originally written to have their evaluation functions built into the code itself. The disadvantage is again that you have to change the code itself and recompile the program every time you change the evaluation function.
One solution we tried was to pass the data instead by normal files by a pipe. This works, of course, only on machines with an operating system that support pipes (forget about MSDOS). Though this idea won't help anything in saving time from the really time consuming ray-tracing procedures we are using, one gets rid of all the file handling stuff, which also takes its time. The saving of the overall time is about 1-3%, which isn't really sufficient to justify its more complicated implementation. But in cases where your evaluation function is much faster than Shadow, this percentage may jump up to 20 or even more percent, because in those cases the file handling may become the bottleneck of the whole optimization procedure.
And last but not least, there is a very promising idea suggested in a couple of articles about this subject: Both methods, GA and SA, are designed to search for ***global*** extrema and both are capable of doing their job well. **But**: (and here their advantages turn into a disadvantages!) They are both very lousy in finding the exact value in the end of the optimization procedure. Therefore, it looks like a good idea to switch from the GA/SA algorithm to one of the much faster and more precise standard algorithms (like Newton iteration or Gradient method [NUMREC,92]), after the region is found were the global extrema lies in. The problem here may be to decide when the right time for the switching has come.

There may be more solutions to increase the performance, but those shown here are, in my eyes, the most important and most practical.

VII. APPENDIX

VII.1. Workschedule

I started my work at the ESRF like probably everyone starting a new work. The first 2-3 weeks I didn't much more than playing around with the new toys I got to my proposal. I first tried to figure out the difference in the single machines, I could reach over the network. Some of the programs I had to use in the future(e.g. Shadow or IDL) are only running on one of the two SUN machines which are much slower and often more overcrowded than the HPs. Other programs then were only licensed for the HPs and not useable from the SUN stations.
After short, my basic knowledge about the UNIX-OS was returning and the times I typed '`DIR`' instead of '`LS`' got fewer and fewer.
One day I spend completely doing nothing else than using 'MOSAIC' to implore the 'INTERNET' which is an invaluable source of knowledge, if you know how to find it (the knowledge, not the INTERNET).
The INTERNET is a worldwide network started in the 60' by the US DEPARTMENT OF DEFENCE as a reliable communication source during a possible war. It first connected only several important sites of defence and science, but after a short while the scientists using the net discovered its invaluable worth as a communication and information source and so, more and more sites with less military importance were added with the time.
Today nearly every university and scientific organisation is connected and reachable by this world wide network. More and more tools were designed to make it easier for the different people to use the resources of the net. One of the newest development, is a program designed at the NCSA in the US called 'MOSAIC'. It is a tool which allows you to travel easily around the world by clicking on coloured words or pictograms which immediately rout you to your desired destination.
I used this facility to find as most as possible information about the global optimization problem that I was supposed to work with. I 'surfed' the net and found a lot of interesting sites that were already handling with similar problems (you can find some of the most interesting one in the references).
The idea was from the beginning, to get some existing programs an adapt them to our own problem. This was thought to avoid the typical beginner mistakes, to shorten the programming time and to make use of the reliable already tested functions of an existing program. After I had collected in this manner about 10 different programs of Genetic Algorithms(GA) and Simulated Annealing(SA), I chose two programs I considered the best to work with: PGA as the one related to GA methods and ASA as the SA approach.
In the same period I collected as much as possible information about the subjects itself. I haven't worked with either of the two methods before and my knowledge was in the beginning just superficial. Every new article led to at least four or five new ones that were recommended or referred to by the respective author. So after about one week I had a remarkable collection of articles concerning these matters.
I realised that it would be more efficient to gain some knowledge about the background of these two methods. Especially for the SA methods my knowledge wasn't sufficient in the beginning. The first time, I even had to search in the darkest spots of my mind for the physical background I should have learned in school and now I needed to understand the principles of the SA idea, but in the end the study of some physical oriented books about annealing and probability proved

worthier. With GA the problem was less difficult, because in each of the general books about GA, I found a short introduction into genetics. Though I may not be able yet to breed some “woolly, egg-laying, milk-giving highland sows”, I at least got enough knowledge to see what’s the idea behind all that stuff. The programmers kept really as close as possible at their natural model and the most astonish thing is, that the most rules valid for the natural genetic, can be applied to the artificial one as well.

