Online Multi-objective Optimisation of MAX IV Electron Storage Ring

A genetic algorithm approach to optimising
lifetime and injection efficiency of an electron storage ring

Ali Kuzey Yanartas

A thesis presented for the degree of
Bachelor of Science

Supervisors:
Marco Apollonio and Sverker Werin

Project duration: 2 months
Division of Synchrotron Radiation
Department of Physics
Lund University
May 2024

Abstract

The aim of this bachelor thesis is to enhance the performance of one of the electron storage rings at MAX IV. The performance is primarily evaluated based on the lifetime and injection efficiency of the ring. Improving these variables will enable smoother machine operation by providing faster injections and reducing background radiation during injections. Consequently, this optimisation project is valuable in terms of the safety and operations of MAX IV. A genetic algorithm approach was used for the optimisation, and the selected algorithm was the Non-dominated Sorting Algorithm II. The algorithm controlled a family of sextupoles in the ring to improve the dynamic aperture and momentum acceptance. Theoretically, these improvements should extend the beam lifetime and elevate the injection efficiency. The algorithm was successful in finding superior sextupole configurations compared to the present conditions of the ring. However, due to the magnetic hysteresis, it was deemed challenging to reproduce these solutions. Despite the issues with hysteresis and, therefore, the reproducibility of solutions, it was demonstrated that the algorithm was able to recover the key performance variables of the lattice, particularly in a poorly set machine.

Acknowledgements

I would like to thank my supervisor, Marco Apollonio, for his consistent help and availability throughout the project. His passion and wisdom in the field of accelerator physics improved my interest and skills as a researcher. I want to thank my co-supervisor, Sverker Werin, for his valuable input and feedback throughout the project; his guidance was much appreciated. I would also like to thank David Olsson for his valuable insights on potential enhancements to the optimisation strategy. I would like to acknowledge the entire MAX IV accelerator development and operations team for their support and guidance on this project. This project would not be possible without their flexibility in providing machine time. I would like to express my gratitude to my family and partner for their patience and support throughout my thesis.

Contents

List of Key Acronyms

Linac	Linear Accelerator
R1	The 1.5 GeV storage ring at MAX IV
R3	The 3 GeV storage ring at MAX IV
SCo	Outer Corrector Sextupole
MOGA	Multi-objective Genetic Algorithm
NSGA-II	Non-dominated Sorting Genetic Algorithm II

1 Introduction

The accelerator complex of MAX IV includes a linear accelerator (Linac), a 1.5 GeV electron storage ring (R1), and a 3 GeV electron storage ring (R3) [1]; the project presented in this thesis aims to improve the performance of R1. An electron storage ring is a structure used for storing and circulating electrons in a closed loop and the performance of this machinery is dependent on several variables; two typical key performance variables are the beam lifetime and the injection efficiency [2]. The first describes how long the circulating electrons last in the ring, while the second expresses the ability to inject electrons into the storage ring, being typically defined as the ratio between the current getting into the ring and the current provided by the Linac. Optimising the lifetime and injection efficiency of an electron storage ring is a non-linear problem and can suffer from discontinuities and local minima. Moreover, it is shown that the improvement of one variable often leads to the degradation of the other; in other words, there is a trade-off between the lifetime and injection efficiency, which can prevent traditional gradient-based searches from succeeding in this optimisation task [2]. Moreover, traditional optimisation methods are typically designed for a single variable, and combining heterogeneous variables into a single quantity may be challenging. For all these reasons, this project aims to optimize both the lifetime and injection efficiency of R1 using an online Multi-Objective Genetic Algorithm (MOGA). The term 'online' indicates that all interventions and data collection will occur in real-time; MOGA is explained further in the 'Method' section. In order to improve the lifetime and injection efficiency of R1, the algorithm will control one of the sextupole magnet families in R1, namely the outer corrector sextupole (SCo) family. The lifetime, injection efficiency and the SCo family are further explained in the 'Background and Theory' section.
MAX IV hosts more than 1700 users annually, which is planned to double in the following years [1]. Improving the lifetime and injection efficiency of R1 means faster electron injections to R1 and reduced unwanted background radiation during injections [3]. Considering the contributions of the experiments in MAX IV to the progress of scientific knowledge, it is crucial to have a well-performing R1, which in return will result in a better quality scientific output by the users.
The optimisation project presented here is a bachelor's thesis affiliated with the Synchrotron Radiation Department of Lund University and is done in collaboration with the Accelerator Development Group at MAX IV. I wrote a comprehensive Python script utilizing a MOGA and carried out numerous tests to optimise the performance variables of R1. To fully understand the scope of this work; a brief introduction to the main components of MAX IV is provided in the following section.

1.1 MAX IV

MAX IV is a synchrotron radiation facility located in Lund, Sweden, which began operating in 2016 [1]. Synchrotron radiation is electromagnetic radiation produced by accelerating charged particles to relativistic speeds and then curving their path using magnets; in the case of MAX IV, the charged particles are electrons [4]. The synchrotron radiation produced at MAX IV is used for various research techniques, including X-ray spectroscopy, scattering/diffraction, and imaging techniques for a wide range of samples [1].
At MAX IV, electrons are emitted from a cathode through thermionic emission and accelerated to relativistic speeds using a 250-meter-long linear accelerator (Linac), running 4.5 meters underground [3]. The Linac consists of radio-frequency cavities that produce an oscillating electric field: if the arrival time of the electron and the oscillation phase of the electric field inside the

radio-frequency cavity are synchronized, then the electron is accelerated by the electric field [4]. In this way, particles with initial energies of 4 to 5 MeV can reach up to 3 GeV at the end of the Linac [3]. Refer to chapter 1.2.2 in [4] for a thorough description of radio-frequency cavities. The electrons are then extracted vertically at two points along the Linac through dedicated transfer lines and injected into two electron storage rings, R1 and R3. The former has a circumference of 96 m and stores electrons of 1.5 GeV energy, whereas the latter has a circumference of 528 m and stores electrons of 3 GeV energy. R1 can host up to ten beamlines, and today, five beamlines are in use. R3 hosts ten working beamlines with a potential of nineteen beamlines. These are structures built adjacent to the ring where scientists can use the synchrotron radiation for their experiments. R1 and R3 are constructed with repeating sections called achromats with straight sections between them where insertion devices are placed to generate the aforementioned synchrotron radiation. The overall sequence of achromats and straights constitutes the rings [3]. A schematic representation of the MAX IV accelerator complex is shown in Figure 1. Refer to [3] for a comprehensive understanding of the entire facility.

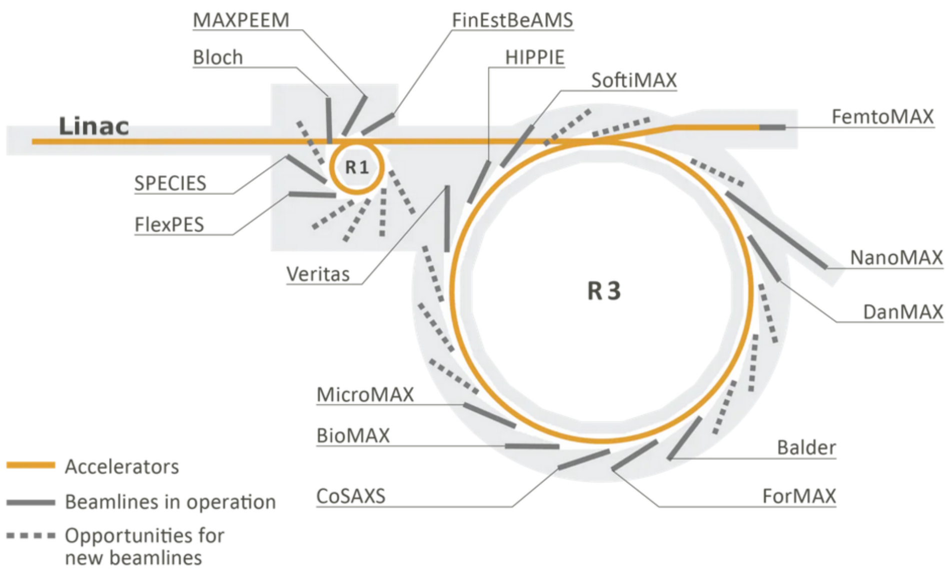

Figure 1: MAX IV accelerator complex including the Linac, R1 and R3. [1]

2 Background and Theory

In this section, the key concepts from the beam dynamics and optics of R1 are introduced. This is crucial to understand the link between the control of the SCo family and the performance of R1, which can be expressed in terms of the key performance variables lifetime and injection efficiency. Lattice optics refer to the arrangement of the magnets in the ring and their corresponding strengths. For a given optics, beam dynamics refers to how charged particles interact with the electromagnetic fields of the ring and how they evolve inside the ring.

2.1 Beta and Dispersion Functions

When the electrons are injected into R1 from the Linac, they form an electron beam, a semi-continuous stream of electrons. The radio-frequency cavities in the ring replace the energy lost to synchrotron radiation while electrons travel through the ring's curved structure at relativistic speeds. The oscillating fields in the radio-frequency cavities force the electron beam to form bunches; these are the densely populated parts of the beam, and there are 32 electron bunches in R1 [3].

An electron that travels through R1 with the ideal energy is called the reference electron, and it follows a closed loop named reference orbit [4]. The motion of an electron through the electron storage ring is usually described by six-dimensional coordinates $(x, p_x, y, p_y, s, \delta)$. The transverse position of an electron is described by x (horizontal) and y (vertical) coordinates, p_x and p_y are the corresponding momentum components. The position of the electron along its trajectory through the ring is denoted by s, and E is the energy of the electron, where the energy of the reference electron is denoted by E_0 [5]. Usually, the parameter δ is used instead of the energy E. This parameter, defined as $\delta = (E - E_0)/E_0$, represents the deviation in energy relative to the reference electron energy [4].

As electrons traverse the storage ring, all electrons, with the exception of the reference electron, undergo transverse oscillations around the reference orbit due to the magnetic restoring forces. These oscillations, known as betatron oscillations, have their amplitude characterized by the beta functions β_x and β_y. Beta functions are derived from the solutions to the equation of motion, which comes in the form of Hill's equation for a complete turn around the ring [4]. The formula for transverse oscillations is given by:

$$u(s) = \sqrt{2J_u\beta_u(s)}\cos(\phi_u) \tag{1}$$

where $u(s)$ is either $x(s)$ for horizontal or $y(s)$ for vertical displacement, J_u is a constant of motion, and ϕ_u represents the betatron phase advance. The term ϕ_u varies according to,

$$\frac{d\phi_u}{ds} = \frac{1}{\beta_u(s)}. \tag{2}$$

This formulation shows that the electron motion in the storage ring resembles that of a simple harmonic oscillator, but the amplitude variations are dictated by $\beta_u(s)$ [4]. Beta functions are plotted in Figure 2 for one achromat of R1.

Another critical concept is the betatron tune, which quantifies the number of betatron oscillations performed in one full turn of the ring. Specific betatron tunes can lead to resonances, which significantly disrupt the beam. These resonances arise when an electron encounters consistent magnetic deflections at specific points within the ring, progressively amplifying the betatron oscillations [4]. As such, tunes sitting on dangerous resonances need to be carefully avoided. In this respect, the amplitude-dependent tune shift, which refers to the variation in the betatron tune with changes in oscillation amplitude, plays a pivotal role in beam stability and dynamics [5]. This can be critical, for example, during the injection phase when electrons usually have large amplitudes. If the lattice is characterized by large amplitude-dependent tune shift, electrons may cross critical resonances and get lost, hampering the injection process. The injection process is further discussed in the 'Dynamic Aperture and Injection Efficiency' section.

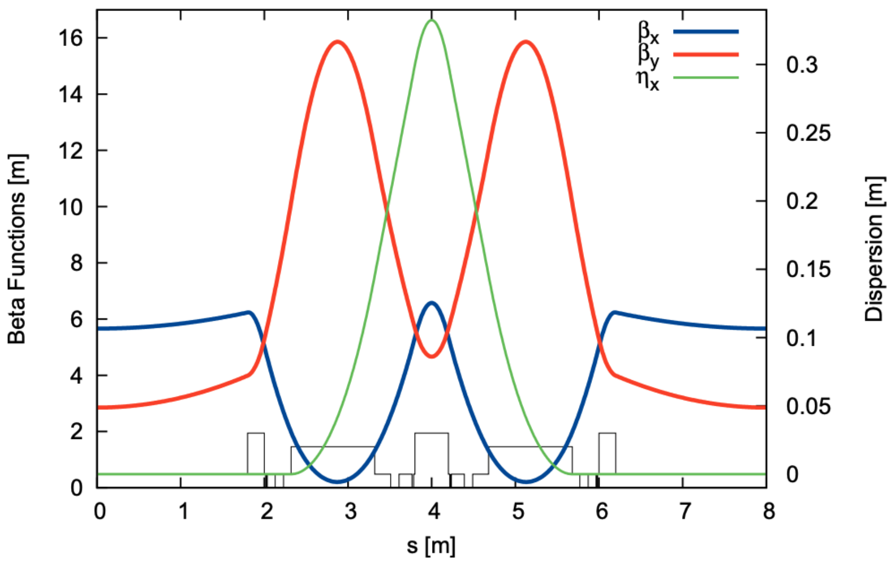

Figure 2: Horizontal (β_x) and vertical (β_y) beta functions are plotted in blue and red, respectively. The dispersion function η_x is plotted in green, the black outline at the bottom represents the magnet layout of one achromat in R1. [3]

The horizontal dispersion function, η_x, highlighted in green in Figure 2, describes how much an electron's horizontal trajectory is affected due to its longitudinal momentum. This is due to the different bending powers of the magnets, which interact differently with electrons that do not precisely match the design momentum specified for that particular ring. Vertical dispersion (η_y) is ideally zero in a storage ring with no vertically bending dipoles and mainly arises due to magnet alignment errors. It is less relevant for this analysis [4]; hence, only η_x is plotted. Referring to Figure 2, the SCo magnets that will be used as variables in the MOGA optimisation are located at 2 and 6 m, *i.e.* in a non-dispersive region of the achromat. The implications of this property are discussed in the following section. Refer to chapters 2.2 and 2.5 in [4] for a comprehensive understanding of beta and dispersion functions.

2.2 R1 Magnets

Dipole, quadrupole, and sextupole magnets are employed in R1 to manipulate the electron beam, ensuring that the electrons oscillate around the reference orbit. The configuration of all magnets in R1 is called the R1 lattice. Dipole magnets bend the beam through the curved parts of the accelerator. Meanwhile, the quadrupole magnets are used to focus and control the size of the beam, ensuring that the electrons remain in a tight, well-defined stream. The quadrupole magnets have magnetic fields varying linearly from where they are centred. Electrons with different energies are focused to different extents in quadrupoles due to the variation in their deflecting angles, which become smaller for higher-energy electrons. This phenomenon is called the chromaticity of the beam and can be compensated by placing certain types of sextupole magnets in the dispersive regions of an achromat. The magnetic field of sextupole magnets varies as the square of the distance from the magnet's centre, allowing it to correct the under or over-focused electron trajectories. The working of the magnets resembles the working of optical lenses; therefore, the layout of the magnets is often referred to as the optics of the ring in the accelerator physics terminology [4].

