HAW HAMBURG

BACHELOR THESIS
Adel Toor Ahmad

Einfluss von RNGs auf die Effizienz von evolutionären Algorithmen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Adel Toor Ahmad

Einfluss von RNGs auf die Effizienz von evolutionären Algorithmen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Christian Lins
Zweitgutachter: Prof. Dr. Phillip Jenke

Eingereicht am: 29. November 2023

Adel Toor Ahmad

Thema der Arbeit

Einfluss von RNGs auf die Effizienz von evolutionären Algorithmen

Stichworte

Evolutionäre Algorithmen, Differential Evolution Algorithmus, Optimierungsalgorithmen, Zufallszahlengenerator, Pseudozufallszahlengenerator

Kurzzusammenfassung

Diese Arbeit untersucht die Auswirkungen unterschiedlicher Pseudozufallszahlengeneratoren (PRNGs) auf die Effizienz evolutionärer Algorithmen. Das Experiment verwendet ein praktisches Optimierungsproblem, bei dem der Differential Evolution Algorithmus eingesetzt wird, um eine sinusförmige Funktion an Daten eines Handgelenksensors anzupassen. Ziel ist die Vorhersage der Kompressionstiefe und Kompressionsfrequenz einer Herz-Lungen-Wiederbelebung. Die Studie analysiert, ob signifikante Unterschiede in der Genauigkeit der Vorhersagen auftreten, abhängig von der Wahl des PRNGs. Diese Analyse wird anhand von sechs verschiedenen Subjekten (Datensätzen) durchgeführt. Die Ergebnisse zeigen, dass der Differential Evolution Algorithmus in diesem Anwendungsfall robust gegenüber der Wahl des PRNGs ist.

Adel Toor Ahmad

Title of Thesis

Influence of RNGs on the Efficiency of Evolutionary Algorithms

Keywords

Evolutionary Computation, Evolutionary Algorithms, Differential Evolution Algorithm, Random Number Generator, Pseudorandom Number Generator

Abstract

This study investigates the impact of different Pseudo-Random Number Generators (PRNGs) on the efficiency of evolutionary algorithms. The experiment utilizes a practical

optimization problem, employing the Differential Evolution Algorithm to fit a sinusoidal function to data from a wrist sensor. The objective is to predict the compression depth and compression frequency of cardiopulmonary resuscitation. The study analyzes whether significant differences in prediction accuracy arise based on the choice of PRNG. This analysis is conducted across six different subjects (datasets). The results demonstrate that, in this particular application, the Differential Evolution Algorithm proves to be robust regardless of the chosen PRNG.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

CCD Chest Compression Depth.

CCF Chest Compression Frequency.

CSTI Creative Space for Technical Innovations.

EA Evolutionäre Algorithmen.

KWT Kruskal Wallis Test.

PRNG Pseudo Random Number Generator.

RMSE Root Mean Squared Error.

RNG Random Number Generator.

SFTP Secure File Transfer Protocol.

SSH Secure Shell.

TRNG True Random Number Generator.

VPN Virtual Private Network.

Quellcodeverzeichnis

1 Einleitung

1.1 Motivation

Die Anwendung evolutionärer Algorithmen hat sich als äußerst erfolgreich bei der Lösung komplexer Optimierungsprobleme in verschiedenen Bereichen erwiesen [7]. Evolutionäre Algorithmen sind heuristische Optimierungstechniken, die von den Prinzipien der biologischen Evolution inspiriert sind, bei der eine Population potentieller Lösungen durch genetische Operationen wie Mutation und Rekombination weiterentwickelt wird [10]. Die Qualität und Vielfalt dieser Lösungen ist entscheidend für eine effiziente Exploration des Suchraums [10].

Der Zufallszahlengenerator (RNG) dient der Erzeugung von Zufallszahlen [12]. Insbesondere Pseudozufallszahlengeneratoren PRNGs werden häufig in Simulationen eingesetzt, da sie einfach zu implementieren und schnell sind [12]. Zufallszahlen werden in evolutionären Algorithmen verwendet, um Variationen in Lösungen zu erzeugen und genetische Operationen wie Rekombination und Mutation durchzuführen [8]. Damit ist der RNG ein Schlüsselelement in evolutionären Algorithmen.

In den letzten Jahren wurden verschiedene RNGs entwickelt, die sich in ihren statistischen Eigenschaften und ihrem Verhalten unterscheiden. Es stellt sich daher die Frage, ob die Wahl des RNGs Auswirkungen auf die Konvergenzgeschwindigkeit, die Qualität der gefundenen Lösungen und die Robustheit des Algorithmus hat.

1.2 Ziel

Die vorliegende Bachelorarbeit beschäftigt sich mit der Analyse des Einflusses von Zufallszahlengeneratoren (RNGs) auf die Effizienz von evolutionären Algorithmen. Effizienz

wird in diesem Zusammenhang definiert als die Konvergenzgeschwindigkeit, mit der Lösungen gefunden werden (gemessen in der Anzahl an benötigten Generationen), sowie die Qualität der gefundenen Lösungen.

Es ist daher zu untersuchen, ob die Wahl des RNGs Einfluss auf die Konvergenzgeschwindigkeit und die Qualität der gefundenen Lösungen hat. Für dieser Analyse werden in dieser Arbeit die RNGs Mersenne Twister, KISS und Xorshift vorgestellt. Diese RNGs wurden ausgewählt, weil sie sich in der Art der Zahlengenerierung und der Periodenlänge unterscheiden.

Der evolutionäre Algorithmus, der in dieser Studie untersucht wird, ist der Differential Evolution Algorithmus. In der Theorie zeigt dieser Algorithmus eine hohe Sensitivität gegenüber der Wahl des RNGs [5]. Es ist daher von Interesse zu untersuchen, ob diese Sensitivität auch in der Praxis beobachtet werden kann. Daher wird anhand eines praktischen Optimierungsproblems untersucht, ob die Wahl des RNGs einen signifikanten Einfluss auf die Effizienz des Differential Evolution Algorithmus hat.

1.3 Aufbau der Arbeit

Im Abschnitt 2 werden die theoretischen Grundlagen von Optimierungsproblemen, Evolutionären Algorithmen, Zufallszahlengeneratoren und dem Differential Evolution Algorithmus ausführlich erläutert. Neben einer detaillierten Vorstellung dieser Grundlagen wird auch auf einschlägige Forschungsarbeiten eingegangen, die einen direkten Bezug zu den Fragestellungen dieser Studie aufweisen. Da die vorliegende Arbeit den Unterschied anhand eines praktischen Anwendungsfalls untersucht, wird ein konkretes Optimierungsproblem vorgestellt, welches einen realen Anwendungsfall repräsentiert. Neben einer eingehenden Problembeschreibung wird auch auf die spezifische Implementierung des Differential Evolution Algorithmus für dieses Optimierungsproblem und die genutzten Zufallszahlengeneratoren (RNGs) im Detail eingegangen. Nach der Erläuterung des praktischen Aufbaus erfolgt eine kurze Darstellung der statistischen Analyse, insbesondere hinsichtlich der Anwendung des KWT zur Identifikation signifikanter Unterschiede in den Daten. Im Anschluss an dieses Kapitel wird die Implementierung genauer beleuchtet. Hierbei wird gezeigt, wie der Algorithmus als Code umgesetzt und angepasst wird, sowie wie die Ergebnisse gespeichert und verarbeitet werden, um sie im Kapitel 4 angemessen präsentieren zu können.

Im Kapitel 4 werden die erzielten Ergebnisse dargestellt. Es werden graphische Darstellungen von Mittelwert, Median und Varianz verwendet. Zusätzlich zu den graphischen Darstellungen werden die p-Werte des zuvor im Kapitel 3 erwähnten KWT für den Mittelwert, den Median und die Varianz tabellarisch dargestellt. Neben diesen Ergebnissen wird auch angegeben, wie viele Zahlen die RNGs im Durchschnitt für jeden Datensatz erzeugt haben und wie viele Generationen benötigt wurden, um eine optimale Lösung zu finden. Das Kapitel 5 widmet sich der ausführlichen Diskussion der Ergebnisse aus dem Kapitel 4, um die zentrale These zu beantworten, ob signifikante Unterschiede auftreten, wenn unterschiedliche RNGs verwendet werden. Neben der Interpretation der Ergebnisse werden weitere Tests zur Klärung offener Fragen durchgeführt.

Abschließend bietet das Kapitel 6 eine kurze Zusammenfassung und Reflexion der gesamten Arbeit, wobei die wichtigsten Erkenntnisse nochmals hervorgehoben werden, um die zentrale These dieser Arbeit zu beantworten. Das Kapitel schließt mit einem Ausblick, der Anregungen für weitere Forschungsarbeiten in diesem Themenfeld gibt.

2 Stand der Technik

2.1 Optimierungsprobleme

Evolutionäre Algorithmen eignen sich besonders gut für Optimierungsprobleme [15]. Um jedoch genau zu verstehen, warum dies der Fall ist, muss zunächst geklärt werden, was unter Optimierungsproblemen zu verstehen ist.