About two weeks later I decided that now my knowledge was sufficient to make the first grave changes in the PGA program (To get not mixed up with the two different methods we decided to test one method after the other and put for now the SA on the wait-stack). I had after several test runs and tracings of the original program code a pretty good idea what was going on into the program. The first and most important change was the rebuilding of the fitness function. While PGA is using all its evaluation functions as built in functions, Optimir is using an extern program to which contains the function (see chapter 4 for more information). While this step was relatively easy to do, it was more tricky to boost the performance of this new technic. PGA with its built in fitness function needn’t to care where and how often it calls its fitness function, while Optimir has its most critical part just at this point. Every call of the function means an external program call which always is time consuming. I tried therefore to reduce the calls to a minimum by evaluating at many chromosomes at once as possible. To do this I used chained lists and dynamic memory management to collect as many chromosomes as possible before calling the fitness function.

PGA was designed as an educational program at the university of Edinburgh, therefore its main task is not performance but a clear, obvious structure (this is why I chose this program as starting point for my own work). Yet, in Optimir performance is the essential and main point. I tried of course to keep it’s structure as clear as possible but sometimes I had to decide between a clear structure or a faster code in such cases I always decided for the faster implementation, trying to compensate it by generous commentaries in the source code.

I worked about two month exclusively on this project, changing more and more of the original program to fit it as good as possible to our problems. New parameters were installed to influence the flow of the program, a lot of file handle oriented routines were designed for input and output of information and for the convenient control of the program.

The alteration of the program was attendant by a steady testing of the new routines with the new and old parameters. This enabled me to recognise mistakes as early as possible. After the main work was done (smaller changes occurred until the end of my intership) I started to shift my interest more and more to the SA methods.

Here my work was more easy because I had a professional SA-package to my disposal. The ASA program is a well maintained and wide used piece of software, which has his own ‘Fan-club’. The program was developed from some earlier versions (VFSA) and the developers were very concerned about the needs and wishes of the users (You can even subscribe to a mail group, to get always the newest news about the ASA project). Because I didn’t want to renounce on this support I decided not to touch the intern functions of the SA routines. The only changes were done into the file ‘`users.c`’ which is intended to be changed by the user. This gives the opportunity to use every new release without changing the program at all (This works of course only if the authors of ASA don’t violate their own concept in the future). The only changes in the code I made were things related to the file handling and the implementation of an extern cost function. After this minor changes of the code a lot of different problems were tested with the program. The main problem of the SA method is to figure out which parameters will yield you

the best result. Every problem may need a different adjustment of the parameters to find the optimal solution. This is probably the worst disadvantage of SA, because you may waste a lot of time searching for the right values for the annealing process.
A closer explanation of the two programs is given in chapter four, the principles and the structure of the methods itself is shown in chapter three.
The last month at the ESRF I used for writing this report and do some optimization with the two programs to find out which is the best to use in our case or if both are useful or neither. The results are summarised in chapter five and in chapter six, where chapter six is a general conclusion about my whole work at the ESRF.

VII.2. Shadow

Introduction

Shadow is a ray-tracing program specially optimised for the design of the synchrotron radiation beamline optics. It has been developed by F. Cerrina and others, at the University of Wisconsin. Shadow generates and traces a *beam* along the *optical system*. The beam is a collection of *rays* in a given point of the beamline which are stored in a disk file. The optical system is a collection of *optical elements (OE)* (mirrors, multilayers, slits, screens, etc.) placed in a sequential order. Each ray is an array of 18 (or 12) variables or *columns*. Each variable of column has an special physical meaning The first six defines the geometry: spatial coordinates (Col. 1,2,3 or x, y and z, respectively) and the direction of the ray (cols. 4,5,6, or x',y' and z', respectively). The rest of the columns defines the history of the ray traversing the optical system (electric vector for s polarization (cols. 7,8,9) and p polarization (cols. 16-18), flag for lost ray (10), wavelength (11) etc.).
The *source* is the beam at the starting point. It is generated by Shadow by sampling the spacial, angular, energy and other qualities of the synchrotron radiation sources (i.e, bending magnets, wigglers and undulators) into a finite number of rays, using a Monte Carlo method. At the source position the intensity of each ray (or better, its probability of observation) is set to 1. This intensity will decrease along the beamline because of the interaction of the ray with the optical elements. The source generated by Shadow samples linearly the real source, which allows to scale the intensity with the number of photons.
Shadow traces the source sequential thought each individual optical element of the optical system. Shadow solves the intercept of each ray at a given OE, calculates the output direction and the decrease in intensity. This decrease is calculated for each ray using a physical model (i.e. Fresnel equations for mirrors, Dynamical Theory of the Diffraction for perfect crystals, etc.)