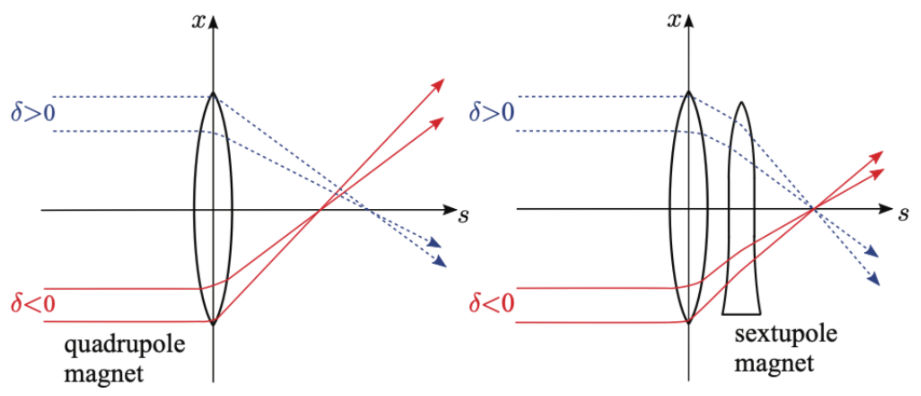

Figure 3: Chromaticity correction of a sextupole with an analogy to optical lenses where δ is the energy deviation from the reference electron energy. The x-axis represents the horizontal direction, and the s-axis is the longitudinal direction. [4]

As can be seen on the left side of Figure 3, electrons with positive δ have a longer focal length (under-focused), and the electrons with negative δ have a shorter focal length (over-focused). Introducing a sextupole magnet, as shown on the right side of Figure 3, can provide additional focusing or defocusing based on the δ of the electron.
R1 consists of 12 achromats where the optics are repeated, and there are six types of sextupole magnets in an achromat. The same type of sextupole magnets are grouped into families in the ring, and the magnets are split into two objective groups called chromatic and harmonic. Chromatic magnets correct the chromaticity aberration of the beam, whereas harmonic magnets correct for the other less predictable non-linear effects [3]. Refer to chapter 4 of [4] for a thorough description of the non-linear effects of an electron beam. The sextupole specifications are presented in Table 1, and their locations in an achromat are depicted in Figure 4.

Magnet Family Name	**Abbreviation**	**Objective**	**per Achromat**
Inner defocusing sextupole	SDi	Chromatic	2
Outer defocusing sextupole	SDo	Harmonic	2
Inner focusing quadrupole/sextupole	SQFi	Chromatic	1
Outer focusing quadrupole/sextupole	SQFo	Harmonic	2
Inner corrector sextupole	SCi	Chromatic	2
Outer corrector sextupole	SCo	Harmonic	2

Table 1: Sextupole magnet specifications of R1, adapted from information presented in [3].

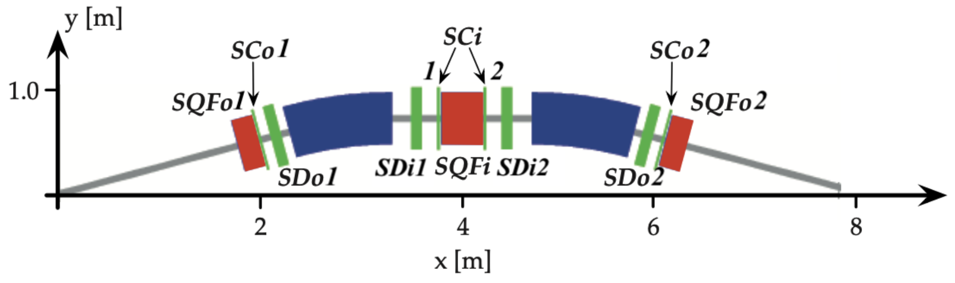

Figure 4: Sextupole magnet locations in an achromat of R1, the grey line represents the trajectory of the electrons. Sextupole magnets from the same family are labelled 1 and 2 in a clockwise direction. [3]

SQFi and SQFo families are integrated into quadruple magnets for a more compact design. Therefore, their strength cannot be adjusted individually. Thus, the relatively smaller and weaker sextupole families, SCi and SCo are introduced for fine-tuning the correction of the SQFi and SQFo families. As mentioned in the previous section, SCo magnets are located in the non-dispersive regions of the achromat, hence these magnets are harmonic and do not correct the chromaticity of the beam. In consideration of its harmonic nature and fine-tuning ability, the SCo family can be used to minimise the impact of the resonances while not interfering with the chromaticity correction of the beam. Moreover, these magnets can shape the amplitude-dependent tune shift, previously mentioned in the 'Beta and Dispersion Functions' section, and provide a more stable beam [3].

2.3 Dynamic Aperture and Injection Efficiency

The dynamic aperture of an electron storage ring determines the maximum permissible distance an electron can deviate from the reference orbit in the transverse plane while still being maintained within the beam [4]. Computed dynamic aperture plots for R1 are illustrated in Figure 5, where the thick line shows an ideal machine and the crosses represent the alterations that occurred in the dynamic aperture due to random errors in the sextupole magnet gradients. The term 'magnet gradient' refers to the rate of change of the magnetic field per unit length.

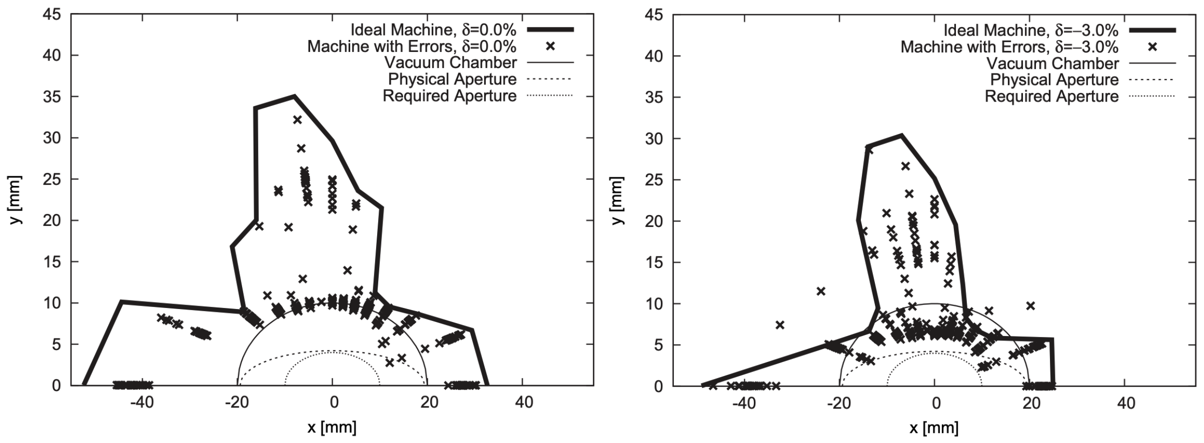

Figure 5: Dynamic aperture plots constructed using the software tool Tracy-3 [6]. The thin full line shows the limits of the vacuum chamber, the dotted line with larger dots shows the limits of the physical aperture, and the dotted line with smaller dots shows the limits of the required aperture of R1 in the transverse plane. On the left, the thick bold line shows the outline of the dynamic aperture for on-energy electrons, indicated by $\delta = 0.0\%$. On the right, the thick bold line shows the outline of the dynamic aperture for off-energy electrons, indicated by $\delta = -3.0\%$. The altered dynamic apertures caused by the random sextupole errors are represented by 'x'. [3]

It is important to note that the electrons can only be physically confined within the limits of the physical aperture, regardless of how large the dynamic aperture may be. Figure 5 indicates that the sextupole errors worsen the dynamic aperture of R1, especially for so-called off-energy electrons, where it jeopardises the physical aperture. Therefore, it is crucial to optimize the sextupole gradients to increase the dynamic aperture.
When the electrons are initially injected into the ring, they tend to have high betatron oscil-

lation amplitudes [4]. Therefore, having a larger dynamic aperture helps retain more of these electrons, resulting in higher injection efficiency of R1 [3]. Injection efficiency is the ratio between the current stored in R1 through electron injection and the current provided by the Linac. Meanwhile, the injection rate is the unit charge injected into R1 per unit time. Assuming that the current provided by the Linac is unchanged, injection efficiency is directly proportional to injection rate, therefore it is common practice in MAX IV to monitor injection rate instead of injection efficiency with regards to the rings. In this thesis, the same strategy will be followed. It is clear that optimising the sextupole gradients, *i.e.* the SCo magnet currents, may result in a better dynamic aperture, which in return improves the injection rate of the ring.

2.4 Momentum Acceptance and Lifetime

The electrons circulating in R1 generate an overall beam current I_{R1}, and R1 presently supports a maximum beam current of 500 mA. Without top-ups, which are regular electron injections to R1, the beam current will gradually decrease over time. This is referred to as R1 being in decay mode and introduces another important concept, the beam lifetime. Lifetime is defined as the time it takes for the beam electron population to decrease by a factor of $1/e$ [4]. The continuous reduction of beam current is essentially due to two main processes: electrons can interact with the residual gas present in the vacuum chamber of R1 undergoing elastic and inelastic scattering, or they can interact inside a bunch, exchanging transversal momentum. The first process depends on the total current of the beam and the quality of the vacuum inside the chamber. The second process is known as Touschek scattering and it is observed when two electrons in an electron bunch collide, and their transverse momenta are translated into longitudinal momenta. If the acquired longitudinal momentum exceeds a threshold known as the momentum acceptance, the motion of the electrons destabilizes, and they are lost from the beam. For a machine with good control of the vacuum, Touschek scattering is the dominant process regarding the loss of electrons in the ring and dictates the lifetime [4]. Over brief periods during which the bunch current remains relatively stable, the decrease in beam current of R1 can be effectively approximated by exponential decay as shown in Equation 3,

$$I_{R1}(t) = I_{R1}(0)e^{-t/\tau_T} \tag{3}$$

where $I_{R1}(t)$ is the beam current at time t, $I_{R1}(0)$ is the initial beam current, t is the time variable, and τ_T is the Touschek lifetime [4]. The aforementioned momentum acceptance measures how much an electron can deviate from the longitudinal momentum of the reference electron and still be contained in its bunch. In other words, it dictates the maximum amplitude of Touschek scattering that electrons can go through and remain in the beam. The proportionality between the Touschek lifetime τ_T and momentum acceptance δ_{acc} is described by Equation 4,

$$\tau_T \propto \frac{\sqrt{\varepsilon_y}\sigma_z}{I_b}\delta_{\text{acc}}^3 \tag{4}$$

where ε_y is the vertical emittance, σ_z is the root mean square bunch length and I_b is the bunch current [7]. Vertical emittance measures the spread of electron trajectories and momenta in the vertical plane [4]. Refer to chapter 2.3 in [4] for a comprehensive understanding of vertical emittance. Based on Equation 4, τ_T is directly proportional to δ_{acc}^3. Therefore, improving the momentum acceptance δ_{acc} will improve the lifetime τ_T, assuming that the other variables are kept constant. The momentum acceptance of R1 is determined by the radio-frequency cavity voltage, and the ring lattice [3]. The momentum acceptance of one achromat of R1 can be plotted as shown in Figure 6.

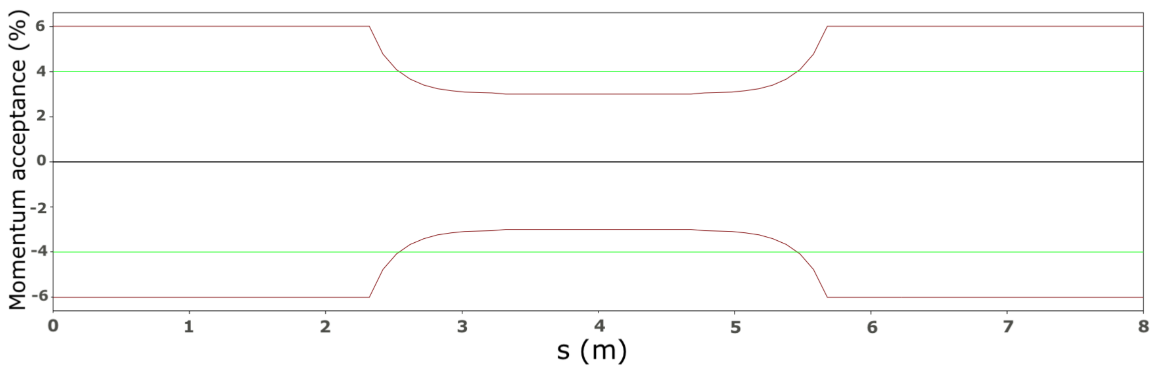

Figure 6: Momentum acceptance of one achromat of R1, as calculated with the software tool OPA [8]. The green line represents the limits imposed on the momentum acceptance by the radio-frequency cavity voltage (0.56 MV), whereas the brown line represents the limits imposed on momentum acceptance by the R1 lattice. [3]

Referring to Figure 6, one can improve the limits imposed on the momentum acceptance (brown line) by improving the R1 lattice, which can possibly be done by optimising the currents of the SCo magnets.
It is significant to highlight that, based on Equation 4, there is an inverse relationship between bunch current I_b and lifetime τ_T. Generally speaking, τ_T decreases as I_b increases. An increase in I_b results in an increase in the overall beam current I_{R1}; hence, the lifetime reading will vary depending on the beam current. Therefore, an important variable used to characterize the performance of the ring other than the plain lifetime is the product of beam current and lifetime $I_{R1} \cdot \tau_T$. In this way, one gets a performance variable almost independent of the beam current variations. However, since the bunch length changes with the charge content of a bunch, this simple equation is not constant for large current variations. Therefore, if one wants to use it as an objective in a process of optimisation spanning through large current variations, $I_{R1} \cdot \tau_T$ has to be scaled with respect to the beam current. As mentioned earlier, Touschek scattering is the dominant process regarding the loss of electrons from the beam and determines the overall lifetime. Hence, τ_T can be equated to τ_{R1} and one can introduce the performance variable, 'scaled $(I \cdot \tau)_{R1}$'. The calculation of this variable is introduced in the 'Method' section.

2.5 Magnet Material and Hysteresis

The magnet material used in MAX IV is ARMCO® Pure Iron. This high-purity iron material is removed from almost all impurities [9]. Due to the magnetic memory of the material, over time, there will be an inconsistency between the applied current to the magnet and the produced magnetic flux density. This is called magnetic hysteresis, whose behaviour can be predicted using a hysteresis loop [10]. The predicted hysteresis loop of the ARMCO® Pure Iron that is magnetised up to 1.5 T is plotted in Figure 7. The positive coercivity value, the material's resistance to demagnetisation, is sourced from the product data bulletin [9]. The positive remanence value, *i.e.* the residual magnetic field after the exciting current is brought back to zero, is calculated based on the slope of the two highest measurements on the virgin curve. The 'virgin curve' is the initial magnetisation curve of a material not previously magnetized, and it is also sourced from the product data bulletin [9]. Subsequently, the negative remanence, coercivity and magnetisation values are determined by mirroring their positive counterparts. Refer to [10] for a comprehensive understanding of hysteresis loops and their terminology.