2.1.1 Definition

Optimierungsprobleme bestehen aus zwei essenziellen Komponenten: einem definierten Suchraum, der als S bezeichnet wird, und einer zugehörigen Zielfunktion, die als f : S dargestellt wird [10]. Der Suchraum S bildet die Grundlage für die Menge aller potenziellen Lösungen des Problems, während die Zielfunktion f die Bewertungs- oder Bewertungskriterien für diese Lösungen festlegt [10]. Das Hauptziel bei Optimierungsproblemen besteht in der Regel darin, entweder die Zielfunktion zu maximieren oder zu minimieren, abhängig von den spezifischen Anforderungen und Zielen des jeweiligen Problems [10].

2.1.2 Beispiel

Ein einfaches Optimierungsproblem besteht darin, Tief- oder Hochpunkte in einer Funktion zu finden. In diesem Kontext erstreckt sich der Suchraum über alle reellen Zahlen, die als x-Werte für die Funktion $f(x) = x^2$ verwendet werden können. Die Funktion selbst bewertet diese x-Werte und gibt entsprechende y-Werte aus. Das Hauptziel bei diesem Problem besteht darin, die reelle Zahl zu identifizieren, die den kleinsten oder größten y-Wert erzeugt. In diesem Zusammenhang gilt die grundlegende Regel bei Optimierungsproblemen, dass die Eingabe gesucht wird, während die Ausgabe bereits bekannt ist [8].

2.1.3 Großer Suchraum

Eine Herausforderung bei Optimierungsproblemen besteht oft darin, dass der Suchraum, in dem nach der besten Lösung gesucht wird, zu groß sein kann [10]. Ein prominentes Beispiel für ein solches Problem ist das „Traveling Salesman“ Problem (TSP), bei dem ein Handlungsreisender eine optimale Route finden muss, um eine bestimmte Anzahl von Städten zu besuchen und dabei die kürzeste Gesamtstrecke zurückzulegen [10].

Das Hauptproblem beim TSP ist, dass die Anzahl der möglichen Routen exponentiell mit der Anzahl der Städte zunimmt [10]. Dies bedeutet, dass die Suche nach der optimalen Lösung auf herkömmliche Weise sehr zeitaufwendig sein kann. In der Praxis könnte dies bedeuten, dass selbst bei einer vergleichsweise geringen Anzahl von Städten die Anzahl der möglichen Routen so groß ist, dass es lange dauern kann, um alle möglichen Optionen durchzugehen, um die beste zu finden [10].

2.1.4 Lokale Optima

Eine zusätzliche Herausforderung in der Welt der Optimierungsprobleme sind multimodale Funktionen. Der Begriff „multimodal“ bezieht sich auf Funktionen, die mehrere lokale Optima aufweisen [8]. Das heißt, es existieren Lösungen, die besser sind als ihre benachbarten Lösungen [8]. Das globale Optimum ist dabei das lokale Optimum, das im Vergleich zu allen anderen lokalen Optima am besten ist [8]. Im Gegensatz dazu stehen unimodale Probleme, die nur ein einziges lokales Optimum besitzen, welches gleichzeitig auch das globale Optimum darstellt [8].

Ein beispielhafter Algorithmus, der mit den Herausforderungen von lokalen Optima konfrontiert ist, ist der Hill Climbing Algorithmus [10]. Dieser Algorithmus startet typischerweise bei einer zufälligen Lösung und versucht von dort aus, eine bessere zu finden, indem er schrittweise in Richtung eines zufälligen Nachbarzustands geht [10]. Wenn dieser Nachbarzustand besser ist als der aktuelle Zustand, wird die aktuelle Lösung durch die neue ersetzt [10]. Andernfalls wird weiterhin in der Umgebung der aktuellen Lösung nach besseren Lösungen gesucht, wobei auch zufällige Schritte unternommen werden [10].

Durch die Einfachheit neigt dieser Algorithmus dazu, in lokale Optima zu konvergieren und anschließend darin stecken zu bleiben, da nur nach benachbarten Lösungen gesucht wird und diese auch sofort angenommen werden [10]. Dies kann zum Problem werden, wenn nach dem globalen Optimum gesucht wird [10].

2.2 Evolutionäre Algorithmen

Da das Konzept von Optimierungsproblemen behandelt wurde, kann nun ein tieferer Einblick in evolutionäre Algorithmen gegeben werden.

2.2.1 Biologische Evolution

Bei den Evolutionären Algorithmen handelt es sich um eine spezielle Klasse von Optimierungsverfahren, die von der biologischen Evolution inspiriert sind [10]. Sie nutzen die Mechanismen der natürlichen Evolution wie Selektion, Mutation und Rekombination um komplexe Probleme zu lösen [10].

Die Idee

Die grundlegende Idee der biologischen Evolution liegt in der Fähigkeit der Natur, sich durch Anpassung an eine sich verändernde Umgebung anzupassen [10]. Mithilfe von Innovationen, die durch die biologische Evolution ermöglicht werden, können Lebewesen ihre Umwelt effizienter nutzen und gleichzeitig die Fähigkeit zum Überleben an ihre Nachkommen weitervererben [10].

Der Zyklus

Die biologische Evolution kann als ein Zyklus dargestellt werden. Es existiert zu Beginn eine Population, welche mittels Reproduktion neue Individuen erzeugt [8]. Dabei wird das Erbgut der Eltern an die Kinder weitergegeben [8]. Zusätzlich werden durch Umwelteinflüsse wie UV-Strahlung Mutationen in den Kindern generiert [10]. Die natürliche Selektion entscheidet anschließend, welche Individuen überleben oder aussterben [8]. Damit soll sichergestellt werden, dass nur stärkere Individuen in der Population überleben (Survival of the fittest) [8]. Durch diesen Zyklus wird die zugrunde liegende Population über Generationen hinweg verbessert [8].

2.2.2 Allgemeine Funktionsweise

Evolutionäre Algorithmen nutzen diesen Zyklus zur Lösung von Optimierungsproblemen [10], siehe Abbildung 2.1.

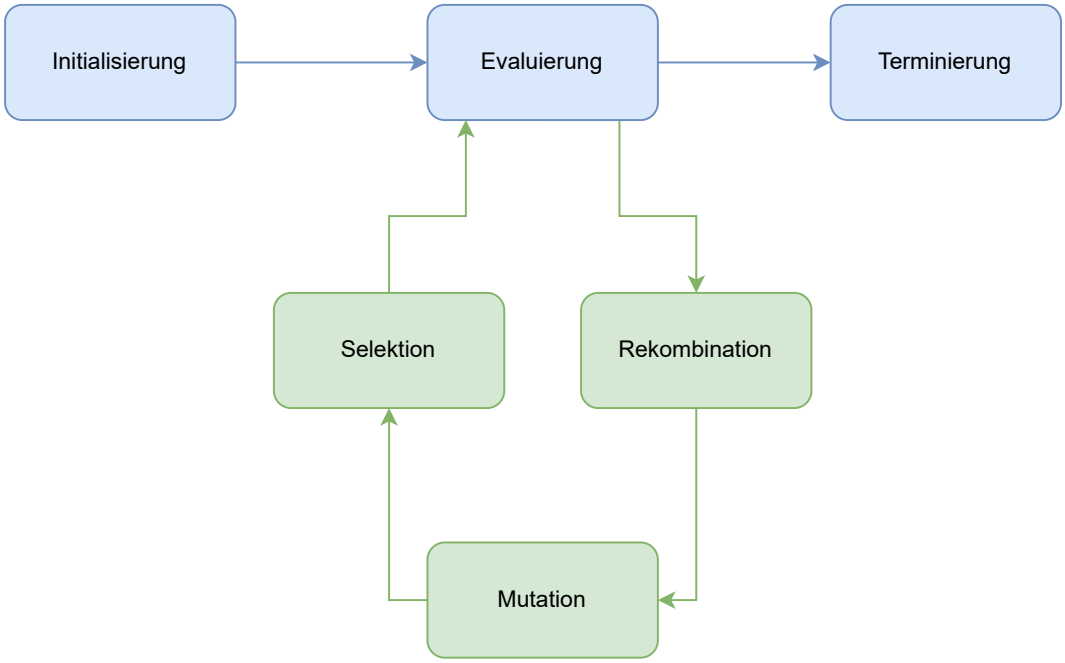


Abbildung 2.1: Abstrakter Durchlauf von evolutionären Algorithmen

Repräsentation

Bevor ein Problem gelöst werden kann, muss zuerst eine Verbindung zwischen der realen Welt und der Welt der EA hergestellt werden [8]. Dafür muss ein Mapping zwischen dem ursprünglichen Problem und dem Lösungsraum aufgebaut werden, in dem die Evolution der Lösungen stattfindet [8]. Die Objekte, die mögliche Lösungen im ursprünglichen Problemkontext repräsentieren, werden als Phänotypen bezeichnet, während ihre codierte Darstellung innerhalb der EA als Genotypen bezeichnet wird [8]. Ziel ist es, die Phänotypen den Genotypen zuzuordnen [8].