The files generated by Shadow

Shadow generates a big amount of files. The most important ones are
`start.xx` an ASCII file with the namelist for the source or optical elements (`start.00` for the source, `start.01` for the first o.e, `start.02` for the second, and so on)
`end.xx` same of `start.xx` but with few parameters recalculated after the run
`begin.dat` binary file containing the beam at the source position
`mirr.xx` binary file containing the beam on each OE (i.e. `mirr.02` is the beam on the second o.e)
`star.xx` binary files with the beam at the image created by each OE The image of a given OE is the source for the following OE
`screen.xxyy` Shadow can (optionally) create the image file in a different points of the beamline, or screens. Screens are associated to a given OE and can allow to define absorption (filters) or apertures (slits or beam-stoppers). `xx` refers to the OE order and yy refers to the screen order (i.e. `screen.0204` means the fourth slit associated to the second OE. Fig VI.1 shows a schematic representation of the coordinate system Shadow uses.

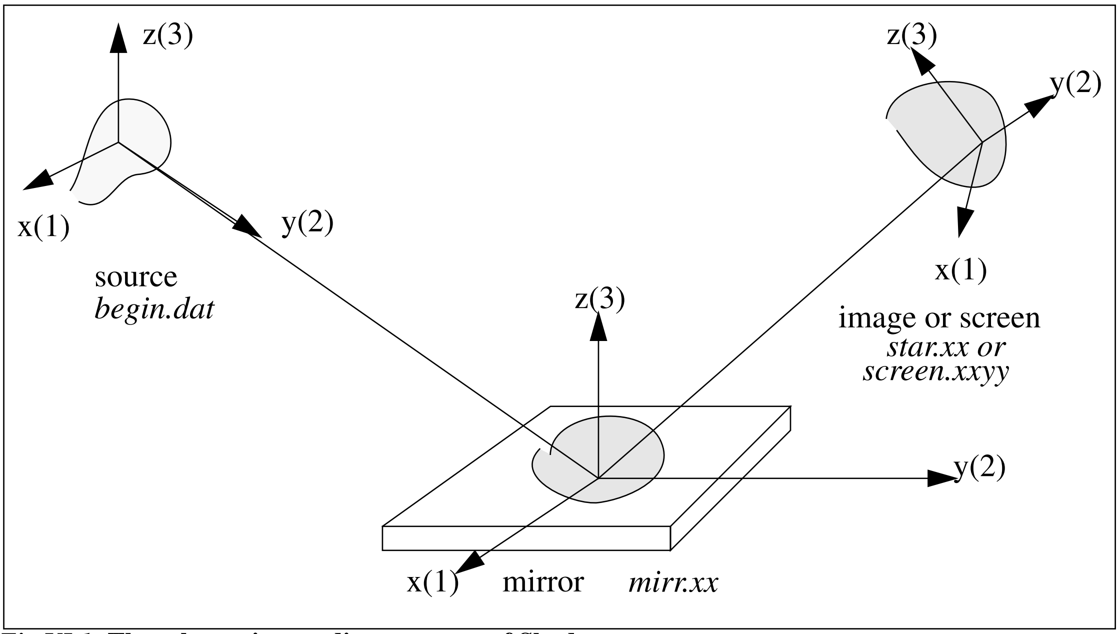


Fig VI.1: **The schematic coordinate system of Shadow**

Running Shadow interactively and in a batch loop

Basically Shadow is called using the commands `go source` for the source generation and `go trace` for the tracing of the optical system. The data can be input using a prompt or menu mode. Sometimes Shadow needs other files generated by pre-processors, in order to allow calculations of sophisticated cases (insertion devices, crystals, etc.). Some of them are shown in figure VI.2.