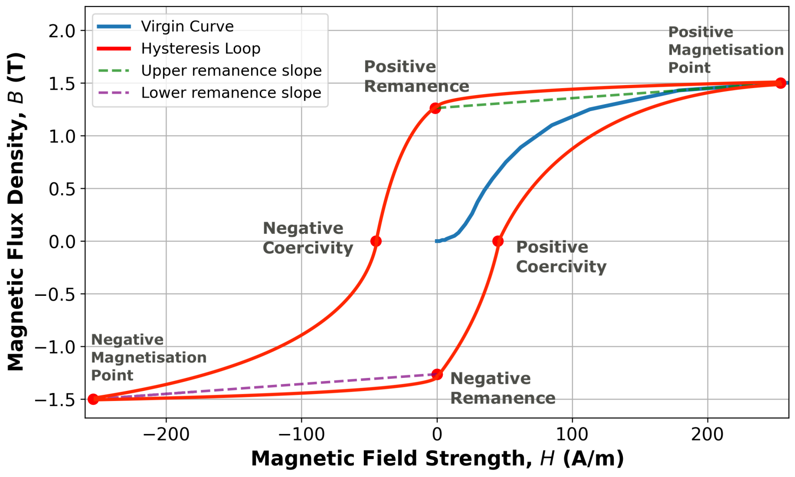

Figure 7: The predicted hysteresis loop of ARMCO® Pure Iron. The blue line represents the virgin curve of the ARMCO® Pure Iron as it is magnetised up to 1.5 T at around 950 °C. The predicted hysteresis loop is plotted in red using the remanence, coercivity and magnetisation values. The figure is adapted from the data presented in [9].

The magnetic field strength H increases linearly with the current of the magnet according to Ampere's Law. In the presence of magnetic hysteresis, a particular magnet current can be associated with multiple magnetic flux density outputs, as demonstrated by the hysteresis loop in Figure 7. One way to deal with hysteresis at MAX IV is to 'cycle R1', which is the process of going all the way up and down in the current range of the magnets for a total of six times and then stopping at the pre-determined current value for the magnets. This process takes about 25 minutes and will get rid of any pre-existing magnetic history. Referring to Figure 7, it is important to note that 1.5 T is not the saturation point of ARMCO® Pure Iron, and it is merely chosen to demonstrate a particular case. The appearance of the hysteresis loop of the magnet material will vary depending on the level of magnetisation.

3 Method

In the 'Background and Theory' section, it was explained that the injection rate instead of the injection efficiency and scaled $(I \cdot \tau)_{R1}$ instead of the lifetime will be considered as the performance variables of R1. A multi-objective genetic algorithm approach will be used to optimise these performance variables, referred to as the 'objectives' of the optimisation. The genetic algorithm utilised in this project is the Non-dominated Sorting Genetic Algorithm II (NSGA-II), first proposed in 2002 [11]. A Python adaptation of NSGA-II (n_ii.py) is taken from the py-Multiobjective Python Library [12], and a comprehensive Python script utilizing 'n_ii.py' was written to optimise R1. The Python script will be referred to as the 'optimiser', and it is presented in the 'Appendix' section. It should be noted that writing and successfully running the optimizer took a significant amount of time, requiring a thorough comprehension of the MAX IV control systems.
Control and monitoring of the MAX IV ring and other parts of the facility are operated by means of the Tango toolkit [13]. Tango provides a communication line between the physical devices in R1 (detectors and instruments) and the MAX IV main control room. Tango is structured into virtual devices mimicking the physical devices used in the rings to monitor and

control the machine. The optimiser can proxy into the necessary Tango devices to perform commands, measurements or calculations. The optimiser consists of three main components: ‘Magnet Control’, ‘Objective Measurements’ and ‘Genetic Algorithm’. These components are separately explained in the following sections.

3.1 Magnet Control

There are 24 SCo magnets in R1, and a visual overview of their locations on the ring is presented in Figure 8.

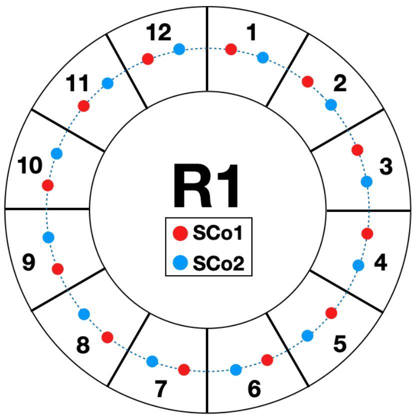

Figure 8: Locations of the SCo magnets in R1. The numbers one through twelve denote the achromat number. Each achromat hosts two SCo magnets, namely SCo1 and SCo2.

The following naming convention will be used to address an individual SCo magnet: (Achromat number, SCo number). For instance, magnet (1,2) refers to the SCo2 magnet in achromat 1. Each SCo magnet is connected to a designated power supply, and it is possible to proxy into the Tango devices of these power supplies to adjust their currents. The designated power supply for each SCo magnet is listed in the optimiser script which can be found in the ‘Appendix’. The SCo configuration of R1, typically used during beam delivery to the beamlines, is presented in Table 2 and will be referred to as the ‘standard lattice’.

Achr.	1		2		3		4		5		6	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
I_{SCo}	0.597	0.597	0.597	0.597	0.597	0.597	0.597	0.597	0.597	0.597	0.597	0.597
Achr.	7		8		9		10		11		12	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
I_{SCo}	0.597	0.597	0.597	0.597	0.597	0.000	0.000	0.597	0.597	0.597	0.597	0.597

Table 2: Magnet current readings of the R1 SCo family, which make up the standard lattice. ‘I_{SCo}’ is the magnet current in A, and the noise of the current reading is $\pm 1.6 \times 10^{-5}$ A. ‘Achr.’ stands for achromat number and ‘SCo’ denotes if the magnet is SCo1 or SCo2.

All SCo magnets in R1 are set to $0.597 \pm 1.6 \times 10^{-5}$ A, the only exceptions being the magnets (9,2) and (10,1), which are set to $0.0 \pm 1.6 \times 10^{-5}$ A. These exceptions are due to the local orbit corrections of BLOCH, which is the beamline located in that region of the ring [14]. Therefore, these two magnets will be excluded from the optimisation, and their current values will not be changed. The noise level of the current readings was determined by taking the standard deviation of 180 readings with 1-second intervals. The results are presented in Figure 9. The magnet (1,2) has the highest standard deviation recorded at 1.6×10^{-5}. Therefore, this value is used as the noise level for all SCo magnet readings.

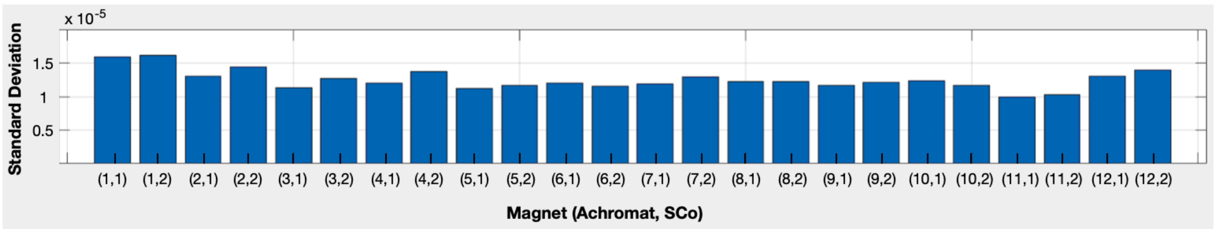

Figure 9: The noise levels of all 24 SCo magnets. The y-axis shows the standard deviation of 180 current readings with 1-second intervals, and the x-axis shows the name of the SCo magnet in the following format (Achromat number, SCo Number).

3.2 Objective Measurements

The R1 beam current I_{R1} is measured using a Tango device called DC current transformer (DCCT), which is connected to a physical DCCT in R1. The DCCT measures the voltage induced in a coil of wire by the magnetic field around the beam [4], which is then converted to a current reading usually expressed in mA.
R1 lifetime τ_{R1} is also calculated by the same Tango device (DCCT) using Equation 3, from the 'Background and Theory' section. Equation 3 can be written in a linear form as follows,

$$I_{R1}(t) = I_{R1}(0)e^{-t/\tau_{R1}} \tag{5}$$

$$\longrightarrow \ln(I_{R1}(t)) = -\frac{1}{\tau_{R1}}t + \ln(I_{R1}(0)) \tag{6}$$

This linear relationship has the form $y = mx + c$, where y is $\ln(I_{R1}(t))$, the slope m is $-1/\tau_{R1}$, x is t and c is $\ln(I_{R1}(0))$. The slope of Equation 6 is then obtained by means of a linear fit of 300 I_{R1} readings taken over a time window of 30 s. The R1 lifetime is then simply inferred by inverting the slope. Lifetime τ_{R1} is typically expressed in hours.
Subsequently, the product of I_{R1} and τ_{R1} gives the variable, $(I \cdot \tau)_{R1}$. However, as mentioned previously, this variable needs to be scaled with respect to the R1 beam current. Therefore, $(I \cdot \tau)_{R1}$ values are collected at beam currents from 150 mA to 500 mA with 50 mA intervals. Then a 'scaling curve' is constructed by plotting $(I \cdot \tau)_{R1}/\text{reference}(I \cdot \tau)_{R1}$ vs R1 beam current, where the reference $(I \cdot \tau)_{R1}$ is equal to the $(I \cdot \tau)_{R1}$ value at 350 mA of beam current. Applying a cubic fit to the scaling curve provides a scaling factor in the following form,

$$\text{scaling factor} = a(I_{R1})^3 + b(I_{R1})^2 + c(I_{R1}) + d \tag{7}$$

where a, b, c, and d are the coefficients of the cubic fit. During the optimisation, all $(I \cdot \tau)_{R1}$ calculations are divided by this scaling factor to align the calculation with respect to the $(I \cdot \tau)_{R1}$ value at 350 mA of beam current. The resulting variable is correlated to the beam lifetime in a virtually independent way from the variations in the beam current.
The injection rate of R1 is read from a Tango device called the injection-meter, which is connected to the physical DCCT device. During an electron injection to R1, the injection-meter takes 50 I_{R1} measurements over a window of 5 seconds and then applies a linear fit to these points. The slope of the linear fit simply produces an injection rate reading in mA/min.

During the optimisation process, both objectives - scaled $(I \cdot \tau)_{R1}$ and injection rate - will be evaluated 10 times at 0.5-second intervals. This frequent assessment is particularly critical for the injection rate reading, which exhibits rapid fluctuations that change almost every 0.5 seconds.

For both objectives, the mean and standard deviation of the data will be calculated; the former will be used as the objective value in the algorithm, and the latter will be utilized to define the uncertainty of the objective value.

3.3 Genetic Algorithm

The algorithm controls the SCo magnet currents to optimise the scaled $(I \cdot \tau)_{R1}$ and injection rate. These objectives have a trade-off, meaning that improving one objective often leads to a decrease in the other. Optimising two objectives with a trade-off results in a non-dominated solution set called a Pareto optimal set, which can be depicted using a Pareto front in the objective space [15]. A visual representation of a Pareto front for this type of optimisation is shown in Figure 10.

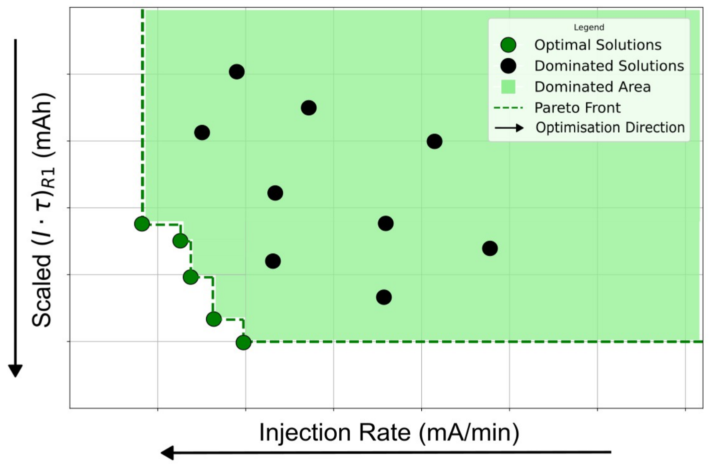

Figure 10: Optimal solutions which construct the Pareto front are painted in green, the dominated area is painted in light green, and the dominated solutions are painted in black. The green dotted line shows the Pareto front, and the black arrows show the direction of the optimisation.

Each solution represents a different R1 lattice distinguished by its SCo current levels, which will be referred to as the SCo configuration of the solution. It is important to note that the direction of the optimisation goes towards the origin of the plot, meaning that the objective values increase in that direction. An arbitrary solution A dominates an arbitrary solution B only if the following conditions are met:

- **Condition 1:** Solution A is no worse than solution B in both objectives.
- **Condition 2:** Solution A is strictly better than solution B in at least one objective.

As shown in Figure 10, the solutions on the Pareto front are not dominated by any solutions. Hence, they are called a non-dominated solution set [11]. The algorithm essentially seeks the best Pareto front possible following the critical steps illustrated in Figure 11.

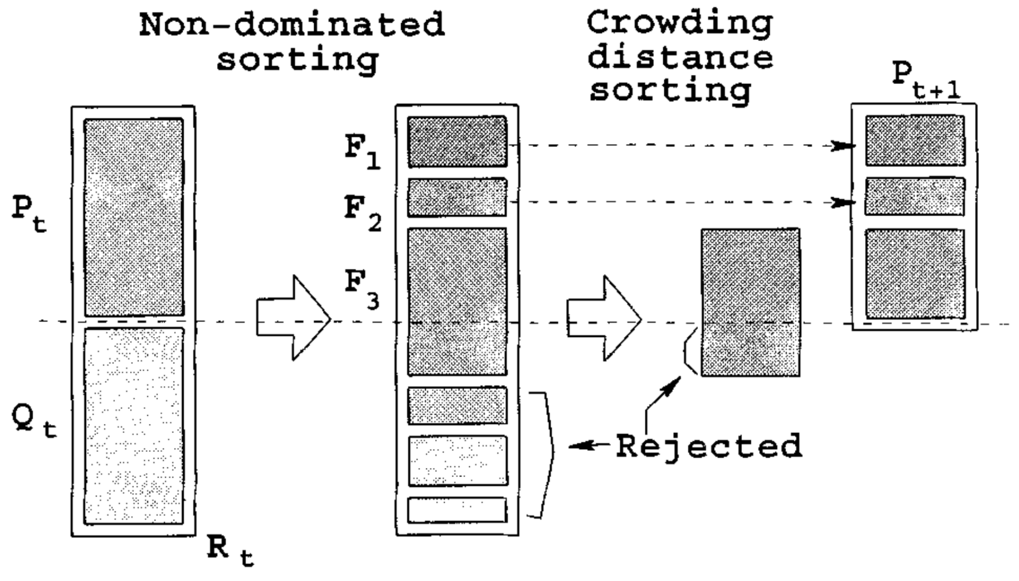

Figure 11: Step by step evolution of NSGA-II. Two sorting methods, namely non-dominated sorting and crowding distance sorting, are annotated on the diagram. R_t represents a generation with the index t, P_t denotes the parent population and Q_t denotes the offspring population. F_1, F_2, and F_3 represent Pareto fronts arranged in descending order of dominance. [11]

As depicted in Figure 11, each generation, denoted by R_t, contains a total of 2N solutions. Here, t represents the generation index, with the sequence of generations progressing as R_0, R_1, R_2 and so on. A generation R_t includes a parent population P_t of size N and an off-spring population Q_t of size N.

The algorithm starts by constructing an initial parent population P_0, which consists of randomly generated solutions. These solutions are then ranked (1, 2, 3... where 1 is the best rank) based on dominance. Subsequently, the solutions with better rank and crowding distance are selected; this process is called 'binary tournament selection'. The crowding distance measures how spread out a solution is on the objective space compared to the other solutions; this ensures that diversity is maintained through the generations. The selected solutions from P_0 give birth to the initial offspring population Q_0 by performing crossover and mutation. Crossover is the process where two parent solutions exchange some parts of their SCo configurations and become two offspring solutions with unique SCo configurations. On the other hand, mutation is the random modification of the SCo configuration of a selected parent solution, which then becomes a unique offspring solution. Then, the initial parent and offspring populations, P_0 and Q_0, are combined and sorted into Pareto fronts (F_1, F_2, F_3... where F_1 is the best Pareto front) based on dominance. This process is labelled as 'non-dominated sorting' in Figure 11.