Beispielsweise könnten bei einem Optimierungsproblem mit ganzen Zahlen die ganzen Zahlen als Phänotypen genommen werden [8]. Die ganzen Zahlen könnten dann im Kontext der EA als Binärcode repräsentiert werden (Genotyp) [8].

Eine Lösung, also ein guter Phänotyp, wird erreicht, indem nach dem Genotypen gesucht wird, wodurch der optimale Phänotyp dargestellt werden kann [8]. Unter dem Phänotyp-Raum bezeichnet man Lösungen als Lösungskandidaten, Phänotypen oder Individuen [8]. Unter dem Genotyp-Raum werden Bezeichnungen wie Genotyp, Chromosom und Individuum verwendet [8].

Fitnessfunktion

Die Fitnessfunktion oder auch Evaluierungsfunktion genannt, wird verwendet, um die Genotypen zu bewerten [8]. Dafür muss der Genotyp in den entsprechenden Phänotyp entschlüsselt werden, damit die Funktion auf den entsprechenden Phänotyp angewendet werden kann [8].

Als Beispiel können wieder die ganzen Zahlen als Phänotyp und die binäre Codierung als Genotyp verwendet werden [8]. Das Optimierungsproblem besteht nun darin, die Fitnessfunktion x^2 zu minimieren, also den Genotyp zu finden, durch den der Phenotyp den kleinsten Wert ergibt, wenn dieser in der Funktion angewendet wird [8]. Ein Beispiel-Individuum im Genotypraum könnte eine Binärzahl sein, z. B. 0101 [8]. Um die Fitness zu berechnen, wird diese als Ganzzahl dekodiert: $0101 -> 5$ [8]. Danach wird sie in der Fitnessfunktion angewendet, um die Fitness des Individuums zu berechnen: $5^2 = 25$ [8]. Somit hat das Individuum eine Fitness von 25.

Population

Die Population ist eine Sammlung von Genotypen, die mögliche Lösungskandidaten repräsentieren [8]. Bevor eine Population initialisiert wird, wird eine Größe festgelegt, die konstant durch den gesamten Verlauf des Algorithmus bleibt [8]. Dadurch wird die Anzahl an Lösungskandidaten begrenzt und ein Wettbewerb in der Population erzeugt [8].

Initialisierung

Evolutionäre Algorithmen beginnen zunächst damit, Individuen mit zufälligen Werten zu generieren [8]. Diese Individuen bilden zusammen eine Population und sind gleichzeitig potenzielle Lösungskandidaten in einem Suchraum [8].

Evaluierung

Bei der Evaluierung erhält jedes Individuum der Population eine Fitness [8]. Die Fitness bestimmt die Qualität der Lösung und legt die Überlebenschance des Individuums fest [8]. Diese wird meist durch eine Funktion vergeben, die das Optimierungsproblem definiert [8]. Während der Evaluierung kann der Algorithmus überprüfen, ob ein Abbruchkriterium erreicht wurde, um den Algorithmus zu beenden [8]. Das Kriterium kann entweder die optimale Fitness oder eine gewisse Präzision $\epsilon > 0$ sein [8]. Aufgrund ihrer stochastischen Natur könnten evolutionäre Algorithmen in bestimmten Situationen Schwierigkeiten haben, eine Lösung gemäß den festgelegten Kriterien zu finden [8]. Dies könnte potenziell dazu führen, dass der Algorithmus nicht erfolgreich terminiert [8]. Damit sichergestellt ist, dass der Algorithmus terminiert, können weitere Terminierungskriterien eingeführt werden, wie z. B. eine maximale Anzahl an Generationen (Durchläufen) [8].

Rekombination

Falls das Terminierungskriterium nicht erreicht wird, wird zur Rekombination übergegangen. Die Rekombination oder auch Kreuzung genannt, dient dazu, die ausgewählten Individuen miteinander zu kombinieren, um neue Individuen zu erzeugen [8]. Hier werden die Merkmale des Genotyps der Individuen zufällig ausgewählt und miteinander kombiniert, um Nachkommen zu generieren [8].

Die Auswahl der Individuen kann probabilistisch erfolgen, indem Individuen mit einer besseren Fitness eine höhere Chance gegeben werden, ausgewählt zu werden, als Individuen mit einer schlechteren Fitness [8]. Individuen mit einer schlechten Fitness werden jedoch nicht ausgeschlossen [8]. Der Vorteil hiervon ist, dass der Algorithmus vermeiden kann, in lokale Optima gefangen zu sein [8].

Mutation

Bei der Mutation werden die Merkmale der Genotypen zufällig modifiziert, um weitere Variationen zu erzeugen [8]. Häufig wird dies mithilfe der Mutationsrate bestimmt, die festlegt, wie oft eine Mutation in einem Individuum auftritt [8].

Selektion

Nachdem die Nachkommen erzeugt wurden, erfolgt eine Evaluierung dieser Individuen [8]. Anschließend wird mittels Selektion bestimmt, welche Mitglieder in der Population verbleiben und welche entfernt werden müssen, da die Populationsgröße immer konstant bleibt [8]. Hierfür kann ein deterministischer Ansatz verwendet werden, bei dem die Fitness der Individuen miteinander verglichen wird [8]. Individuen, die eine bessere Fitness haben, werden in die nächste Generation übernommen, während Individuen mit einer schlechten Fitness verworfen werden [8].

Der Algorithmus geht von dieser Stelle aus zurück zur Evaluierung, siehe Abbildung 2.1, wo erneut überprüft wird, ob das Terminierungskriterium erreicht wurde [8].

2.2.3 Warum evolutionäre Algorithmen verwenden?

Die Schwierigkeiten, wie Lokale Optima und großer Suchraum, die im Abschnitt 2.1 erklärt wurden, können zum Teil von evolutionären Algorithmen angegangen werden. In Abschnitt 2.2.2 wurde erklärt, dass durch die probabilistische Selektion vermieden wird, dass der Algorithmus in lokalen Optima stecken bleibt, da auch schlechteren Lösungen eine Chance gegeben wird. Der Vorteil, schlechten Lösungen eine Chance zu geben, ist, dass dadurch möglicherweise neue Lösungen gefunden werden können, die zu Beginn nicht ersichtlich waren [8]. Zudem besitzen evolutionäre Algorithmen eine Population von Lösungen und sind somit nicht auf eine Lösung zurzeit begrenzt, wodurch sie mit großen Suchräumen und lokalen Optima in gewisser Weise umgehen können [8].

Zusammenfassend bieten evolutionäre Algorithmen eine leistungsfähige Methode, um komplexe Optimierungsprobleme anzugehen, insbesondere wenn herkömmliche Verfahren aufgrund der Größe des Suchraums oder der Existenz von mehreren lokalen Optima an ihre Grenzen stoßen würden [10].

2.2.4 Parameter Optimierung

Die Parameter von evolutionären Algorithmen, wie Populationsgröße, Crossoverrate und Mutationsrate, müssen jedoch angepasst werden [10]. Nur den Basisalgorithmus einzusetzen, ohne die Parameter zu optimieren, kann zwar Ergebnisse liefern, aber meistens sind die Ergebnisse nicht von guter Qualität [10].

2.2.5 No Free Lunch Theorem

Das "No Free Lunch Theorem"(Theorem vom kostenlosen Mittagessen) im Bereich der Optimierung besagt, dass es keine allumfassende Optimierungsmethode gibt, die in jeder Situation die beste Leistung erbringt [8]. Mit anderen Worten bedeutet dies, dass die Effektivität eines Optimierungsalgorithmus stark von den spezifischen Eigenschaften des zu lösenden Problems abhängt [8]. Wenn ein Algorithmus für ein bestimmtes Optimierungsproblem gut geeignet ist, besteht die Wahrscheinlichkeit, dass er für ein anderes Problem weniger effizient ist [8].

2.3 Differential Evolution Algorithm

2.3.1 Aufbau und Prinzipien

Der Differential Evolution Algorithmus ist ein effizienter und einfach zu implementierender evolutionärer Algorithmus, um das globale Optimum in einem Optimierungsproblem zu finden [9]. Der Unterschied zu anderen evolutionären Algorithmen besteht darin, dass dieser Algorithmus nicht nur von der Evolution inspiriert wurde, sondern auch von mathematischen (geometrischen) Prinzipien [9]. Die Individuen in der Population werden durch Vektoren im Suchraum repräsentiert [9]. Diese Vektoren bestehen aus Kombinationen von Variablen aus der zu optimierenden Funktion [9]. Der Algorithmus verwendet die Differenz zwischen Individuen, um neue Individuen zu generieren [9].

Der Differential Evolution Algorithmus benötigt verschiedene Parameter, um erfolgreich funktionieren zu können [9]. Diese Parameter sind auf Tabelle 2.1 aufgeführt.