When one uses the prompt mode Shadow asks for each individual parameter. The prompt mode is recommended for a first run. In the menu mode, recommended after a first run, one has to load (`load source` or `load system`) modify (`select source` or `sel oe`), save (`save source` or `save sys`) and run (`exit`) the current system.
After a successful run of Shadow one has to analyse the results. For that purpose there is a set of utilities written in the IDL language, which allows visualization and histograms of the results. Some of the basic utilities are: `infosh`, `histo1` and `plotxy`.
In order to use Shadow in the Global Optimization (GO) programs a batch job must be prepared to automatize the changes in the system, the run of Shadow and the analysis of the results including the calculation of the Figure of Merit (FM) function. This batch job can be either a unix shell script or a program in any language. We have chosen to write the batch job in IDL, because we can use the available analysis facilities to calculate the Figure of Merit and only a very small programming effort is required for each new system. The basic procedure is the following: prepare and run Shadow for your system with some initial values. Decide the parameters to be optimised. These parameters correspond to some variables in the `start.xx` files and are passed from the GO program and substituted in the `start` files by using the `chparm` Shadow utility. Then run the source and/or the trace part of Shadow by doing a system call from IDL. Once the Shadow run is completed, the analysis is done to extract the suitable parameters (i.e. the spot

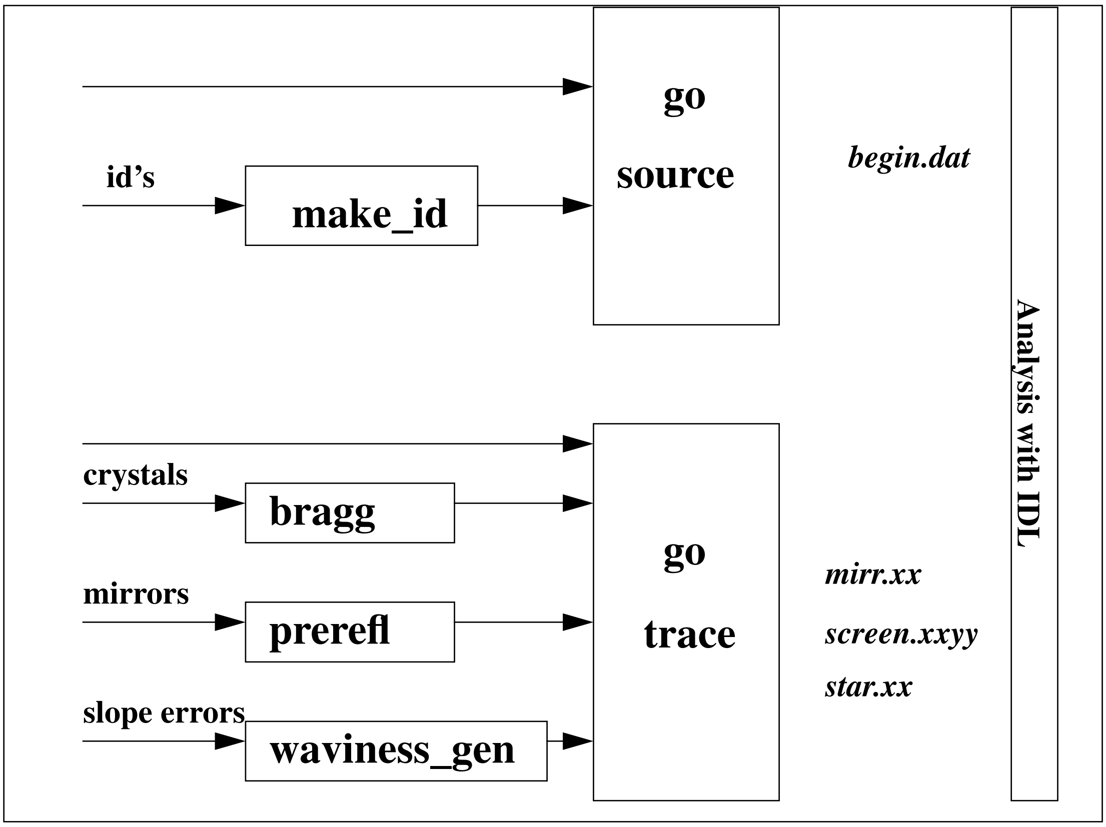


Fig VI.2: Data communication of Shadow with other programs via files.

dimensions, calculated by using the PLOTXY utility and fitting the resulting histograms with gaussian functions). Elaborated figure of merits can be calculated in function of several parameters calculated with the analysis facilities. Then the FM value is stored in the file to be read by the GO program.