At this point, the construction and sorting of the initial generation R_0 is complete, and it is time to construct R_1. The parent population P_1 is then created by selecting the solutions in the top 50% of R_0 based on the Pareto front sorting. If the cut-off line goes through a Pareto front, which is the case for F_3 as shown in Figure 11, the solutions with the higher crowding distance are favoured; this process is called 'crowding distance sorting'. Subsequently, the parent population P_1 gives birth to Q_1 through binary tournament selection, crossover and mutation as mentioned earlier. Then, P_1 and Q_1 are combined and sorted into Pareto fronts. The following generations R_2, R_3... are constructed following the same steps used to construct R_1 [11].

The sequence explained above runs until the number of generations selected by the user is reached. Key parameters utilized to tune the performance of NSGA-II are explained in Table 3.

Parameter	Code	Description
Population Size	population_size	Determines the number of solutions created for a parent or offspring population.
Number of Generations	generations	Determines the number of generations created during the optimisation.
Mutation Rate	mutation_rate	Determines the probability of a mutation occurring during the creation of an offspring. This parameter is typically set to 0.1 in the algorithm.
Mutation Intensity (μ)	mu	Controls the intensity of mutations, affecting how significantly offspring can differ from the parent solution. This parameter is typically set to 1 in the algorithm.
Crossover Intensity (η)	eta	Describes how closely offspring resemble their parents, influencing the crossover process. This parameter is typically set to 1 in the algorithm.
Min I_{SCo}	min_values	The lowest possible current setting for an SCo magnet.
Max I_{SCo}	max_values	The highest possible current setting for an SCo magnet.

Table 3: Summary of the key parameters of NSGA-II. These parameters, referred to by their 'Code', are used in the algorithm function (non_dominated_sorting_genetic_algorithm_II) of the optimiser. The table is adapted from the information provided in [12].

3.4 Optimiser Overview

Thus far, the three main components of the optimiser have been discussed, namely 'Magnet Control', 'Objective Measurements' and 'Genetic Algorithm'. A visual representation of how the optimiser works using these components is shown in Figure 12.

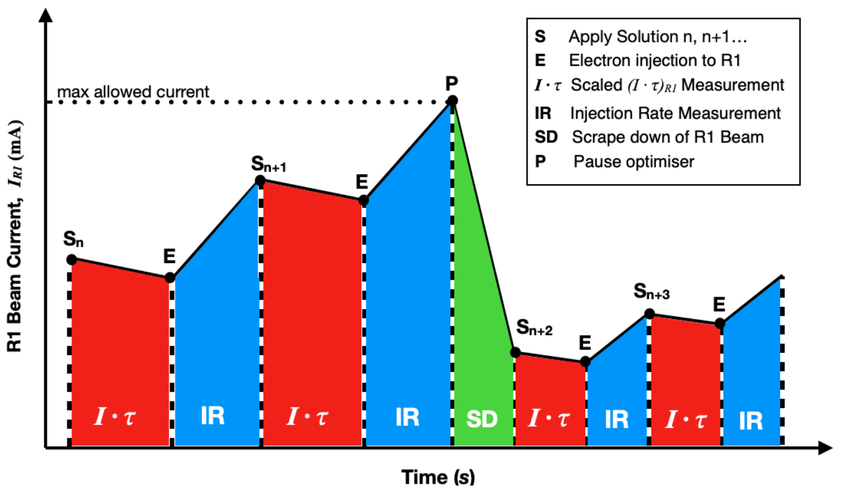
S Apply Solution n, n+1...
E Electron injection to R1
$I\cdot\tau$ Scaled $(I\cdot\tau)_{R1}$ Measurement
IR Injection Rate Measurement
SD Scrape down of R1 Beam
P Pause optimiser
max allowed current
S_n
E
S_{n+1}
E
P
S_{n+2}
E
S_{n+3}
E
$I\cdot\tau$
IR
$I\cdot\tau$
IR
SD
$I\cdot\tau$
IR
$I\cdot\tau$
IR
R1 Beam Current, I_{R1} (mA)
Time (s)

Figure 12: A general overview of an optimisation run, which shows the evaluation of several solutions.

Determining the objective values corresponding to a specific solution S_n chosen by the algorithm starts with applying the solution to R1, which entails implementing the SCo current values corresponding to that solution. As mentioned in the 'Objective Measurements' section, the DCCT requires at least 30 seconds to provide a lifetime reading. Therefore, the scaled $(I \cdot \tau)_{R1}$ is calculated after a 50-second delay; the additional 20 seconds are added as a buffer to ensure the beam and the lifetime reading are stable. This process is denoted by '$I \cdot \tau$' in Figure 12. Following this, an injection to R1 is started with a 2 Hz repetition rate. The optimiser waits 20 seconds for the injection-meter to reach a steady reading, measures the injection rate and then stops the injection; this process is denoted by 'IR' in Figure 12. This sequence repeats for the proceeding solutions S_{n+1}, S_{n+2}... until the maximum allowed beam current is reached (typically 450 mA). At this point, the optimiser is paused, and the beam current is decreased by means of beam scrapers. These tools are inserted into the ring and physically intercept the outer region of the electron beam, causing current loss. When a pre-defined baseline current is reached, the optimiser is resumed and continues until the total number of solutions are evaluated. One can use the following equation to determine the total number of solutions created and evaluated during an optimisation run:

$$\text{Number of Solutions} = 2 \cdot \text{Population Size} \cdot \text{Number of Generations} \tag{8}$$

Given that the evaluation of the objectives for a single solution requires approximately 1.5 minutes, and accounting for occasional scrape downs, which take about 10 minutes each, the duration of an optimisation run can be estimated in minutes as follows:

$$\text{Optimisation Duration} = (\text{Number of Solutions} \cdot 1.5) + (\text{Number of Scrape Downs} \cdot 10) \tag{9}$$

Thus, a typical optimisation run with a population size of 10 across 7 generations, assuming one scrape down per generation, will take approximately 5 hours to complete.

3.5 Optimisation Plan

This thesis will present the results of the final optimisation attempt out of a series of 15 optimisation tests, which amount to about 150 hours of experimental work. Initially, efforts were made to optimise the standard lattice of R1, which proved to be rather challenging to improve in terms of scaled $(I \cdot \tau)_{R1}$ and injection rate. It is most likely because R1 has reached a well-calibrated SCo configuration over the years and is rather optimised.
In view of this, it has been decided on a different strategy, essentially meant to prove the validity of the optimiser. Two separate optimisation runs were executed, named Part 1 and Part 2. In Part 1, the standard lattice was altered to generate an initially spoiled version of the machine, and an optimisation run was performed using this altered lattice as the starting point. The aim was to demonstrate how the genetic algorithm method could be used to improve the objectives and possibly to recover or even improve the standard machine. In Part 2, a solution was chosen from the resulting Pareto front of Part 1, and an optimisation run was performed using the chosen solution as the starting point. The focus was on refining the minimum and maximum SCo current limits, concentrating on the SCo configuration of the selected solution. The procedure for both parts is explained in detail in the following sections.
During both optimisation runs, the existing feedback systems at MAX IV were utilized to correct the beam's orbit and manage betatron oscillations effectively. These systems include the slow orbit feedback (SOFB) and the tune feedback. The former adjusts beam trajectory misalignments using corrector dipole magnets, while the latter regulates the betatron tune of the ring by employing SQFO and DIPC magnet families [14]. For a detailed understanding of the SOFB and tune feedback, refer to chapter 3.4.6 in [3] and to [16], respectively. For the purposes of this thesis, it is sufficient to note that the feedback systems were operational during both optimisation runs, ensuring consistent working conditions.

3.5.1 Part 1: Altered Lattice Optimisation

Part 1 started with measuring the $(I \cdot \tau)_{R1}$ scaling curve of the standard lattice to be able to carry out scaled $(I \cdot \tau)_{R1}$ calculations. Then began the process of altering the standard lattice to have a relatively worsened start to the optimisation. This process resembles the random sextupole error study in Figure 5 from the 'Dynamic Aperture and Injection Efficiency' section. In this case, the currents of the SCo magnets were manually changed while simultaneously observing the scaled $(I \cdot \tau)_{R1}$ and injection rate of R1 until a lattice with relatively worse objective values was found. When the search was over, R1 was cycled to clear the magnetic history of the SCo magnets, and the altered lattice was applied to R1. After this, the $(I \cdot \tau)_{R1}$ scaling curve of the altered lattice was measured, and during the optimisation run, this scaling curve was used to calculate the scaled $(I \cdot \tau)_{R1}$ values of all the solutions. The optimisation run was conducted using the parameters in Table 4 and the range of the beam current was set from 150 to 450 mA.

Pop Size	**Generations**	**Mutation Rate**	$\boldsymbol{\mu}$	$\boldsymbol{\eta}$	**Min $\boldsymbol{I_{SCo}}$ (A)**	**Max $\boldsymbol{I_{SCo}}$ (A)**
10	7	0.1	1	1	0	1

Table 4: Optimisation parameters for Part 1. 'Pop size' stands for 'Population Size' and 'Generations' stands for 'Number of Generations'. Refer to Table 3 for the parameter descriptions.

The parameters 'Min I_{SCo}' and 'Max I_{SCo}' were set to 0 and 1 A, respectively. This decision was guided by the SCo configuration of the standard lattice, where SCo magnets are typically set at $0.597 \pm 1.6 \times 10^{-5}$ A. Optimizing within this range (0 to 1 A) allows for the potential recovery of the SCo configuration of the standard lattice. After the optimisation run was complete, the solutions that make up the Pareto front of the final generation were re-evaluated after cycling R1. Additionally, the $(I \cdot \tau)_{R1}$ scaling curves of two more solutions were measured after the optimisation run to compare the scaling curves of several different lattices.

3.5.2 Part 2: Solution-based Optimisation

In this part, one of the solutions from the resulting Pareto front of Part 1 was selected, and an optimisation run was conducted trying to operate around the selected solution. To do this, the SCo current values were carefully varied close to the chosen point as illustrated in Table 5. Hence, if one represents the SCo configuration of the selected solution using a 22-element vector $S = (s_1, s_2, s_3 ...)$, where each term represents the current of an individual SCo magnet, the magnet current range of this optimisation run was $S \pm 0.1$ A. It is important to remember that it is a 22-element vector instead of 24, because two SCo magnets are excluded from the optimisation as explained in the 'Magnet Control' section. This optimisation run was conducted using the parameters summarized in Table 5 and the beam current ranging from 250 to 450 mA.

Pop Size	**Generations**	**Mutation Rate**	$\boldsymbol{\mu}$	$\boldsymbol{\eta}$	**Min $\boldsymbol{I_{SCo}}$ (A)**	**Max $\boldsymbol{I_{SCo}}$ (A)**
10	7	0.1	1	1	S - 0.1	S + 0.1

Table 5: Optimisation Parameters for Part 2. Refer to Table 3 for the parameter descriptions.

The choice of 'Min I_{SCo}' and 'Max I_{SCo}' values allowed a more focused, solution-based optimisation. The $(I \cdot \tau)_{R1}$ scaling curve of the selected solution from Part 1 was used to calculate the scaled $(I \cdot \tau)_{R1}$ of all the solutions. After the optimisation run, three solutions from the resulting Pareto front were re-evaluated after cycling R1. To ensure that $(I \cdot \tau)_{R1}$ scaling produces consistent results regardless of the beam current, the selected solutions were re-evaluated at three different beam current levels.

4 Results

The results of the optimisation plan will be presented in two parts following the same titling convention, namely ‘Altered Lattice Optimisation’ and ‘Solution-based Optimisation’.

4.1 Part 1: Altered Lattice Optimisation

The optimisation process began by altering the standard lattice of R1. During this process, the electron beam was lost multiple times, indicating the sensitivity of R1 to the significant changes made in the SCo magnets. Therefore, the adjustments were approached with caution to minimize disruptions to the beam. To compare the SCo configuration of the standard and altered lattices, a plot is presented in Figure 13. The data of the standard lattice is taken from Table 2 in the ‘Method’ section, while the data of the altered lattice can be found in the ‘Appendix’.

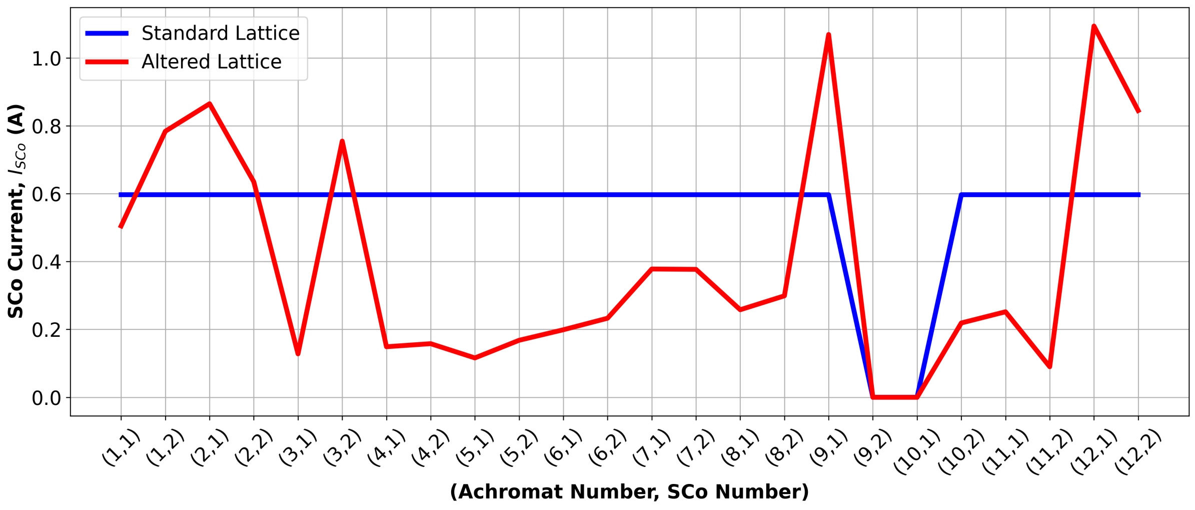

Figure 13: SCo configurations of the standard (blue) and altered (red) lattices.

At a first glance, one can see that the altered lattice deviates from the standard lattice symmetric scheme, exhibiting significant differences from achromat to achromat. The key properties of the standard and altered lattices are evaluated and presented in Table 6.

Lattice	I_{R1} **(mA)**	τ_{R1} **(h)**	**Scaled** $(I \cdot \tau)_{R1}$ **(mAh)**	**IR (mA/min)**
Standard	149.216 ± 0.018	9.633 ± 0.018	3222.842 ± 0.342	53.673 ± 0.545
Altered (BC)	149.480 ± 0.023	8.666 ± 0.001	2825.385 ± 0.322	28.180 ± 0.412
Altered (PC)	150.652 ± 0.020	10.157 ± 0.001	3400.929 ± 0.350	30.827 ± 1.195

Table 6: Key properties of the standard and altered lattice, ‘BC’ stands for before-cycling, ‘PC’ stands for post-cycling and ‘IR’ stands for injection rate. I_{R1} indicates the R1 beam current at which the evaluations of the scaled $(I \cdot \tau)_{R1}$ and injection rate started. The uncertainties are the standard deviations of the corresponding measurements or calculations performed 10 times at 0.5-second intervals.