Parameter	Erklärung
F	Mutationsfaktor, um die Stärke der Mutation zu beeinflussen.
D	Beschreibt die Anzahl an Dimensionen für ein Optimierungsproblem.
Cr	Legt die Wahrscheinlichkeit fest, wie häufig das Individuum mutieren soll.
NP	Legt die Populationsgroße fest.
GEN	Legt die maximale Anzahl an Generationen fest.
L	Legt die Untergrenze des Suchraumes fest.
H	Legt die Obergrenze des Suchraumes fest.

Tabelle 2.1: Parameter für den Differential Evolution Algorithm

Angenommen es existiert ein Optimierungsproblem $f(x)$, das minimiert werden muss, gesucht ist dann ein X^*, welches die optimalste Lösung darstellt [9]. Diese Lösung kann als Vektor von Parametern dargestellt werden, wobei $i = \{1, ..., D\}$ gilt [9]. Diese optimale Lösung liegt in einem festgelegten Bereich $L \leq X \leq H$. Die Aufgabe besteht nun darin, diese optimale Lösung zu finden [9].

Initialisierung

Um die optimale Lösung zu finden, wird zuerst eine zufällige Population initialisiert [9]. Die Werte der Variablen jedes Individuums werden normalerweise innerhalb eines vordefinierten Wertebereichs oder einer Domäne zufällig ausgewählt [9]. Dieser Bereich wird durch die Variablen L für die Untergrenze und H für die Obergrenze festgelegt [9]. Neben der Grenze ist es notwendig zu wissen, wie viele Individuen überhaupt erzeugt werden sollten [9]. Dies wird durch die Populationsgröße NP festgelegt [9]. Ein Individuum wird durch folgende Notation beschrieben: $X \in \mathbb{R}^D$ [9]. Angenommen es handelt sich um eine zweidimensionale Funktion, dann wäre $D = 2$, somit wäre $X \in \mathbb{R}^2$ und $X = [X_1, X_2]$ [9]. Die Population ist somit $Pop \in \mathbb{R}^{[D \times NP]}$ [9]. Um die Population generieren zu können, werden drei Laufvariablen $i, j, g \in \mathbb{N}$ benötigt, wobei $j = \{1, ..., NP\}$, $i = \{1, ..., D\}$ und $g = \{1, ..., GEN\}$ gelten [9]. Die Individuen der Population werden mittels Gleichung 2.1 generiert [9].

$$Pop_{i,j} = L + (H - L) \cdot rand_{i,j}[0, 1) \tag{2.1}$$

$rand[0, 1)$ steht für einen Zufallszahlengenerator, der Zufallszahlen zwischen 0 und 1, inklusive 0, generiert [9].

Evaluierung

Die Fitness $f \in \mathbb{R}$ wird dann für jedes Individuum berechnet durch die zu optimierende Funktion $f(x)$ [9], siehe Gleichung 2.2. Das beste Individuum in der Population, wird mittels des Index $iBest \in \mathbb{N}$ gespeichert [9].

$$Fit_j = f(Pop_j) \tag{2.2}$$

Differential Evolution

Nun durchläuft der Algorithmus mithilfe des Index j durch alle Individuen der Population und führt für jedes Individuum die folgenden Operationen durch: Mutation, Rekombination, Validierung und Selektion [9].

Mutation

Zuerst müssen drei Individuen zufällig ausgewählt werden, mittels drei Indizes $r \in \mathbb{N}^3$ [9]. Die Indizes dürfen nicht auf dieselben Individuen zeigen und auch nicht auf das aktuelle Individuum, auf das j zeigt $r1 \neq r2 \neq r3 \neq j$ [9].

Diese drei Individuen werden verwendet, um das Trial-Individuum zu erzeugen, welches später mit dem Individuum auf das j zeigt, verglichen wird [9]. Dafür wird die Gleichung 2.3, verwendet [9]. In der Gleichung wird die Differenz zwischen $(x_{i,r1} - x_{i,r2})$ genommen und mit dem Mutationsfaktor F multipliziert $F \cdot (x_{i,r1} - x_{i,r2})$, dass Produkt wird dann anschließend mit dem dritten Individuum $X_{i,r3}$ addiert [9].

$$X_{i,r3} + F \cdot (x_{i,r1} - x_{i,r2}) \tag{2.3}$$

Rekombination

Bei der Rekombination muss nun mittels einer Bedingung 2.4 entschieden werden, ob der Trial Vektor x_i den Parameter des Mutantenvektors $X_{i,r3} + F \cdot (x_{i,r1} - x_{i,r2})$ erhält oder des aktuellen Vektors $x_{i,j}$ [9].

$$(rand_{i,j}[0,1) < Cr) \vee (Rnd = i) \tag{2.4}$$

Dafür wird die Crossoverrate Cr verwendet. Es wird der Parameter des Mutantenvektors genutzt, wenn die Bedingung erfüllt ist [9]. Ansonsten wird der Wert des Vektors $x_{i,j}$ verwendet [9]. Die Bedingung $(Rnd = i)$ dient als Zusatz, um sicherzustellen, dass mindestens ein Parameter des Vektors verändert wird [9]. Rnd generiert dabei eine Zahl im Bereich [1,...,D] [9].

Validierung

Nachdem der Trial-Vektor erstellt wurde, müssen die Parameter des Vektors nun daraufhin überprüft werden, ob sie sich im gültigen Bereich befinden [9]. Dafür wird durch alle Parameter des Trial-Vektors iteriert und überprüft, ob die folgende Bedingung 2.5 erfüllt wird [9]. Falls ein Parameter nicht im gültigen Bereich liegt, wird dieser durch einen neuen zufälligen Parameter ersetzt, der im Bereich liegt [9], Siehe Formel 2.6.

$$(x_i \notin [L, H]) \tag{2.5}$$

$$x_i = L + (H - L) \cdot rand_i[0, 1)] \tag{2.6}$$

Selektion

Nach der Validierung wird der Trial-Vektor mit dem aktuellen Individuum verglichen [9]. Falls die Fitness des Trial-Vektors kleiner ist als die des aktuellen Individuums, wird das aktuelle Individuum durch den Trial-Vektor ersetzt, da er eine bessere Lösung darstellt [9]. Anschließend wird überprüft, ob der Trial-Vektor, falls er besser als das aktuelle Individuum ist, besser als das beste Individuum der Population ist, das beim Evaluieren durch den Index $iBest$ festgelegt wurde [9]. Ist dies der Fall, wird der Index $iBest$ entsprechend aktualisiert [9].

Aktualisierung

Der Algorithmus aktualisiert die Population iterativ über mehrere Generationen hinweg [9]. Wenn die Laufvariable $g = GEN$ erreicht, terminiert der Algorithmus, da die maximale Anzahl an Generationen erreicht wurde [9].

Zusammenfassung

Wenn nun alle oben genannten Schritte zusammengefügt werden, ergibt sich der folgende Pseudocode [9].

```
Require:    D -- problem dimension (optional)
            NP, F, Cr -- control parameters
            GEN -- stopping condition
            L, H -- boundary constraints
Initialize population Pop_ij <- rand_ij[L, H] and Evaluate fitness Fit_j <-
    f(Pop_j)
for g = 1 to GEN do
    for j = 1 to NP do
        Choose randomly r1, r2, r3 ∈ [1, ..., NP], r1 ≠ r2 ≠ r3 ≠ j
        Create trial individual X <- −S(r, F, Cr, Pop)
        Verify boundary constraints if (x_i not in [L, H]) x_i <-
    randi[L, H]
        Select better solution (X or Pop_j), and update iBest if
    required
    end for
end for
```

Quellcode 2.1: Pseudocode des Differential Evolution Algorithmus [9]

2.4 Random Number Generators (RNGs)

Zufällige Zahlen sind in Bereichen wie Kryptografie, Glücksspiele und Computersimulationen von großer Bedeutung [12]. Um Zufallszahlen zu generieren, werden RNGs eingesetzt [12]. Es gibt zwei Arten von RNGs: True Random Number Generators (TRNGs) und Pseudo Random Number Generators (PRNGs) [12].

2.4.1 True Random Number Generators (TRNGs)

TRNGs verwenden unvorhersehbare physikalische Prozesse, um echte zufällige Zahlen zu generieren [12]. Falls die physikalischen Prozesse nicht frei von Verzerrungen sind, werden Ausgleichsmechanismen verwendet, um diese zu entfernen [12]. Ein Beispiel für einen TRNG wäre der Münzwurf [12]. Wenn die Münze perfekt symmetrisch ist, besteht eine Chance von jeweils 50% für Kopf oder Zahl [12]. Mithilfe des Münzwurfs kann eine 32-Bit-Zahl generiert werden, wobei Kopf für 0 und Zahl für 1 steht.[12]

Der Vorteil von TRNGs ist, dass sie echte zufällige Zahlen generieren und die Zahlen keine Korrelation zueinander haben [12]. Jedoch sind TRNGs langsam und daher nur für bestimmte Anwendungen geeignet [12]. TRNGs werden bei kryptografischen Anwendungen eingesetzt, da diese auf echte Zufallszahlen angewiesen sind [12].