Availability of Shadow

Shadow is available free of charge for non-profit research institutions. In order to get it, contact:

F. Cerrina [`cerrina@xraylith.wisc.edu`] or
C. Welnak [`cwelnak@xraylith.wisc.edu`]
at:
Center for x-ray Lithography
3731 Schneider Drive
Stoughton, WI 53589
USA.

References: [1]
Abstract
C. Welnak, G-J Chen and F. Cerrina.
Nucl. Instrum. and Meth. A. (1994) In press
[2]
Abstract
C. Welnak, P. Anderson, M. Khan, S. Singh and F. Cerrina,
Rev. Sci. Instrum. 63(1) (1992) 864

Further information can be retrieved from the mosaic servers of the CXrL (Center for X-ray Lithography) or the ESRF:

CXrL: http://ishtar.xraylith.wisc.edu/Home.html
ESRF: http://expga.esrf.fr:3600/computing/doc/theory/xray.html

VII.3. References

Books:

[Goldberg,89]
Genetic Algorithms in Search, Optimization & Machine Learning
David E. Goldberg
1989 Addison-Wessley
(The standard literature !)

[Aarts,89]
Simulated Annealing and Boltzmann Machines
E. Aarts & J. Korst
1989 John Wiley & Sons Ltd.

[Rawlins]
Foundations of GA
Editor: J.E.Rawlins
Morgan Kaufmann Puplishers, San Mateo, CA
(Collection of Papers about GA)

[Schoeneburg,94]
Genetische Algorithmen und Evolutionsstrategien
E. Schoeneburg
1994 Addison-Wessley

[NUMREC,92]
Numerical Recipes in C
Press, Teukolsky,Vetterling, Flannery
1992 Cambridge University Press

Journals:

[FAQ]
Hitch-Hiker's Guide to Evolutionary Computation
J. Heitkoetter
Retrivable: alife.santafe.edu - pub/USER-AREA/EC/FAQ
(exhausting collection of FAQs. Many pointers to progs & books)

[Metropolis,53]
Equation of state calculations by fast computing machines
Metropolis, A&N Rosenbluth, A&E Teller
Journal of Chemical Physics 21, 1087-1092

[Kirkpatrick,83]
Optimization by Simulated Annealing
Kirkpatrick, Gelatt & Vecchi
Science 220, 671-680

[ASAPAP]
Collection of papers related with the ASA program
PS files can be found in the same directory as the ASA program
(see [ASA,93])]

[Vanderbilt,84]
A Monte Carlo SA Approach to Optimization over Continuous Variables
Vanderbilt & Louie
Journal of computational Physics 56, pag 259-271

[Stein,92]
Real Artificial Life
R. M. Stein
BYTE magazine Jan. 1991

[Shafer,94]
Global Optimization in Optical Design
D. Schafer
Computers in Physics 8, 2, 1994, pag 188-195.

[Zeyher,94]
Optical packages look for global minima
Allen Zeyher
Computer in Physics 8, 2, 1994 pag 137-140
[Sparks,80]
Abstract
C.J. Sparks, B.S. Borie and J.B. Hastings
Nuclear Instruments and Methods 172, 1980

[INFO]
Leavelets about the ESRF in France
can be orderd from:
European Synchrotron Radiation Facility
38043 Grenoble Cedex 9 - France

Programs:

[Optimir,94]
OPTIMIR - general global optimisation tool
Ch. Böddicker
Retrievable from: expga.esrf.fr - /dist/SUN4/SUN_Optimir.tar.Z
or /dest/HP700/HP_Optimir.tar.Z (the source code is identical)

[PGA,93]
PGA - paralell genetic algorithm testbed
G.H.Ballinger & P.Ross
Retrievable from: alife.santafe.edu - /pub/USER-AREA/EC/GA/

[ASA,93]
Adaptive Simulated Annealing
Lester Ingber
Retrievable from: ftp.alumni.caltech.edu - /pub/ingber/]

[Bosin,93]
Simulated Annealing
Andrea Bosin
Retrievable from: itncpl.cineca.it/pub/physics/SIMAN/
(Warning: documentation in italian)

Questions & Contact:

Christian Böddicker
boeddi@ai-lab.fh-furtwangen.de

This account could be deactivated in about a year. So if you have major problems and can’t contact me by this address, try the following:

Manuel Sanchez del Rio
srio@esrf.fr

It may need awhile before you get a answer but we do our best.