It is important to note that the altered lattice was evaluated twice, before and after cycling R1. This was done to understand the effect of hysteresis. During the search for the altered lattice, the currents in the SCo magnets were varied multiple times, determining a magnetic history that was subsequently cleared with the cycling process. The intention of altering the lattice was to decrease the scaled $(I \cdot \tau)_{R1}$ and injection rate values compared to the standard lattice, which was achieved in the before-cycling evaluation. However, in the post-cycling evaluation, the altered lattice showed improved objective values, with the scaled $(I \cdot \tau)_{R1}$ even exceeding that of the standard lattice. This indicates the significant impact of magnetic hysteresis on the performance of R1. It is important to note that even though the altered lattice (post-cycling) had a higher scaled $(I \cdot \tau)_{R1}$ than the standard lattice, it is not necessarily a better lattice as its injection rate was significantly worse. In fact, in order to claim a successful optimisation, both the scaled $(I \cdot \tau)_{R1}$ and injection rate must be improved. The optimisation run was then conducted using the post-cycling altered lattice as the starting point. The generation-by-generation evolution of the optimisation run is presented in Figure 14, where only every second generation (even-numbered generations) is presented for the sake of clarity.

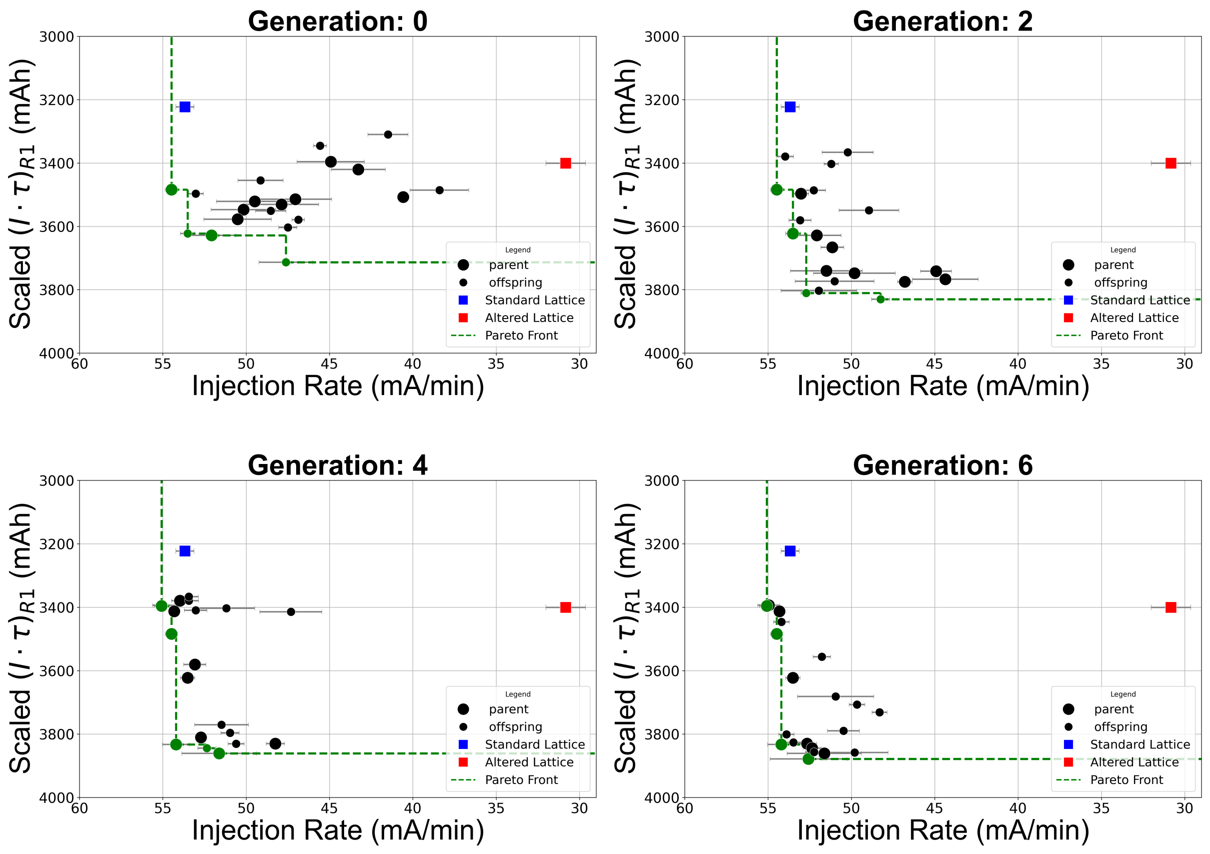

Figure 14: Optimisation results are presented generation by generation, showing only even generations. The green dotted line represents the Pareto front of each generation and the solutions that are a part of the Pareto front are painted in green. The larger circles represent the parent population, while the smaller circles show the offspring population of each generation. The performance of the standard (blue square) and altered (red square) lattices are also presented in each plot as references.

The obtained results indicate that, with each generation, the solutions gradually move away from the starting point, which is the altered lattice, and achieve higher scaled $(I \cdot \tau)_{R1}$ and injection rate values. Already in generation 0, some solutions reach the injection rate value of the standard lattice and even surpass it. The solutions start to converge towards the optimal levels, and by generation 6, a Pareto front of four different solutions is obtained: these are the best outcomes of this optimisation run. As shown in Figure 15, these solutions (circles), colour-coded as 1, 2, 3, and 4, are re-evaluated after cycling (triangles).

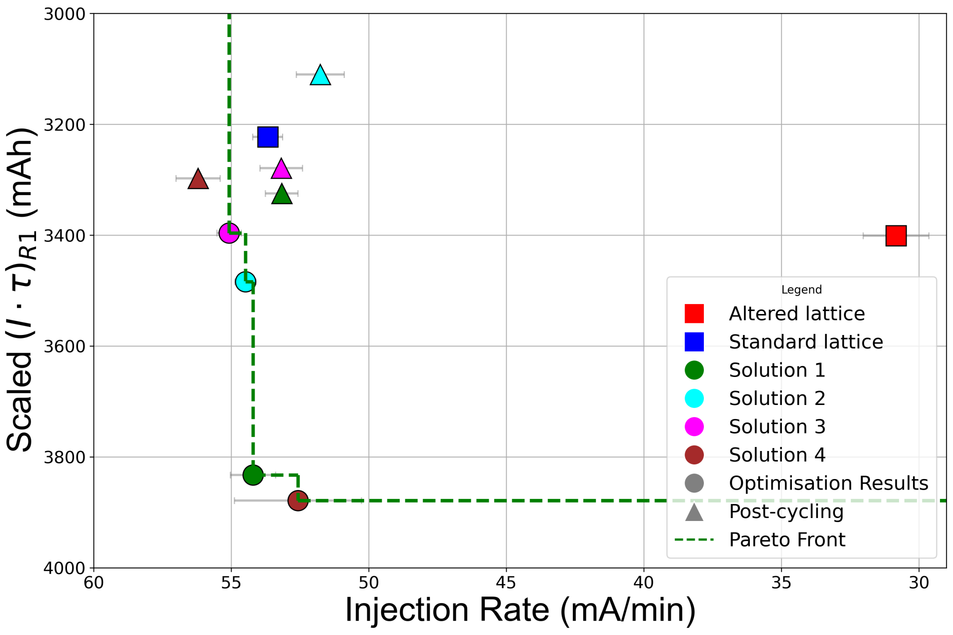

Figure 15: Resulting Pareto front (dotted green line) of Part 1. The optimisation results are represented by circles, and their post-cycling evaluation is represented by triangles using the corresponding colours detailed in the legend. The standard (blue) and altered (red) lattices are also shown for reference.

There is a noticeable difference between the optimisation results and post-cycling results, suggesting that clearing the magnetic history by means of cycling may reduce the efficacy of the optimisation, sometimes in a very negative fashion. However, while these solutions did not confirm the expected results, they are still located close to the standard lattice, which makes them a viable alternative. Moreover, the sequence in the plots in Figure 14 and the graph in Figure 15, ensure that the optimisation method works as intended and is able to recover a spoiled machine even providing better alternatives before cycling R1. The SCo configurations of the selected solutions are illustrated in Figure 16, with the corresponding data detailed in the 'Appendix'.

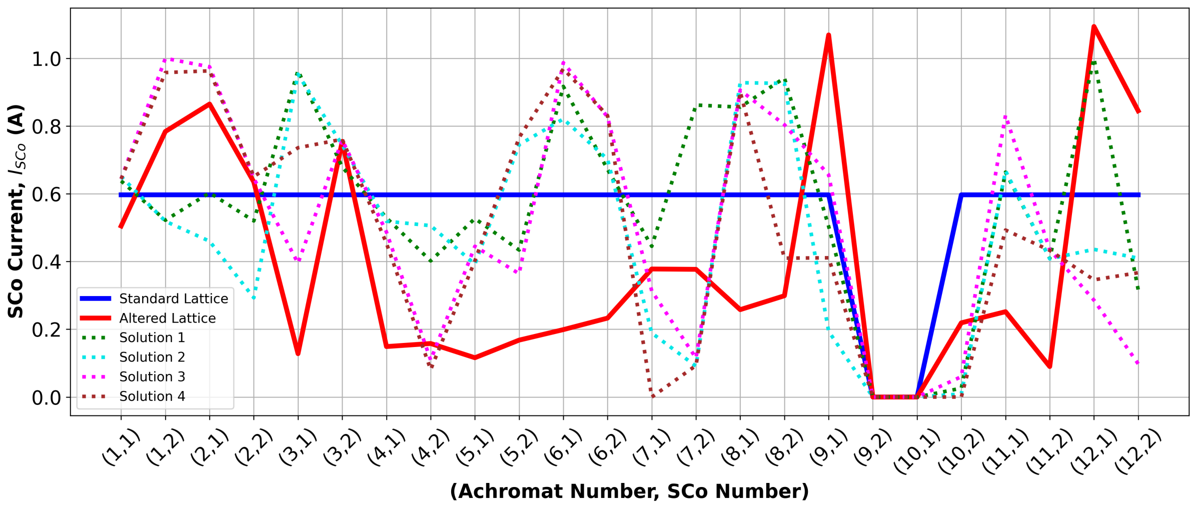

Figure 16: The thick blue and red lines show the SCo configurations of the standard and altered lattices, respectively. The dotted lines illustrate the SCo configurations of the selected solutions; see the legend for the colour assignment.

There is a considerable overlap between the SCo configurations of the selected solutions. This suggests that the algorithm might have found an SCo configuration that performs as well as the standard lattice even after cycling R1. Solution 1 (post-cycling) is a suitable starting point for the next optimisation run because it offers an optimal balance between both objectives, as seen in Figure 15. Also, one can claim that it deviates the least from the SCo configuration of the standard lattice; therefore, it might be a sensible starting point for further exploration.
$(I{\cdot}\tau)_{R1}$ scaling curves of the standard and altered lattices were measured before the optimisation run. Additionally, the scaling curves of solutions 1 and 4 were measured after the optimisation, and all four scaling curves are plotted in Figure 17.

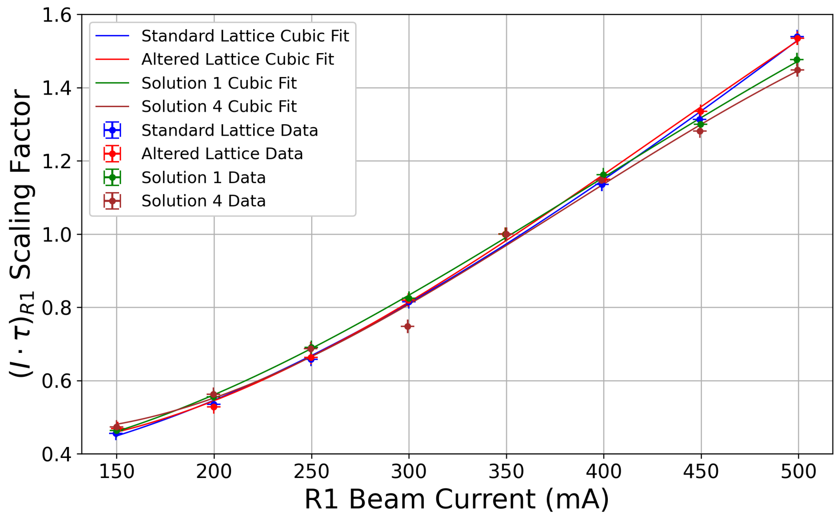

Figure 17: The $(I \cdot \tau)_{R1}$ scaling curves of the standard lattice, altered lattice, solution 1, and solution 4 are collectively plotted. Uncertainty bars represent the standard deviation of measurements or calculations, which were performed 10 times at 0.5-second intervals. The corresponding coefficients for the cubic fits are detailed in the 'Appendix'.

It can be concluded that the four $(I \cdot \tau)_{R1}$ scaling curves are sufficiently similar. This justifies the assumption that one can use an initially calculated scaling curve through the whole optimisation. It is also apparent that the scaling is mostly linear between the beam currents 250 and 450 mA. Thus, the next optimisation run (Part 2) was conducted between these beam current values to have the most optimal $(I \cdot \tau)_{R1}$ scaling.

4.2 Part 2: Solution-based Optimisation

In the second optimisation run, solution 1 (post-cycling) was used as the starting point, and the scaling curve of the said solution was used to calculate the scaled $(I \cdot \tau)_{R1}$ of all the solutions. As explained in the 'Optimisation Plan' section, the minimum and maximum SCo current limits were reduced to be within ± 0.1 A of the SCo configuration of solution 1 in an attempt to refine the chosen case. The generation-by-generation evolution of the optimisation run is presented in Figure 18. Once again, only every second generation is presented for clarity purposes.