2.4.2 Pseudo Random Number Generators (PRNGs)

PRNGs hingegen verwenden deterministische Algorithmen, um eine Sequenz von zufälligen Zahlen zu generieren [12]. Die Grundidee besteht darin, dass eine Funktion f einen sogenannten Seed, also eine Zahl x, erhält, um eine Folge von Zahlen zu generieren, die so zufällig wie möglich ist [12].

Der Vorteil von PRNGs ist, dass sie schnell und effizient sind [12]. Im Gegensatz zu TRNGs benötigen PRNGs keine Ausgleichsmechanismen oder spezialisierte Hardware und sind deshalb sehr portabel [12]. Durch den Seed können PRNGs Sequenzen von Zahlen reproduzieren, was das Debuggen ermöglicht [12]. Auf der anderen Seite tendieren PRNGs dazu, Zahlen zu generieren, die miteinander korreliert sind [12]. Durch Korrelationen kann das Vorhersagen von Zahlen erleichtert werden, was in kryptografischen Anwendungen ein Problem darstellen kann [12]. Zudem haben PRNGs nur eine endliche Sequenzlänge (Periodenlänge), was bedeutet, dass die Zahlen in der Sequenz nach einem bestimmten Punkt wiederholt werden [12].

2.4.3 Qualität von PRNGs

Die Fähigkeit, unvorhersehbare, unabhängige und gleichmäßige Zahlen zu generieren, beschreibt die Qualität eines RNG [6]. Je höher die Qualität eines RNG ist, desto unvorhersehbarer sind die generierten Zahlen [6]. Diese Eigenschaft ist zum Beispiel bei kryptografischen Anwendungen von Bedeutung, da die Kryptografie auf unvorhersehbare Zahlen angewiesen ist, um sichere Schlüssel zu generieren [6]. Um die Qualität messen zu können, werden eine Menge von Tests verwendet, die Testsuite genannt werden [6]. Eines der weit verbreitetsten Testsuiten sind die NIST- & BSI-Testsuiten, da sie als zuverlässig und effektiv gelten [6].

2.4.4 Bedeutung von RNGs in evolutionären Algorithmen

Wie im Abschnitt 2.2.2 erklärt wurde, nutzen EA stochastische Methoden wie Selektion, Mutation und Rekombination, die auf zufälligen Zahlen basieren. RNGs ermöglichen die Erzeugung dieser Zahlen, um die Umsetzung dieser stochastischen Methoden zu realisieren.

2.5 Verwandte Arbeiten

Im Rahmen dieser wissenschaftlichen Untersuchung ist es unerlässlich, den Wissensstand auf dem Gebiet der vorliegenden Forschung ausführlich zu untersuchen. Durch die Analyse kann ein umfassender Überblick über bereits durchgeführte Studien und Erkenntnisse gegeben werden, die im Zusammenhang mit dem Thema dieser Arbeit stehen. Durch die Analyse dieser verwandten Arbeiten können wertvolle Einblicke gewonnen werden, die nicht nur das Verständnis vertiefen, sondern auch als Grundlage dienen, um eine verbesserte und differenziertere Analyse zu gestalten.

On Random Numbers and the Performance of Genetic Algorithms
Die Arbeit von Erick Cantú-Paz [4] untersucht den Einfluss von Zufallszahlengeneratoren (RNGs) und verschiedenen Methoden zur Erzeugung von Zufallszahlen auf genetische Algorithmen. Dabei wurde der Mersenne Twister (PRNG) in seiner originalen Implementierung sowie in einer Implementierung mit einer auf 1000 begrenzten Periodenlänge verwendet. Zusätzlich wurde ein echter Zufallszahlengenerator (TRNG) verwendet. Diese RNGs wurden auf verschiedenen Testfunktionen/Benchmark-Funktionen angewendet. Die Untersuchung umfasste insgesamt 100 Durchläufe mit unterschiedlichen Seeds. [4]

Die Ergebnisse der Studie zeigen, dass die Wahl der Methode zur Erzeugung von Zufallszahlen einen signifikanten Einfluss auf die Leistung des genetischen Algorithmus hat. In einigen Fällen erzielten sogar weniger qualitativ hochwertige PRNGs bessere Ergebnisse als ihre qualitativ besseren Gegenstücke. Diese Ergebnisse waren jedoch nicht konsistent über alle Testfunktionen hinweg. Darüber hinaus wurde festgestellt, dass die Initialisierung der Population mithilfe des RNGs entscheidend für die Leistung des genetischen Algorithmus ist. [4]

Sensitiveness of Evolutionary Algorithms to the Random Number Generator
In der Arbeit von Miguel Cárdenas-Montes, Miguel A. Vega-Rodríguez und Antonio

Gómez-Iglesias [5] wurde untersucht, wie sich verschiedene RNGs auf die Leistung von evolutionären Algorithmen auswirken. Die Autoren verwenden verschiedene Algorithmen, darunter den genetischen Algorithmus, den Partikelschwarm-Optimierungs-Algorithmus und den Differential Evolution Algorithmus, um verschiedene Testprobleme zu lösen. Sie verwenden verschiedene RNGs, um zu untersuchen, ob die Wahl des Generators die Qualität der Lösungen beeinflusst. Dabei wurde jede Testfunktion 10^4 mal von jedem RNG durchlaufen. Die Ergebnisse der Durchläufe wurden mittels eines statistischen Tests evaluiert, um aussagekräftige Schlussfolgerungen treffen zu können. Die Ergebnisse haben gezeigt, dass die Wahl des RNG einen Einfluss auf die Leistung von evolutionären Algorithmen hat, insbesondere auf den Differential Evolution Algorithmus. [5]

Comparison of Pseudorandom Numbers Generators and Chaotic Numbers Generators used in Differential Evolution

Die Studie von Lenka Skanderova and Adam Ŕehoŕ [26] untersucht die Leistung verschiedener PRNGs und chaotischer Zufallszahlengeneratoren im Kontext der Differential Evolution. Das Ziel besteht darin, die Konvergenzgeschwindigkeit des Algorithmus zum globalen Minimum unter Verwendung unterschiedlicher PRNGs und chaotischer Zufallszahlengeneratoren zu vergleichen. [26]

Die Autoren analysieren PRNGs wie Mersenne Twister, Crypto Random, Microsoft .NET System.Random class, Visual Studio 2010, Multiply-with-Carry und Xorshift. Zusätzlich werden chaotische Zufallszahlengeneratoren wie die Logistische Map, Arnold Cat Map und Sinai betrachtet. Der Algorithmus wird auf sechs ausgewählten Testfunktionen mittels der PRNGs und chaotischer Zufallszahlengeneratoren angewendet. Die Ergebnisse sind in Tabellen dargestellt, die die Mindest-, Höchst- und Durchschnittswerte für jeden PRNG und chaotischen Zufallszahlengenerator über die ausgewählten Testfunktionen hinweg vergleichen. Die Autoren haben herausgefunden, dass die Logistische Map der effektivste Generator in Bezug auf die Konvergenzgeschwindigkeit der Differential Evolution für die meisten Testfunktionen ist. [26]

Population initialization techniques for evolutionary algorithms for single-objective constrained optimization problems: Deterministic vs. stochastic techniques

Die Studie von Alaa Tharawat und Wolfram Schenck [28] untersucht, ob sich bei der Anwendung verschiedener Methoden zur Initialisierung der Population Unterschiede in evolutionären Algorithmen zeigen. Dabei werden sowohl deterministische als auch stochastische Methoden wie RNGs, Latin Hypercube, Sobol, Halton und Kronecker verwen-

det. Diese Methoden werden durch verschiedene evolutionäre Algorithmen auf Benchmarkfunktionen angewendet. [28]

Die Untersuchung ergibt, dass Populationen, die mithilfe deterministischer Methoden wie Sobol und Halton erstellt wurden, besser verteilt sind als solche, die stochastische Methoden verwenden. Dieses Ergebnis steht im Einklang mit der Erkenntnis, dass die Verwendung von niedrig diskrepanter Sequenzen wie Halton und Sobol zu einer gleichmäßigeren Verteilung der generierten Punkte im Raum führt im Vergleich zu RNGs, die häufig als Standardmethode zur Initialisierung von Populationen in evolutionären Algorithmen genutzt werden. Darüber hinaus zeigen die Ergebnisse, dass evolutionäre Algorithmen bei ausreichender Anzahl von Iterationen nicht empfindlich auf die Initialisierungsmethoden reagieren und es keine signifikanten Unterschiede zwischen den genannten Methoden zur Initialisierung der Population gibt. Außerdem zeigen die Methoden mit geringer Diskrepanz, dass sie die Erkundungsfähigkeit von evolutionären Algorithmen in den frühen Iterationen verbessern. [28]