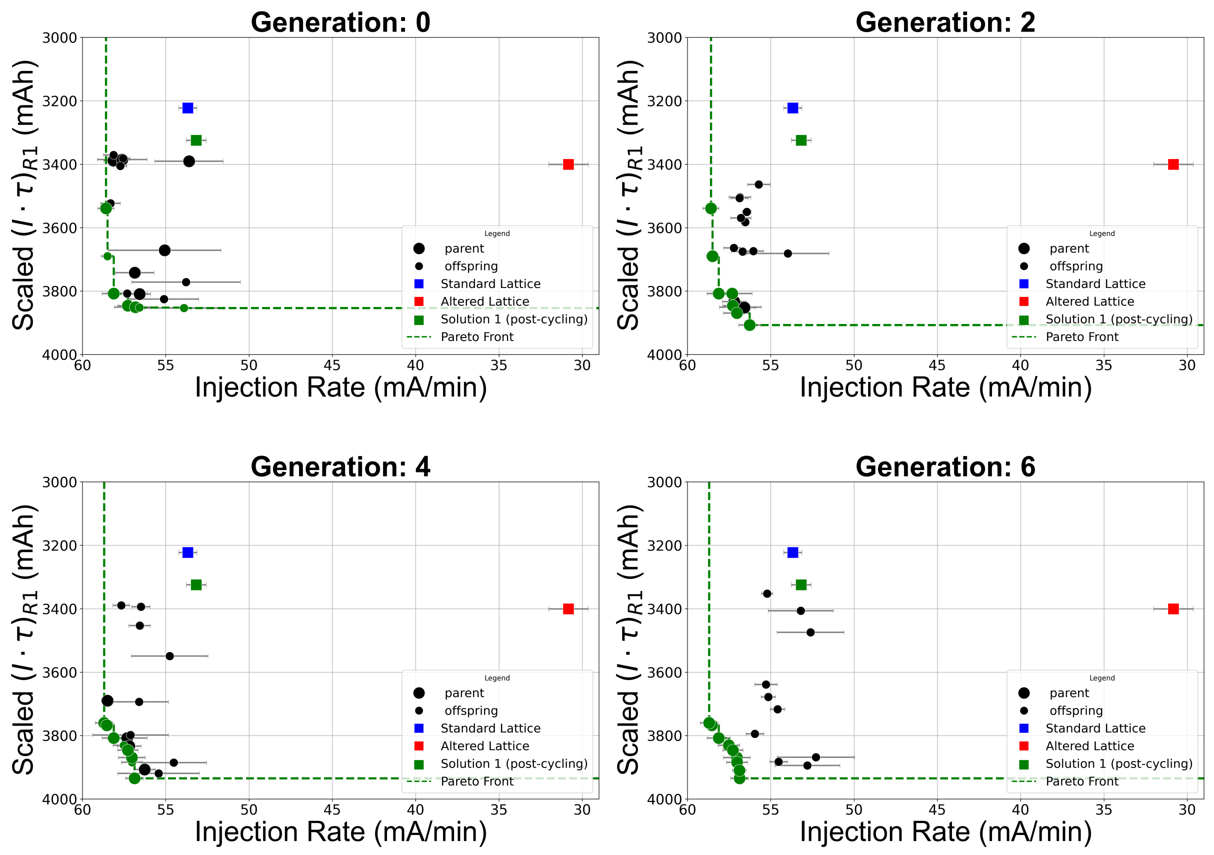

Figure 18: Optimisation results are presented generation by generation, showing only even generations. The green dotted line represents the Pareto front of each generation and the solutions that are a part of the Pareto front are painted in green. The larger circles represent the parent population, while the smaller circles show the offspring population of each generation. The performance of the standard and altered lattices, as well as solution 1 (post-cycling), are also presented in each plot as references.

Already with generation 0, about half of the solutions are in the proximity of the Pareto front. As generations progress, solutions gradually improve and the density of solutions around the Pareto front increases. At generation 6, the solutions converge and form a Pareto front from which three solutions (designated as 1.1, 1.2, and 1.3) are selected and colour-coded. These solutions are then evaluated after cycling at three different beam current levels. The results of this evaluation are presented in Figure 19.

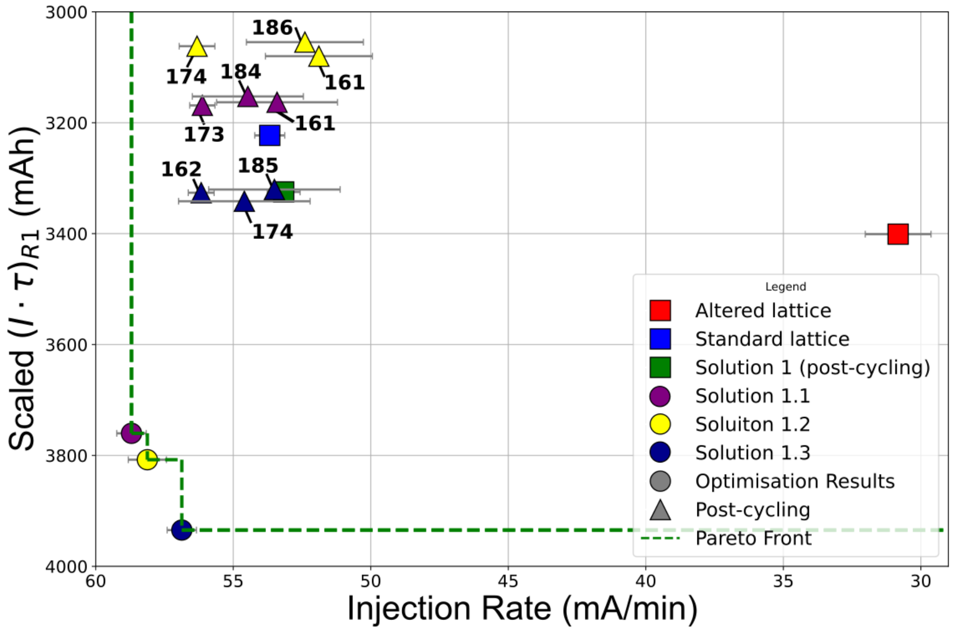

Figure 19: Pareto front (green dotted line) constituted by the selected solutions in Part 2. The optimisation results are represented by circles and their post-cycling evaluation is represented by triangles using the colour assignment detailed in the legend. The standard and altered lattices are represented by blue and red squares, respectively. The solution 1 (post-cycling) is plotted as a green square as the reference point. Bold numbers annotated on the post-cycling solutions represent the R1 beam current (mA) at which their evaluation started.

It is apparent that there is a significant difference between the optimisation results and the post-cycling data, which was also observed in Part 1 of the optimisation. The beam current has no influence on the scaled $(I \cdot \tau)_{R1}$, indicating that the $(I \cdot \tau)_{R1}$ scaling is working as intended. Variations in mean injection rate are well within the statistical error bars, as seen in Figure 19, except for solution 1.2 (post-cycling) at 174 mA. The optimisation run provided promising results (circles). However, after cycling, these solutions performed similarly to the standard lattice and the starting point of this optimisation run, solution 1 (post-cycling).
The SCo configurations of solutions 1, 1.1, 1.2 and 1.3, as well as the standard and altered lattices, are plotted in Figure 20. The precise magnet current values are detailed in the 'Appendix'.

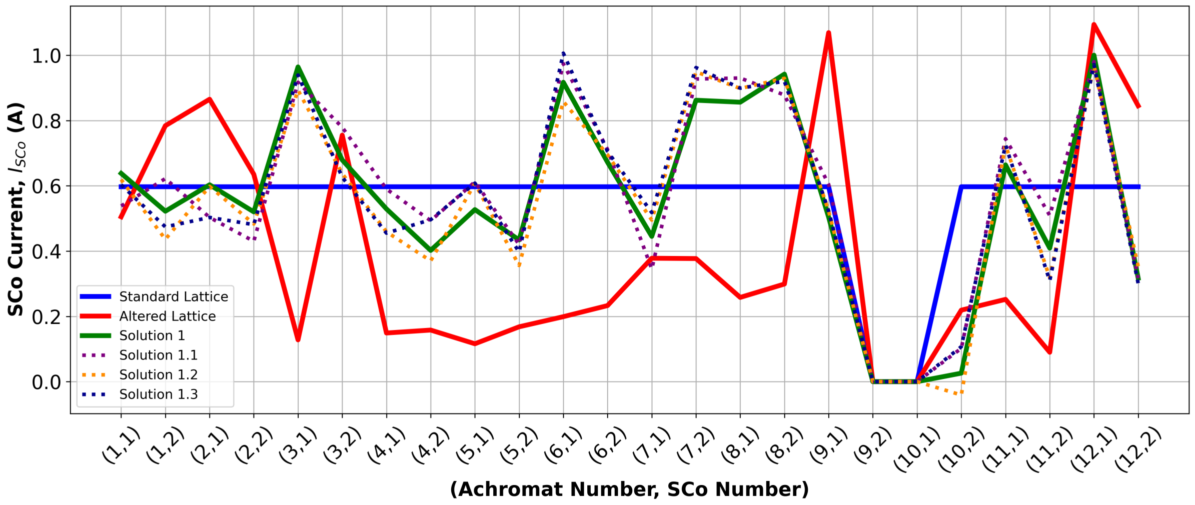

Figure 20: The thick blue and red lines show the SCo configurations of the standard and altered lattices, respectively. The dotted lines are colour-coded, as shown on the legend, and illustrate the SCo configurations of the selected solutions.

Due to the ± 0.1 A limits around solution 1, one can see that solutions 1.1, 1.2 and 1.3 do not deviate too much from the SCo configuration of solution 1. These minor deviations from solution 1 did not significantly improve the machine performance post-cycling. However, unlike the case of altered lattice, where the asymmetric nature caused a significant reduction of the injection rate, there was a significant improvement in the injection rate of the machine compared to the altered lattice. This is an indication that the SCo configuration found by the optimisation restored a good dynamic aperture in the machine, possibly by reducing the amplitude-dependent tune shifts and the consequent crossing of dangerous resonances.

5 Conclusion and Outlook

The main objective of this thesis project was to demonstrate that the performance of the 1.5 GeV storage ring at MAX IV (R1) can be improved using an online multi-objective genetic algorithm approach. Its performance can be measured in terms of its lifetime and injection efficiency; these objectives were actually represented by the scaled $(I \cdot \tau)_{R1}$ and injection rate, respectively. It was demonstrated that the algorithm is capable of finding solutions significantly better than the standard lattice of the machine during the optimisation process. These solutions were better for both objectives, which showed the algorithm's ability to improve two conflicting quantities. It was also shown that the $(I \cdot \tau)_{R1}$ scaling works as intended at different beam currents, decoupling the $(I \cdot \tau)_{R1}$ from the actual beam current. Unfortunately, after cycling the ring, the performance of the solutions suffered quite considerably in terms of the scaled $(I \cdot \tau)_{R1}$. This was observed numerous times, and it is almost certainly due to the magnetic hysteresis. However, even after clearing the magnetic history, the algorithm was able to recover an altered machine and improve its injection rate to the likes of the standard machine, which was about an 80% increase in the injection rate from the altered lattice. As mentioned before, a high injection rate translates into a high injection efficiency if the current provided by the Linac is kept constant. High injection efficiency results in faster injections and reduced unwanted background radiation during these injections; these are valuable improvements for the operations and safety of MAX IV.

Due to time limitations, it was not possible to conduct optimisation runs with a diverse NSGA-II parameter selection; it will be advised to run optimisations with varied mutation rate, mutation intensity, and crossover intensity parameters to explore a more diverse set of solutions. The magnetic hysteresis was deemed the major obstacle in replicating the solutions after cycling the machine. Therefore, an attempt not presented in this thesis was performed to re-create the magnetic history of the solutions with the hope that they could provide the same objective values initially discovered during the optimisation run. Finding a way to mitigate magnetic hysteresis, following the aforementioned strategy or by any other means, could mean that the solutions found during the optimisation can be replicated. Hence, the performance of the standard machine can be improved significantly in terms of lifetime and injection efficiency.

The Python script implementing NSGA-II, the optimiser, can be easily made available to run the same type of optimisation for the 3 GeV storage ring (R3). Moreover, its objective variables can be replaced with other conflicting variables in the rings or potentially in other parts of the facility, *e.g.* the injection transfer lines. The optimiser script is written explicitly for optimising the lifetime and injection efficiency using the NSGA-II method; however, the script can be improved to include other multiple-objective optimisation methods such as particle swarm optimiser (MOPSO) or simulated annealing (MOSA) [17]. As a future development, the structure of the optimisation code can be made more modular, for example, with the use of a graphic user interface.

6 References

[1] M. I. Laboratory, *Max iv laboratory*, [Online]. Available: https://www.maxiv.lu.se/about-us/the-max-iv-facility/. [Accessed: 01-Apr-2024].

[2] I. Martin, M. Apollonio, R. Bartolini, M. Furseman, D. Obee, and G. Bird, "An online multi-objective optimisation package," in *Proceedings of the 8th International Particle Accelerator Conference (IPAC2017)*, paper THPAB153, Copenhagen, Denmark, May 2017, pp. 4092–4095.

[3] M. I. Laboratory, "Max iv detailed design report," MAX IV Laboratory, Lund, Sweden, Tech. Rep., Aug. 2010.

[4] A. Wolski, *Introduction to Beam Dynamics in High-Energy Electron Storage Rings*. San Rafael, CA, USA: Morgan & Claypool Publishers, 2018, DOI: 10.1088/978-1-6817-4989-1.

[5] H. Wiedemann, *Particle Accelerator Physics*, 4th ed. Cham, Switzerland: Springer International Publishing, 2015, ISBN: 978-3-319-18317-6.

[6] J. Bengtsson, *Tracy-3.5: Self-Consistent Charged Particle Beam Dynamics Model*, Available: github.com/jbengtsson/tracy-3.5, 2023, [Online]. Available: https://github.com/jbengtsson/tracy-3.5.

[7] N. Carmignani, *Touschek Lifetime Studies and Optimization of the European Synchrotron Radiation Facility* (Springer Theses). Cham: Springer International Publishing, 2016, ISBN: 978-3-319-25797-6, DOI: 10.1007/978-3-319-25798-3.

[8] P. S. Institut, *Opa - optical parametric amplifier*, https://ados.web.psi.ch/opa/, Accessed: May 7, 2024.

[9] C.-C. Inc., *Armco®pure iron product data bulletin*, 2024, [Online]. Available: https://www.maxiv.lu.se/beamlines-accelerators/accelerators/accelerator-documentation-2/.

[10] S. Sgobba, "Physics and measurements of magnetic materials," in *CAS 2009 - CERN Accelerator School: Magnets, Proceedings*, European Organization for Nuclear Research (CERN), Geneva, Switzerland, 2011.

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002, DOI: 10.1109/4235.996017.

[12] Valdecy, *pyMultiobjective: A Python Library for Multi-objective Optimization*, https://github.com/Valdecy/pyMultiobjective, Accessed: 28-Jan-202, 2023.

[13] *Tango controls*, https://www.tango-controls.org/, Accessed: April 29, 2024.

[14] M. Sjöström, *Personal communication regarding the use of sextupole magnets and feedback systems in R1 (max iv)*, Personal communication, Apr. 2024.

[15] S. Rostami, F. Neri, and K. Gyaurski, "On algorithmic descriptions and software implementations for multi-objective optimisation: A comparative study," *SN Computer Science*, vol. 1, Aug. 2020, DOI: 10.1007/s42979-020-00265-1.

[16] D. Olsson, Å. Andersson, F. Cullinan, P. Tavares, *et al.*, "Commissioning of the bunch-by-bunch feedback system in the max iv 1.5 gev ring," in *2018 9th International Particle Accelerator Conference (IPAC)*, vol. 29, Vancouver, BC, Canada, May 2018.

[17] M. Apollonio, R. T. Fielder, I. P. S. Martin, R. Bartolini, J. Rogers, and G. Henderson, “Online multi-objective optimisation at diamond light source,” in *Proceedings of the 9th International Particle Accelerator Conference (IPAC2018)*, Vancouver, BC, Canada, 2018, DOI: 10.18429/JACoW-IPAC2018-WEPAF054.

7 Appendix

7.1 Scaling Curve Data

Lattice	**Coefficient of** $(I_{R1})^3$	**Coefficient of** $(I_{R1})^2$	**Coefficient of** I_{R1}	**Constant Term**
Standard	-4.492×10^{-9}	7.631×10^{-6}	-0.0003081	0.3385
Altered	-1.027×10^{-8}	1.321×10^{-5}	-0.001958	0.4903
Solution 1	-9.025×10^{-9}	1.063×10^{-5}	-0.0008795	0.3842
Solution 4	-1.524×10^{-8}	1.736×10^{-5}	-0.00323	0.6258

Table 7: Coefficients of cubic fit applied to the $(I \cdot \tau)_{R1}$ scaling curves of the standard lattice, altered lattice, solution 1 and solution 4.

7.2 SCo Configuration Data

This section presents the magnet current readings of the R1 SCo family for different solutions. The noise of the current reading is $\pm 1.6 \times 10^{-5}$ A. ‘Achr. N.’ stands for achromat number, ‘SCo’ shows if the magnet is SCo1 and SCo2 and ‘SCo$_I$ (A)’ shows the current reading of the magnet.

Achr. N.	1		2		3		4		5		6	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.505	0.784	0.865	0.635	0.128	0.755	0.149	0.158	0.116	0.168	0.199	0.233
Achr. N.	7		8		9		10		11		12	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.378	0.377	0.258	0.299	1.069	0.000	0.000	0.219	0.252	0.090	1.094	0.845

Table 8: SCo configuration of altered lattice

Achr. N.	1		2		3		4		5		6	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.638	0.522	0.603	0.520	0.964	0.679	0.529	0.402	0.527	0.434	0.917	0.672
Achr. N.	7		8		9		10		11		12	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.445	0.862	0.856	0.942	0.504	0.000	0.000	0.026	0.664	0.409	1.000	0.317

Table 9: SCo configuration of solution 1

Achr. N.	1		2		3		4		5		6	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.651	0.519	0.460	0.293	0.953	0.744	0.519	0.506	0.392	0.745	0.821	0.702
Achr. N.	7		8		9		10		11		12	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.189	0.091	0.928	0.926	0.194	0.000	0.000	0.009	0.668	0.408	0.436	0.410

Table 10: SCo configuration of solution 2

Achr. N.	1		2		3		4		5		6	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.644	1.000	0.975	0.646	0.395	0.759	0.487	0.109	0.445	0.364	0.987	0.827
Achr. N.	7		8		9		10		11		12	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.313	0.115	0.906	0.804	0.656	0.000	0.000	0.060	0.832	0.435	0.286	0.098

Table 11: SCo configuration of solution 3

Achr. N.	1		2		3		4		5		6	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.644	0.958	0.963	0.651	0.736	0.761	0.454	0.081	0.402	0.766	0.969	0.831
Achr. N.	7		8		9		10		11		12	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.000	0.093	0.901	0.410	0.411	0.000	0.000	0.000	0.493	0.433	0.346	0.367

Table 12: SCo configuration of solution 4

Achr. N.	1		2		3		4		5		6	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.538	0.622	0.503	0.430	0.917	0.779	0.590	0.493	0.612	0.420	0.974	0.711
Achr. N.	7		8		9		10		11		12	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.345	0.927	0.930	0.878	0.604	0.000	0.000	0.102	0.744	0.509	0.980	0.317

Table 13: SCo configuration of solution 1.1

Achr. N.	1		2		3		4		5		6	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.619	0.436	0.598	0.483	0.893	0.637	0.459	0.370	0.607	0.356	0.858	0.695
Achr. N.	7		8		9		10		11		12	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.495	0.949	0.898	0.929	0.522	0.000	0.000	-0.041	0.725	0.318	0.968	0.353

Table 14: SCo configuration of solution 1.2

Achr. N.	1		2		3		4		5		6	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.603	0.474	0.503	0.481	0.939	0.626	0.455	0.496	0.608	0.397	1.006	0.705
Achr. N.	7		8		9		10		11		12	
SCo	1	2	1	2	1	2	1	2	1	2	1	2
SCo$_I$ (A)	0.518	0.962	0.899	0.920	0.523	0.000	0.000	0.107	0.725	0.309	0.971	0.301

Table 15: SCo configuration of solution 1.3

7.3 Optimiser Python Script

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on January 2024

@author: Ali Kuzey Yanartas
"""

#####################################################################

# Import the necessary modules

import statistics
import datetime
import time
import matplotlib.pyplot as plt
from PyTango import DeviceProxy
import concurrent.futures
import shutil
import logging
import copy
import numpy  as np
import random
import os

#####################################################################

# Calling SCo magnets and proxying in

sextupole_SXCO = [
    'R1-D110210CAB01/MAG/PSIE-01', # Achromat 1, SCo 1
    'R1-D110210CAB01/MAG/PSIE-02', # Achromat 1, SCo 2
    'R1-D110210CAB01/MAG/PSIE-03', # Achromat 2, SCo 1
    'R1-D110210CAB01/MAG/PSIE-04', # Achromat 2, SCo 2
    'R1-D110310CAB01/MAG/PSIE-01', # Achromat 3, SCo 1
    'R1-D110310CAB01/MAG/PSIE-02', # Achromat 3, SCo 2
    'R1-D110310CAB01/MAG/PSIE-03', # Achromat 4, SCo 1
    'R1-D110310CAB01/MAG/PSIE-04', # Achromat 4, SCo 2
    'R1-D110610CAB01/MAG/PSIE-01', # Achromat 5, SCo 1
    'R1-D110610CAB01/MAG/PSIE-02', # Achromat 5, SCo 2
    'R1-D110610CAB01/MAG/PSIE-03', # Achromat 6, SCo 1
    'R1-D110610CAB01/MAG/PSIE-04', # Achromat 6, SCo 2
    'R1-D110710CAB01/MAG/PSIE-01', # Achromat 7, SCo 1
    'R1-D110710CAB01/MAG/PSIE-02', # Achromat 7, SCo 2
    'R1-D110710CAB01/MAG/PSIE-03', # Achromat 8, SCo 1
    'R1-D110710CAB01/MAG/PSIE-04', # Achromat 8, SCo 2
    'R1-D110910CAB01/MAG/PSIE-01', # Achromat 9, SCo 1
    'R1-D110910CAB01/MAG/PSIE-04', # Achromat 10, SCo 2
    'R1-D111210CAB01/MAG/PSIE-01', # Achromat 11, SCo 1
    'R1-D111210CAB01/MAG/PSIE-02', # Achromat 11, SCo 2
    'R1-D111210CAB01/MAG/PSIE-03', # Achromat 12, SCo 1
    'R1-D111210CAB01/MAG/PSIE-04'] # Achromat 12, SCo 2

sextupole_SXCO_proxy = [DeviceProxy(dev) for dev in sextupole_SXCO]

```

```
#Achromat 9, SCo 2 and achromat 10 SCo 1 are excluded from the optimisation
    .

##################################################################

# Reading from SCo magnets

logging.basicConfig(level=logging.ERROR)

def read_magnet(magnet_device, max_retries=3):
    for _ in range(max_retries):
        try:
            magnet_device_proxy = DeviceProxy(magnet_device)
            magnet_current = magnet_device_proxy.read_attribute("Current").
    value
            return magnet_current
        except TimeoutError as e:

            logging.error(f"Timeout reading {magnet_device}: {e}")
        except Exception as e:

            logging.error(f"Error reading {magnet_device}: {e}")

        time.sleep(1)


    logging.error(f"All retries failed for {magnet_device}. Returning None.
    ")
    return None

def read_family():
    magnet_devices = sextupole_SXCO_proxy
    with concurrent.futures.ThreadPoolExecutor() as executor:
        results = list(executor.map(read_magnet, magnet_devices))
        print("Readings from magnet devices:")
        for device, result in zip(sextupole_SXCO, results):
            print(f'({device}, {result})')

##################################################################

# Recording the magnet values before the optimisation starts

magnet_values_SXCO = [(dev, read_magnet(dev)) for dev in sextupole_SXCO]

all_magnet_names = [name for name, _ in magnet_values_SXCO]
all_magnet_values = [str(value) for _, value in magnet_values_SXCO]

#Step: Record the machine lattice before the optimisation

with open('R1_preopt_data.txt', 'w') as f:
        # Write magnet names as titles
    f.write(', '.join(all_magnet_names) + ',injection_rate,scaled_I_Tau,
    std_dev_inj,std_dev_I_Tau,current_r1\n')
        # Write magnet values
    f.write(', '.join(all_magnet_values))

##################################################################

```

```python
# Writing to SCo magnets

def write_family(change, max_retries = 3):
    for i in range(len(magnet_values_SXCO)):
        new_value = change[i]
        dev = magnet_values_SXCO[i][0]
        sextupole_SDE_proxy = DeviceProxy(dev)

        for _ in range(max_retries):
            try:
                sextupole_SDE_proxy.write_attribute("Current", new_value)
                break  # Success, exit the loop
            except TimeoutError as e:

                logging.error(f"Timeout writing {dev}: {e}")
            except Exception as e:

                logging.error(f"Error writing {dev}: {e}")

            time.sleep(1)


###################################################################

# Reading Scrapers

r1_scraper_01V_proxy = DeviceProxy("r1-101s/VAC/scrp-01-v")
r1_scraper_02V_proxy = DeviceProxy("r1-101s/VAC/scrp-02-v")
r1_scraper_03H_proxy = DeviceProxy("r1-101s/VAC/scrp-03-h")

def read_r1_scrapers():
    try:
        scraper_01V = r1_scraper_01V_proxy.read_attribute('position').value
        scraper_02V = r1_scraper_02V_proxy.read_attribute('position').value
        scraper_03H = r1_scraper_03H_proxy.read_attribute('position').value
        return scraper_01V, scraper_02V, scraper_03H
    except TimeoutError as e:

        logging.error(f"Timeout reading scrapers: {e}")
    except Exception as e:

        logging.error(f"Error reading scrapers: {e}")
    return None, None, None


###################################################################

# Scaled I*Tau calculation

lifetime_device_proxy = DeviceProxy("r1-101s/dia/dcct-01")
current_device_proxy = DeviceProxy("r1-101s/dia/dcct-01")

#must write the coefficients of the scaling curve on a .txt file, one row
   per coefficient

def read_coefficients_from_file(filename='Scaling_factor_coefficients.txt')
   :
    with open(filename, 'r') as file:
        coefficients = [float(line.strip()) for line in file.readlines()]
    return coefficients

coefficients = read_coefficients_from_file()

def cubic_fit_scaling_factor(I):
    a, b, c, d = coefficients
    scaling_factor = a*I**3 + b*I**2 + c*I + d
    return scaling_factor

def read_lifetime_r1():
    try:
        lifetime_r1 = lifetime_device_proxy.read_attribute("Lifetime").value / 3600
        return lifetime_r1
    except TimeoutError as e:

        logging.error(f"Timeout reading lifetime: {e}")
    except Exception as e:

        logging.error(f"Error reading lifetime: {e}")
    return None

def read_ring_current_r1():
    try:
        current_r1 = current_device_proxy.read_attribute("Current").value * 1000
        return current_r1
    except TimeoutError as e:

        logging.error(f"Timeout reading ring current: {e}")
    except Exception as e:

        logging.error(f"Error reading ring current: {e}")
    return None

def calculate_I_Tau(lifetime_r1, current_r1):
    I_Tau = current_r1 * lifetime_r1
    return I_Tau

def measure_I_Tau_values(number_of_measurements=10, time_between_measurements=0.5):
    I_Tau_values = []

    for i in range(number_of_measurements):
        lifetime_r1 = read_lifetime_r1()
        current_r1 = read_ring_current_r1()

        if lifetime_r1 is None or current_r1 is None:
            logging.error("Failed to obtain necessary readings. Skipping measurement.")
            continue  # Skip this measurement if either value is None

        scaling_factor = cubic_fit_scaling_factor(current_r1)

        result = calculate_I_Tau(lifetime_r1, current_r1)/(abs(scaling_factor)-1e-5)
        I_Tau_values.append(result)
        time.sleep(time_between_measurements)

    mean_I_Tau = statistics.mean(I_Tau_values)
    std_deviation_I_Tau = statistics.stdev(I_Tau_values)

    return mean_I_Tau, std_deviation_I_Tau


# mean_I_Tau, std_deviation_I_Tau = measure_I_Tau_values()
# print(f'Mean I_Tau: {mean_I_Tau} mAh +/- {std_deviation_I_Tau}.')
# print('-' * 30)


##################################################################

# Reading the Injection Rate

injection_device_proxy = DeviceProxy("R1/DIA/InjectionMeter")

def read_injection_rate(number_of_measurements=10, time_between_measurements=0.5):
    injection_rate_values = []

    for i in range(number_of_measurements):
        try:
            result = injection_device_proxy.read_attribute("InjectionRateMapm").value
            injection_rate_values.append(result)
        except TimeoutError as e:

            logging.error(f"Timeout reading injection rate: {e}")
        except Exception as e:

            logging.error(f"Error reading injection rate: {e}")
        time.sleep(time_between_measurements)

    mean_injection_rate = statistics.mean(injection_rate_values)
    std_deviation_injection_rate = statistics.stdev(injection_rate_values)

    return mean_injection_rate, std_deviation_injection_rate


# mean_injection_rate, std_deviation_injection_rate = read_injection_rate()
# print(f'Mean injection rate: {mean_injection_rate} mA/min +/- {std_deviation_injection_rate}.')
# print('-' * 30)

##################################################################

# Injection to R1

stm_proxy = DeviceProxy("g/ctl/stm-01")

def write_r1_injection(max_current):
    try:
        stm_proxy.write_attribute("R1MaxCurrent", max_current)
        stm_proxy.command_inout("InjectR1")
        print(f"Injecting R1 to {max_current} mA!")
        print(30*'-')
    except TimeoutError as e:

        logging.error(f"Timeout writing R1 injection: {e}")
    except Exception as e:

        logging.error(f"Error writing R1 injection: {e}")

def stop_r1_injection():
    try:
        stm_proxy.command_inout("StopInjectR1")
        print("R1 Injection Stopped!")
        print(30*'-')
    except TimeoutError as e:

        logging.error(f"Timeout stopping R1 injection: {e}")
    except Exception as e:

        logging.error(f"Error stopping R1 injection: {e}")

def read_stm_status():
    try:
        stm_state = stm_proxy.read_attribute("CurrentState").value
        return stm_state
    except TimeoutError as e:

        logging.error(f"Timeout reading STM status: {e}")
    except Exception as e:

        logging.error(f"Error reading STM status: {e}")
    return None


###################################################################

# Measuring the machine performance before the optimisation

mean_I_Tau, std_deviation_I_Tau = measure_I_Tau_values(
    number_of_measurements=10, time_between_measurements=0.5)

print(f'Scaled I_Tau (Initial measurement): {mean_I_Tau} mAh +/- {
    std_deviation_I_Tau}.')

current_r1 = read_ring_current_r1()

write_r1_injection(500)

time.sleep(20)

mean_injection_rate, std_deviation_injection_rate = read_injection_rate(
    number_of_measurements=10, time_between_measurements=0.5)

print(f'Mean injection rate (Initial measurement): {mean_injection_rate} mA
    /min +/- {std_deviation_injection_rate}.')

stop_r1_injection()

with open('R1_preopt_data.txt', 'a') as f:
    f.write(f" ,{mean_injection_rate}, {mean_I_Tau}, {std_deviation_injection_rate}, {std_deviation_I_Tau}, {current_r1}\n")
shutil.copy('R1_preopt_data.txt',f'R1_preopt_data_{datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")}.txt')

###################################################################

# Objective functions: Scaled I*Tau and Injection Rate

with open('R1_opt_raw_data.txt', 'w') as f:
    f.write(', '.join(all_magnet_names) + "I_Tau, I_Tau_std_dev, injection_rate, injection_rate_std_dev, current_r1, time\n")

def I_Tau_objective(change):

    print(30*'-')

    print('Scaled I_Tau objective running...')

    time.sleep(5)

    write_family(change)

    time.sleep(50)

    mean_I_Tau, std_deviation_I_Tau = measure_I_Tau_values(number_of_measurements=10, time_between_measurements=0.5)

    print(f'Scaled I_Tau: {mean_I_Tau} mAh +/- {std_deviation_I_Tau}.')