Do Evolutionary Algorithms Indeed Require Random Numbers? Extended Study

Die Arbeit von Ivan Zelinka, Mohammed Chadli, Donald Davendra, Roman Senkerik, Michal Pluhacek und Jouko Lampinen [29] untersuchte, ob RNGs in einem evolutionären Algorithmus notwendig sind. Es wurden PRNGs mit deterministischen Methoden verglichen. Als Benchmark-Funktionen wurden Funktionen wie die Schwefel-Funktion verwendet. Die Autoren fanden heraus, dass RNGs durch deterministische Methoden ersetzt werden können, ohne die Leistung des Algorithmus zu verringern. [29]

Weitere Arbeiten

- Is Differential Evolution Sensitive to Pseudo Random Number Generator Quality? – An Investigation [22]
- Study on the Effects of Pseudorandom Generation Quality on the Performance of Differential Evolution [30]

Erkenntnisse

Basierend auf den einzelnen Durchführungen der Tests in den verwandten Arbeiten lässt sich für den eigenen Durchlauf die Schlussfolgerung ziehen, dass es notwendig ist, mehrere Durchläufe mit unterschiedlichen Seeds durchzuführen, um statistisch relevante Ergebnisse zu erhalten. Diese statistisch relevanten Ergebnisse sollten nicht nur mit dem Auge untersucht werden, sondern auch mittels statistischen Test evaluiert werden. Darüber

hinaus verdeutlicht die Studie von Miguel Cárdenas-Monte et al. [5], dass der Differential Evolution Algorithmus in der Theorie (unter Verwendung von Benchmark-Funktionen) empfindlich auf die Auswahl des Zufallszahlengenerators reagiert. Daher sollte untersucht werden, ob sich dieses Verhalten auch in der Praxis zeigt.

3 Methodik

3.1 Aufbau

In diesem Kapitel wird erklärt, wie der grundsätzliche Aufbau des Versuchs sein wird. Zuerst wird das Optimierungsproblem vorgestellt und anschließend wird der Differential Evolution Algorithmus für das Optimierungsproblem erläutert.

3.1.1 Optimierungsproblem

Die bisher im Kapitel 2.5 vorgestellten verwandten Arbeiten haben ihre Untersuchungen hauptsächlich mithilfe von Benchmark-Funktionen durchgeführt. Das Ziel dieser Arbeit ist es, zu untersuchen, ob in der Praxis ein signifikanter Unterschied in der Effizienz des Differential Evolution Algorithmus existiert, wenn unterschiedliche RNGs verwendet werden. Für diese Analyse wird daher ein praktisches Optimierungsproblem verwendet. Hierbei wird der Datensatz und das Vorgehen aus dem Paper der Autoren Christian Lins, Daniel Eckhoff, Andreas Klausen, Sandra Hellmers, Andreas Hein und Sebastian Fudickar mit dem Titel „An evolutionary approach to continuously estimate CPR quality parameters from a wrist-worn inertial sensor“ [13] verwendet. Das Vorgehen des Papers wird übernommen und mit verschiedenen RNGs durchgeführt, um analysieren zu können, ob ein signifikanter Unterschied in den Ergebnissen existiert.

Inhalt des Papers

Das Ziel des Papers war es, zu untersuchen, ob es möglich ist, mittels eines Trägheitssensors, der am Handgelenk fixiert war, die Kompressionsfrequenz CCF und Kompressionstiefe CCD des Probanden während einer Herz-Lungen-Wiederbelebung abzuschätzen, um die Effektivität der Herz-Lungen-Wiederbelebung bewerten zu können. Die Motivation hinter diesem Vorhaben liegt darin, die Effektivität der Herz-Lungen-Wiederbelebung zu

verbessern, da diese einen entscheidenden Beitrag zur Rettung von Leben bei plötzlichem Herzstillstand leistet. [13]

Datensatz

Der Datensatz enthält die Beschleunigungswerte eines dreidimensionalen Inertialsensors [13]. Der Sensor zeichnet die Beschleunigungswerte des Handgelenks während einer Herz-Lungen-Wiederbelebung auf [13]. Der Beschleunigungsmesser des Sensors erfasst die Beschleunigungskräfte, die auf den Sensor wirken, und enthält auch die Konstante der Erdanziehungskraft, die auf die drei Achsen des Sensors verteilt ist [13]. Diese Verteilung ist abhängig von der Ausrichtung des Sensors [13]. Da der Sensor am Handgelenk getragen wird, ist die Position des Sensors nicht orthogonal zur Brust, so dass die Verteilung der Erdanziehungskraft nicht gleichmäßig auf alle drei Achsen verteilt ist [13].

Für dieses Optimierungsproblem sind nur die Beschleunigungswerte relevant, die durch den Druck auf den Brustkorb ausgeübt werden, weshalb die Erdanziehungskraft abgezogen werden muss [13]. Zum Abziehen der Erdanziehungskraft wird der euklidische Abstand a über alle drei Achsen berechnet und daraus die Konstante der Erdanziehungskraft ($9.81\frac{m}{s^2}$) abgezogen [13], siehe Gleichung 3.1.

$$a_i = \sqrt{(a_{i,x}^2) + (a_{i,y}^2) + (a_{i,z}^2)} - 9.81\frac{m}{s^2}, i \in \{1, ..., S_{len}\} \tag{3.1}$$

Somit ergibt sich eine eindimensionale Datenreihe, die mittels eines Fensters in Abschnitte unterteilt werden kann [13]. Dabei legt S_{len} die Länge dieses Fensters fest [13].

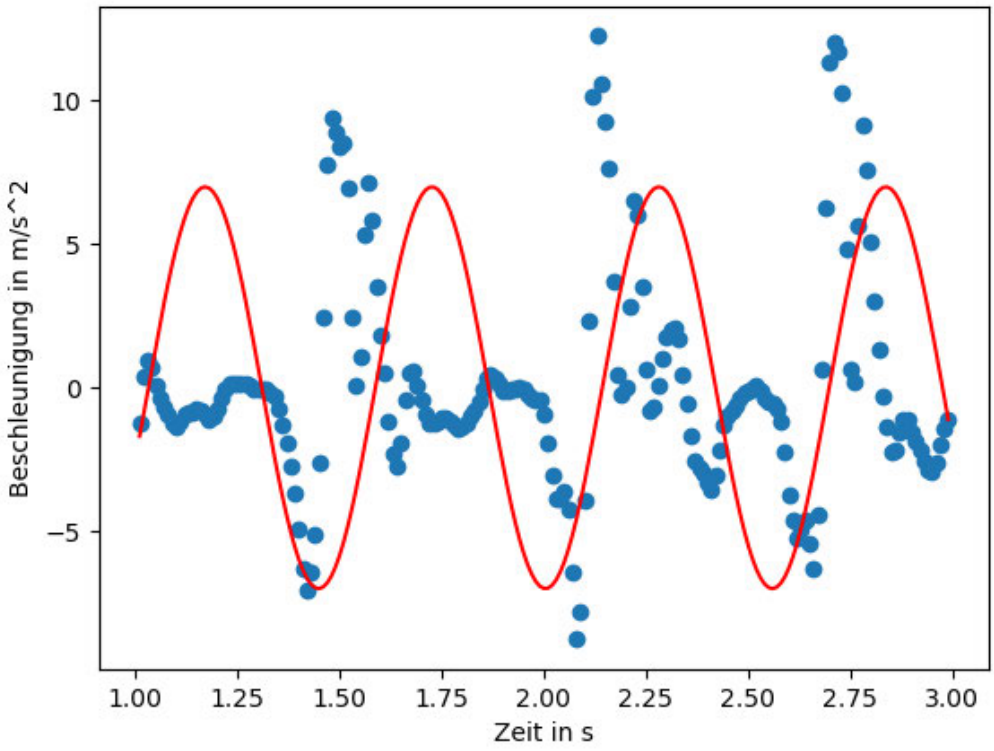


Abbildung 3.1: Plot eines Fenster mit einer Sinusfunktion in Rot

Beim Plotten eines solchen Fensters, siehe Abbildung 3.1, zeigt sich ein periodischer Verlauf [13]. Da eine Sinusfunktion einen ähnlichen periodischen Verlauf hat, kann dann versucht werden, diese an den Daten in diesem Fenster anzupassen, um die Kompressionsfrequenz CCF und Kompressionstiefe CCD abzuschätzen [13]. Die Sinusfunktion ist folgendermaßen aufgebaut:

$$f(t) = A \cdot sin(\omega t + \varphi) + D \tag{3.2}$$

Im Detail besteht eine Sinusfunktion aus vier Parametern: A, ω, φ und D [13]. A steht für die Amplitude, ω für die Kreisfrequenz, φ ist die Phasenverschiebung und D für die vertikale Verschiebung [13]. Somit gibt es vier freie Variablen, was ein 4-dimensionales Optimierungsproblem darstellt, $X \in \mathbb{R}^4 \rightarrow X = [X_1, X_2, X_3, X_4]$.