    current_r1 = read_ring_current_r1()

    magnet_values_SXCO = [(dev, read_magnet(dev)) for dev in sextupole_SXCO]

    all_magnet_values = [str(value) for _, value in magnet_values_SXCO]

    with open('R1_opt_raw_data.txt', 'a') as f:
        f.write(', '.join(all_magnet_values) + f", {mean_I_Tau}, {std_deviation_I_Tau}")

    current_r1 = read_ring_current_r1()
    while current_r1 > 450:
        user_input = input("Ring current is over 450 mA, turn off BBB, scrape down to 250 mA, turn on BBB and pull out all the scrapers. Are you ready to continue ? (yes/no): ")
        if user_input.lower() == "yes":
            print("Algorithm continues.")
            break

    current_r1 = read_ring_current_r1()
    while current_r1 < 240:
        user_input = input("Ring current is under 250 mA, inject up to at least 250 mA. Are you ready to continue? (yes/no): ")
        if user_input.lower() == "yes":
            print("Algorithm continues.")
            break

    scraper_01V, scraper_02V, scraper_03H = read_r1_scrapers()
    while scraper_01V < 29 or scraper_02V < 30.1 or scraper_03H < 29.5 :
        user_input = input("The scrapers are not fully pulled out, pull out the scrapers. Are you ready to continue? (yes/no): ")
        if user_input.lower() == "yes":
            print("Algorithm continues.")
            break

    stm_state = read_stm_status()
    while stm_state == 'R3' :
        user_input = input("Injecting to R3, please wait! Are you ready to continue? (yes/no): ")
        if user_input.lower() == "yes":
            print("Algorithm continues.")
            break

    return -mean_I_Tau

def Injection_Rate_objective(change):

    print(30*'-')

    print('Injection Rate objective running...')

    time.sleep(5)

    write_r1_injection(500)

    time.sleep(20)

    mean_injection_rate, std_deviation_injection_rate = read_injection_rate(number_of_measurements=10, time_between_measurements=0.5)

    print(f'Mean injection rate: {mean_injection_rate} mA/min +/- {std_deviation_injection_rate}.')

    stop_r1_injection()

    current_r1 = read_ring_current_r1()

    with open('R1_opt_raw_data.txt', 'a') as f:
        f.write(f' ,{mean_injection_rate}, {std_deviation_injection_rate}, {current_r1}, {datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")}\n')

    while current_r1 > 450:
        user_input = input("Ring current is over 450 mA, turn off BBB, scrape down to 250 mA, turn on BBB and pull out all the scrapers. Are you ready to continue ? (yes/no): ")
        if user_input.lower() == "yes":
            print("Algorithm continues.")
            break

    current_r1 = read_ring_current_r1()
    while current_r1 < 240:
        user_input = input("Ring current is under 250 mA, inject up to at least 250 mA. Are you ready to continue? (yes/no): ")
        if user_input.lower() == "yes":
            print("Algorithm continues.")
            break

    scraper_01V, scraper_02V, scraper_03H = read_r1_scrapers()
    while scraper_01V < 30.1 or scraper_02V < 30.1 or scraper_03H < 29.5 :
        user_input = input("The scrapers are not fully pulled out, pull out the scrapers. Are you ready to continue? (yes/no): ")
        if user_input.lower() == "yes":
            print("Algorithm continues.")
            break

    stm_state = read_stm_status()
    while stm_state == 'R3' :
        user_input = input("Injecting to R3, please wait! Are you ready to continue? (yes/no): ")
        if user_input.lower() == "yes":
            print("Algorithm continues.")
            break

    return -mean_injection_rate


###################################################################

# Functions required for the NSGA-II algorithm

# Function: Initialize Variables, function adapted from Valdecy, refer to [12] in the references section.
def initial_population(population_size = 5, min_values = [-5,-5], max_values = [5,5], list_of_functions = [Injection_Rate_objective, I_Tau_objective]):
    population = np.zeros((population_size, len(min_values) + len(list_of_functions)))
    for i in range(0, population_size):
        for j in range(0, len(min_values)):
             population[i,j] = random.uniform(min_values[j], max_values[j])
        for k in range (1, len(list_of_functions) + 1):
            population[i,-k] = list_of_functions[-k](list(population[i,0:population.shape[1]-len(list_of_functions)]))
    return population



# Function: Fast Non-Dominated Sorting, function adapted from Valdecy, refer to [12] in the references section.
def fast_non_dominated_sorting(population, number_of_functions = 2):
    S     = [[] for i in range(0, population.shape[0])]
    front = [[]]
    n     = [0 for i in range(0, population.shape[0])]
    rank  = [0 for i in range(0, population.shape[0])]
    for p in range(0, population.shape[0]):
        S[p] = []
        n[p] = 0
        for q in range(0, population.shape[0]):
            if ((population[p,-number_of_functions:] <= population[q,-number_of_functions:]).all()):
                if (q not in S[p]):
                    S[p].append(q)
            elif ((population[q,-number_of_functions:] <= population[p,-number_of_functions:]).all()):
                n[p] = n[p] + 1
        if (n[p] == 0):
            rank[p] = 0
            if (p not in front[0]):
                front[0].append(p)
    i = 0
    while (front[i] != []):
        Q = []
        for p in front[i]:
            for q in S[p]:
                n[q] = n[q] - 1
                if(n[q] == 0):
                    rank[q] = i+1
                    if q not in Q:
                        Q.append(q)
        i = i+1
        front.append(Q)
    del front[len(front)-1]
    rank = np.zeros((population.shape[0], 1))
    for i in range(0, len(front)):
        for j in range(0, len(front[i])):
            rank[front[i][j], 0] = i + 1
    return rank

# Function: Sort Population by Rank, function adapted from Valdecy, refer to [12] in the references section.
def sort_population_by_rank(population, rank):
    idx        = np.argsort(rank[:,0], axis = 0).tolist()
    rank       = rank[idx,:]
    population = population[idx,:]
    return population, rank

# Function: Crowding Distance (Adapted from PYMOO), function adapted from Valdecy, refer to [12] in the references section.
def crowding_distance_function(pop, M):
    infinity   = 1e+11
    population = copy.deepcopy(pop[:,-M:])
    population = population.reshape((pop.shape[0], M))
    if (population.shape[0] <= 2):
        return np.full(population.shape[0], infinity)
    else:
        arg_1      = np.argsort(population, axis = 0, kind = 'mergesort')
        population = population[arg_1, np.arange(M)]
        dist       = np.concatenate([population, np.full((1, M), np.inf)]) - np.concatenate([np.full((1, M), -np.inf), population])
        idx        = np.where(dist == 0)
        a          = np.copy(dist)
        b          = np.copy(dist)
        for i, j in zip(*idx):
            a[i, j] = a[i - 1, j]
        for i, j in reversed(list(zip(*idx))):
            b[i, j] = b[i + 1, j]
        norm            = np.max(population, axis = 0) - np.min(population, axis = 0)
        norm[norm == 0] = np.nan
        a, b            = a[:-1]/norm, b[1:]/norm
        a[np.isnan(a)]  = 0.0
        b[np.isnan(b)]  = 0.0
        arg_2           = np.argsort(arg_1, axis = 0)
        crowding        = np.sum(a[arg_2, np.arange(M)] + b[arg_2, np.arange(M)], axis = 1) / M
    crowding[np.isinf(crowding)] = infinity
    crowding                     = crowding.reshape((-1,1))
    return crowding

# Function:Crowded Comparison Operator, function adapted from Valdecy, refer to [12] in the references section.
def crowded_comparison_operator(rank, crowding_distance, individual_1 = 0, individual_2 = 1):
    selection = False
    if (rank[individual_1,0] < rank[individual_2,0]) or ((rank[individual_1,0] == rank[individual_2,0]) and (crowding_distance[individual_1,0] > crowding_distance[individual_2,0])):
        selection = True
    return selection

# Function: Offspring, function adapted from Valdecy, refer to [12] in the references section.
def breeding(population, rank, crowding_distance, min_values = [-5,-5], max_values = [5,5], mu = 1, list_of_functions = [Injection_Rate_objective, I_Tau_objective]):
    offspring   = np.copy(population)
    parent_1    = 0
    parent_2    = 1
    b_offspring = 0
    for i in range (0, offspring.shape[0]):
        i1, i2, i3, i4 = random.sample(range(0, len(population) - 1), 4)
        if (crowded_comparison_operator(rank, crowding_distance, individual_1 = i1, individual_2 = i2) == True):
            parent_1 = i1
        elif (crowded_comparison_operator(rank, crowding_distance, individual_1 = i2, individual_2 = i1) == True):
            parent_1 = i2
        else:
            rand = int.from_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)
            if (rand > 0.5):
                parent_1 = i1
            else:
                parent_1 = i2
        if (crowded_comparison_operator(rank, crowding_distance, individual_1 = i3, individual_2 = i4) == True):
            parent_2 = i3
        elif (crowded_comparison_operator(rank, crowding_distance, individual_1 = i4, individual_2 = i3) == True):
            parent_2 = i4
        else:
            rand = int.from_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)
            if (rand > 0.5):
                parent_2 = i3
            else:
                parent_2 = i4
        for j in range(0, offspring.shape[1] - len(list_of_functions)):
```

```python
            rand   = int.from_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)
            rand_b = int.from_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)
            rand_c = int.from_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)
            if (rand <= 0.5):
                b_offspring = 2*(rand_b)
                b_offspring = b_offspring**(1/(mu + 1))
            elif (rand > 0.5):
                b_offspring = 1/(2*(1 - rand_b))
                b_offspring = b_offspring**(1/(mu + 1))
            if (rand_c >= 0.5):
                offspring[i,j] = np.clip(((1 + b_offspring)*population[parent_1, j] + (1 - b_offspring)*population[parent_2, j])/2, min_values[j], max_values[j])
            else:
                offspring[i,j] = np.clip(((1 - b_offspring)*population[parent_1, j] + (1 + b_offspring)*population[parent_2, j])/2, min_values[j], max_values[j])
        for k in range (1, len(list_of_functions) + 1):
            offspring[i,-k] = list_of_functions[-k](offspring[i,0:offspring.shape[1]-len(list_of_functions)])
    return offspring

# Function: Mutation, function adapted from Valdecy, refer to [12] in the references section.
def mutation(offspring, mutation_rate = 0.1, eta = 1, min_values = [-5,-5], max_values = [5,5], list_of_functions = [Injection_Rate_objective, I_Tau_objective]):
    d_mutation = 0
    for i in range (0, offspring.shape[0]):
        for j in range(0, offspring.shape[1] - len(list_of_functions)):
            probability = int.from_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)
            if (probability < mutation_rate):
                rand   = int.from_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)
                rand_d = int.from_bytes(os.urandom(8), byteorder = 'big') / ((1 << 64) - 1)
                if (rand <= 0.5):
                    d_mutation = 2*(rand_d)
                    d_mutation = d_mutation**(1/(eta + 1)) - 1
                elif (rand > 0.5):
                    d_mutation = 2*(1 - rand_d)
                    d_mutation = 1 - d_mutation**(1/(eta + 1))
                offspring[i,j] = np.clip((offspring[i,j] + d_mutation), min_values[j], max_values[j])
        for k in range (1, len(list_of_functions) + 1):
            offspring[i,-k] = list_of_functions[-k](offspring[i,0:offspring.shape[1]-len(list_of_functions)])
    return offspring

############################################################################

# Plotting generations during the optimisation and writing solutions to file

```

```python
def plot_population_and_offspring(population, offspring, generation):
    pop_injection_rate = population[:, -2]
    pop_I_Tau = population[:, -1]

    off_injection_rate = offspring[:, -2]
    off_I_Tau = offspring[:, -1]

    plt.figure(figsize=(10, 6))
    plt.scatter(pop_injection_rate, pop_I_Tau, color='blue', label='Population', alpha=0.6, edgecolor='k')
    plt.scatter(off_injection_rate, off_I_Tau, color='red', label='Offspring', alpha=0.6, edgecolor='k')
    plt.title(f'Generation {generation}: I_Tau vs Injection Rate')
    plt.xlabel('Injection Rate')
    plt.ylabel('I_Tau')
    plt.legend()
    plt.grid(True)
    plt.show()


def write_population_to_file(file, pop, gen, pop_type, all_magnet_names, list_of_functions):
    rank = fast_non_dominated_sorting(pop, number_of_functions=len(list_of_functions))
    for individual in range(pop.shape[0]):
        rank_val = int(rank[individual, 0])
        var_vals = pop[individual, :-len(list_of_functions)]
        obj_vals = pop[individual, -len(list_of_functions):]
        file.write(f'{gen}, {rank_val}, {pop_type}, {obj_vals[0]}, {obj_vals[1]}, ' + ', '.join(map(str, var_vals)) + '\n')
    file.flush()  # Force flush the buffer to file after each write operation

###################################################################

# The main NSGA-II function, function adapted from Valdecy, refer to [12] in the references section.

def non_dominated_sorting_genetic_algorithm_II(population_size = [], mutation_rate = [], min_values = [], max_values = [], list_of_functions = [Injection_Rate_objective, I_Tau_objective], generations = [], mu = [], eta = [], verbose = True, all_magnet_names=[]):
    count      = 0
    population = initial_population(population_size = population_size, min_values = min_values, max_values = max_values, list_of_functions = list_of_functions)
    offspring  = initial_population(population_size = population_size, min_values = min_values, max_values = max_values, list_of_functions = list_of_functions)

    # Plot initial population and offspring
    plot_population_and_offspring(population, offspring, 'Initial')

    with open('R1_opt_data.txt', 'w') as file:
        file.write('generation,rank,type,injection_rate,scaled_I_Tau,' + ', '.join(all_magnet_names) + '\n')

        # Initial population write
        write_population_to_file(file, population, count, 'population', all_magnet_names, list_of_functions)
        write_population_to_file(file, offspring, count, 'offspring', all_magnet_names, list_of_functions)

    while (count <= generations):
        if (verbose == True):
            print('Generation = ', count)
        population         = np.vstack([population, offspring])
        rank               = fast_non_dominated_sorting(population, number_of_functions = len(list_of_functions))
        population, rank  = sort_population_by_rank(population, rank)
        population, rank  = population[0:population_size,:], rank[0:population_size,:]
        print(population, rank)
        crowding_distance = crowding_distance_function(population, len(list_of_functions))
        offspring          = breeding(population, rank, crowding_distance, mu = mu, min_values = min_values, max_values = max_values, list_of_functions = list_of_functions)
        offspring          = mutation(offspring, mutation_rate = mutation_rate, eta = eta, min_values = min_values, max_values = max_values, list_of_functions = list_of_functions)
        print(offspring)
        with open('R1_opt_data.txt', 'a') as file:
            write_population_to_file(file, population, count, 'population', all_magnet_names, list_of_functions)
            write_population_to_file(file, offspring, count, 'offspring', all_magnet_names, list_of_functions)

        plot_population_and_offspring(population, offspring, count)

        count              = count + 1

    return population

##################################################################

#Calling the algorithm function

non_dominated_sorting_genetic_algorithm_II()

##################################################################

#Adding date to the data files

shutil.copy('R1_opt_raw_data.txt',f'R1_opt_raw_data_{datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")}.txt')
shutil.copy('R1_opt_data.txt',f'R1_opt_data_{datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")}.txt')

##################################################################
```

Listing 1: Python script adapting NSGA-II to optimise R1 performance using the SCo magnets.