Durch die Anpassung der Sinusfunktion an die Beschleunigungsdaten kann jedoch nicht auf die vertikale Verschiebung des Handgelenks geschlossen werden [13]. Dazu müssen die Beschleunigungsdaten zweimal integriert werden, um die Verschiebung in Metern zu erhalten [13]. Aufgrund von Fehlern und Ungenauigkeiten in den Beschleunigungsdaten wird anstelle der doppelten Integration eine doppelte Ableitung der Sinusfunktion durchgeführt [13], wie in Gleichung 3.3 und 3.4 zu sehen ist. Hierbei wird die Funktion $f(t)$ zweimal nach t abgeleitet.

$$f(t) = \frac{\delta^2}{\delta t^2}(A \cdot sin(\omega t + \varphi) + D) \tag{3.3}$$

$$f(t) = -A\omega^2 \cdot sin(\omega t + \varphi) \tag{3.4}$$

Der Parameter D fällt dann aus der Funktion heraus, so dass aus dem 4-dimensionalen Problem ein 3-dimensionales wird $X \in \mathbb{R}^3 \rightarrow X = [X_1, X_2, X_3]$ [13]. Ziel ist es, die Parameter zu finden, mit denen die Sinusfunktion möglichst nah an die Werte im Fenster angepasst werden kann [13].

In einer angepassten Funktion entspricht der Parameter w der Kompressionsfrequenz CCF, während $2 \cdot A$ die Kompressionstiefe CCD darstellt [13]. Durch die Auswertung dieser Parameter kann festgestellt werden, ob die zugrundelegende Herz-Lungen-Wiederbelebung erfolgreich durchgeführt wird oder nicht, wenn die Kompressionsfrequenz im Bereich von 100-120 Kompressionen pro Minute (cpm) und die Kompressionstiefe im Bereich von 5-6 cm liegt [13].

Um herauszufinden, wie nah die Sinusfunktion an den Parametern ist, wird der Root Mean Squared Error (RMSE) verwendet [13], siehe Gleichung 3.5.

$$L = \sqrt{\frac{1}{S_{len}} \sum_{t=0}^{T} (a(t) - f(t))^2} \tag{3.5}$$

Der RMSE ist eine statistische Metrik, um die Qualität eines Modells oder einer Vorhersage zu bewerten [11]. In diesem Fall würde der RMSE bewerten, wie nah die Sinusfunktion an den tatsächlichen Daten liegt [13]. Ziel des Differential Evolution Algorithmus ist es, diesen Wert zu minimieren [13]. Somit kann das Optimierungsproblem folgendermaßen

definiert werden: Gesucht werden die Parameter A*, ω^* und $\varphi^* \in \mathbb{R}$, um eine Sinusfunktion zu bilden, die den kleinsten RMSE L^* liefert.

Dabei ist a der T-dimensionale Tupel der Beschleunigungswerte, entweder als die totale Beschleunigung oder vertikale Beschleunigung. $|a|$ steht für die Dimension, genauer gesagt für die Anzahl an Elementen im Tupel. T setzt sich aus der Fenstergröße S_{len} und der Frequenz, mit der die Daten aufgenommen wurden, zusammen. Im Paper wurden die Daten mit 100 Hz aufgenommen und es wurde ein Fenster von 3 s verwendet, weshalb $T = 300$ ist. [13]

3.1.2 Der Algorithmus

Nachdem nun das Optimierungsproblem festgelegt wurde, wird nun der Aufbau des Differential Evolution Algorithmus für dieses Optimierungsproblem erklärt.

Wie bereits im Kapitel 2.1 erklärt wurde, können evolutionäre Algorithmen lokale Optima überwinden, indem sie eine Vielzahl an Lösungen in einer Population bereitstellen. Da es sich bei diesem Optimierungsproblem um eine Sinusfunktion handelt, existieren keine unterschiedlichen lokalen Optima, sondern nur periodische [13]. Daher werden in jeder Generation G der Population nur λ Individuen mutiert und rekombiniert [13].

Abbruchskriterium

Der Algorithmus läuft über mehrere Generationen G bis G_{max} erreicht wird oder die Konvergenz in der Generation G kleiner als C_{min} ist [13]. Bei jeder Transition von einer Generation zur nächsten werden, wie im Kapitel 2.3 erklärt, die Schritte Selektion, Mutation und Crossover durchgeführt.

Grenzen der Parameter

Für die Initialisierung der Individuen, muss festgelegt werden, welche Grenzen die Parameter in den Vektoren haben. Dies wird auf Tabelle 3.1 welche von dem Paper [13] stammt, dargestellt.

Individuum x	Intervall für die Parameter	CCF/CCD
$x_{i,G}(0)$	$A \in [0.1, 5.0]$	CCD is 2A in cm
$x_{i,G}(1)$	$\omega \in [\pi, 7\pi]$	CCF is $\frac{\omega}{2\pi} \cdot 60$ in cpm
$x_{i,G}(3)$	$\varphi \in [0, 2\pi]$	nur für das anpassen des Modells

Tabelle 3.1: Intervalle der Parameter vom Vektor x

Initialisierung

$X_{i,G}$ ist ein 3-dimensionaler Vektor, dabei sind die Parameter für das Individuum i A, ω und φ in der Generation G, mit $i \in [1, \mu]$ und $G \in [1, G_{max}]$ [13].

Bei der Initialisierung müssen neue Individuen erzeugt werden. Dazu werden μ Individuen zufällig erstellt, indem PRNGs, die später vorgestellt werden, zufällige Werte für die Parameter A, ω, φ erzeugen, die jeweils in den Intervallen der Tabelle 3.1 liegen.

Für jede Generation werden zwei Individuen mittels Mutation und drei durch Rekombination neu generiert, so dass $\lambda = 1 + 1 + 3 = 5$ ist [13]. Das bedeutet, dass je Generation G, $X_{\mu+1}$ bis $X_{\mu+\lambda}$ Individuen neu generiert werden [13].

Mutation

Die Mutation findet in zwei Fällen statt [13]. Zunächst wird ein zufällig neues Individuum $\hat{x}_{\mu+1}$ erstellt, um sicherzustellen, dass der Suchraum effektiv weiter genutzt wird [13]. In jeder Generation wird das beste Individuum x_0 genommen und mutiert, um das zweite Kind $\hat{x}_{\mu+1}$ zu generieren [13]. Auf diese Weise wird eine Feinabstimmung am besten Individuum vorgenommen, um die bestehende Lösung zu optimieren [13]. Das Individuum wird folgendermaßen erstellt, indem eine Konstante $M = 0,999$ verwendet wird, um eine zufällige Zahl zwischen M und $2 - M$ zu generieren [13], siehe Gleichung 3.6.

$$m_k = Rand(M, 2 - M) \tag{3.6}$$

Der Mutationsfaktor m_k wird anschließend mit dem Parameter von $x_0(k)$ für $k \in [0, 1, 2]$ multipliziert, um einen neuen Parameter zu erzeugen [13]. Dabei ist k der Index des 3-dimensionalen Vektors [13].

$$\hat{x}_{\mu+2}(k) = m_k \cdot x_0(k) \tag{3.7}$$

Rekombination

Neben der Mutation wird im Anschluss die Rekombination durchgeführt [13]. Hier werden zufällig zwei Elternindividuen ausgewählt [13], siehe Gleichung 3.8 und 3.9.

$$x_a \text{ mit } a = Rand(1, \mu) \tag{3.8}$$

$$x_b \text{ mit } b = Rand(1, \mu), b \neq a \tag{3.9}$$

Anders als bei Kapitel 2.3 wird nicht die Differenz zwischen den Individuen genommen, sondern es wird das Minimum und Maximum zwischen den beiden Parametern der Eltern-Individuen genommen [13]. Damit kann dann eine zufällige Zahl zwischen diesem Minimum und Maximum zu generiert werden, um den neuen Parameter für das neue Individuum zu generieren [13], siehe Gleichung 3.10.

$$\hat{x}_{\mu+i}(k) = Rand(min(x_a(k), x_b(k)), max(x_a(k), x_b(k))) \text{ mit } k \in [0, 1, 2] \tag{3.10}$$

Selektion

Nun müssen die Individuen bewertet werden, um später entscheiden zu können, welches Individuum in der Generation verbleibt [13]. Dazu wird der RMSE als Kostenfunktion verwendet [13], siehe Gleichung 3.11.

$$\sum_{t=0}^{T} (a(t) - f_{x_i,G}(t))^2 \tag{3.11}$$

Dabei ist a das T-dimensionale Tupel mit S Beschleunigungsmessungen und $f_{x_i,G}(t)$ die parametrisierte Sinusfunktion [13]. Es wird festgestellt, wie nahe die Parameter die Sinusfunktion an die Beschleunigungsdaten gebracht haben [13]. Je näher, desto kleiner ist der RMSE [13]. Das beste Individuum ist das Individuum mit dem kleinsten RMSE

der Generation [13]. Sobald die Optimierung mit dem Erreichen von G_{max} oder c_{min} abgeschlossen ist, können dem besten Individuum x_0 der Generation die Parameter entnommen werden, um die $CCF = \frac{\omega}{2\pi} \cdot 60$ in cpm und $CCD = 2A$ in cm zu bestimmen [13].

Fenstergröße

Der Algorithmus wird nur für ein bestimmtes Fenster S durchgeführt [13]. Dieses Fenster wurde auf drei Sekunden festgelegt (S_{len}) [13]. Sobald ein neues Datenfenster verfügbar ist, wird das Fenster verschoben, sodass sich 2 Sekunden überlappen und 1 Sekunde eine neue Datenreihe darstellt [13]. Nachdem der Algorithmus abgeschlossen ist, wird ein neuer Durchlauf gestartet, wobei nur ein Teil ϵ der Population neu initialisiert wird [13]. Damit wird sichergestellt, dass gute Lösungen weiterhin in der Population bestehen, um neue Lösungen zu finden, die zum neuen Fenster passen und auf der anderen Seite werden neue Individuen erzeugt, um eine große Diversität in der Population zu gewährleisten [13]. Diese neue Population wird dann auf das neue Datenfenster evaluiert [13].

CCD und CCF

Wenn nun eine Lösung gefunden wurde, können die CCD und CCF abgeleitet werden, indem die Parameter vom besten Individuum entnommen werden [13]. In dem Paper war das Ziel herauszufinden, ob das Erfassen von CCD und CCF mithilfe eines Handgelenksensors realisierbar ist [13]. Dafür muss verglichen werden, ob der vorhergesagte Wert mittels der Sinusfunktion tatsächlich dem entspricht, was das Modell erfasst hat [13]. Zur Referenz wird ein Trainingsmodell der Resusci Anne verwendet, um die CCD und CCF aufzuzeichnen [13]. Es wird erfasst, wann eine Kompression begonnen und aufgehört hat, um einen Kompressionszyklus aufzunehmen und die Chest Compression Frequency zu vergleichen [13]. Zudem wird erfasst, wie tief die Kompression durchgeführt wurde, um damit die CCD zu vergleichen [13]. Mithilfe der Differenz kann nun bestimmt werden, wie groß der Fehler in der Vorhersage im Vergleich zu den Werten des Referenzmodells ist [13]. Da das Fenster jedoch eine feste Größe von 3 Sekunden hat, kann nicht sichergestellt werden, dass der Kompressionszyklus in dem Fenster von 3 Sekunden des Referenzmodells enthalten ist [13]. Deshalb müssen mehrere Vorhersagen aus verschiedenen Fenstern genommen werden, um den gesamten Kompressionszyklus zu repräsentieren [13]. Dazu muss nun der gewichtete Mittelwert berechnet werden, siehe Gleichung 3.12, um den

tatsächlichen CCD/CCF-Wert abhängig von der Zeit zu erhalten, die sich zwischen dem Kompressionszyklus des Referenzmodells und dem Fenster überschneidet [13].

$$p(t) = \frac{1}{\sum_{i=1}^{n} \sigma_i} \sum_{i=1}^{n} \sigma_i f_i \tag{3.12}$$

σ wird durch die Zeit, welche sich zwischen dem Kompressionszyklus vom Referenzmodell und dem Fenster überlappt festgelegt [13]. Diese Zeit fungiert gleichzeitig als Gewichtungsfaktor, der mit der CCD/CCF des Fensters multipliziert wird [13]. Daher gibt p(t) die tatsächliche CCF/CCD in Abhängigkeit von der Überlappung wieder [13]. Nun kann der absolute Fehler berechnet werden, indem die Differenz zwischen der CCD/CCF des Vorhersagemodells und der CCD/CCF des Referenzmodells genommen wird [13]. Der absolute Fehler kann dann als Metrik verwendet werden, um zu entscheiden, ob ein Handgelenksensor für solche Vorhersagen geeignet ist [13].

Diese Arbeit legt jedoch nicht den Fokus darauf zu analysieren, ob ein Handgelenksensor geeignet wäre für Vorhersagen über die CCD/CCF, sondern eher, ob signifikante Unterschiede in den absoluten Fehlern existieren, wenn unterschiedliche PRNGs verwendet werden.

3.1.3 RNGs

Da das Optimierungsproblem und die Metriken zur statistischen Analyse bereits festgelegt wurden, ist es nun an der Zeit, die PRNGs vorzustellen, die verwendet werden. Für diese Analyse werden die PRNGs Mersenne Twister, KISS und Xorshift ausgewählt. Der Grund für diese Auswahl ist, dass sich diese PRNGs in der Art und Weise unterscheiden, wie sie Zahlen erzeugen. Der Mersenne Twister und Xorshift wurden bereits in verwandten Arbeiten untersucht, was ihre Auswahl zusätzlich begründet.

Mersenne Twister
Der Mersenne Twister (MT19937) ist ein schneller Generator, der eine riesige Periodenlänge von $2^{19937} - 1$ hat [16]. Der Generator hat einen 624 großen Zustands-Vektor, welcher mit pseudozufälligen Zahlen gefüllt wird [16]. Diese Pseudozufallszahlen können anschließend aus dem Vektor ausgelesen werden [16].

KISS
KISS (Keep it Simple Stupid), ist ein einfacher und effizienter PRNG [23]. Der Generator besteht aus vier Sub-Generatoren [23]:

1. Ein Linear Congruential Generator CONG (2^{32})
2. Ein 3-shift Register SHR3 (2^{32})
3. Zwei Multiply-with-carry Generatoren (2^{16})

Die Ausgaben der Generatoren werden mittels Addition und Xor kombiniert, um eine Ausgabe zu generieren [23]. Der Generator hat eine Periodenlänge von ungefähr 2^{123} [23].

Xorshift
Xorshift ist eine Beschreibung von PRNGs, die einfach und schnell Zufallszahlen generieren können, mittels bitweiser Operatoren wie Xor und Bitshiften [14]. Diese Generatoren können unterschiedliche Periodenlängen von $2^{32}-1$, $2^{64}-1$, $2^{128}-1$ bis $2^{192}-1$ haben [14]. Für diese Analyse wird konkret der SHR3 PRNG verwendet, welcher eine Periodenlänge von $2^{32}-1$ hat [23].

3.2 Statistische Analyse

Um die Ergebnisse später vergleichen und fundierte Entscheidungen treffen zu können, ist es notwendig, statistische Verfahren einzusetzen, um signifikante Unterschiede aufzudecken, die nicht mit bloßem Auge erkennbar sind.

3.2.1 Kruskal Wallis Test

Der Kruskal-Wallis-Test ist eine statistische Methode, die dazu dient, signifikante Unterschiede zwischen mehreren Gruppen zu identifizieren [31]. Diese Testmethode wird angewendet, wenn die Annahmen für eine Varianzanalyse nicht erfüllt sind, insbesondere wenn die Daten nicht normalverteilt sind [31].

Der Kruskal-Wallis-Test formuliert zwei Hypothesen [27]:

- H_0: Es gibt keinen signifikanten Unterschied zwischen den Gruppen.

- H_1: Es gibt einen signifikanten Unterschied zwischen mindestens zwei Gruppen.

Die Durchführung des Tests erfolgt in mehreren Schritten. Zuerst werden alle Daten aller Gruppen kombiniert und nach Größe sortiert [24]. Dann werden den Daten Ränge zugewiesen, von niedrig bis hoch [24]. Anschließend wird ein p-Wert berechnet der auf den Rängen basiert [24]. Der p-Wert wird dann mit dem Signifikanzniveau α verglichen, um zu entscheiden, ob die Nullhypothese N_0 angenommen oder verworfen wird [24]. Das Signifikanzniveau α wird dabei meistens auf 0,05 festgelegt [20].

Wenn der p-Wert kleiner als α ist, wird die Nullhypothese H_0 abgelehnt, und die Alternativhypothese H_1 wird akzeptiert [24]. Falls der p-Wert größer oder gleich α ist, wird die Nullhypothese angenommen [24]. Es ist zu beachten, dass der Kruskal-Wallis-Test erfordert, dass die Daten ordinalskaliert sind [27].

Dieser Test eignet sich für die Analyse, ob es einen signifikanten Unterschied zwischen den absoluten Fehlern der CCD- und CCF-Werte der PRNGs gibt.

3.3 Implementierung

Nach Festlegung des Optimierungsproblems und des zugehörigen Algorithmus wird nun die Implementierung erläutert, die in Python erfolgt. Der Code für das Laden der Sensordaten, die Implementierung des Algorithmus, sowie das Speichern der Ergebnisse werden von Prof. Dr. Christian Lins bereitgestellt. Der Code bezüglich der PRNGs und der Verarbeitung und Analyse der Daten wird hingegen selbst implementiert.

3.3.1 Bibliotheken

Für die Implementierung der PRNGs werden die Bibliotheken „random“ und „simplerandom“ verwendet. Standardmäßig wird in der Python-Bibliothek „random“ der Mersenne Twister als PRNG verwendet [21]. Die PRNGs SHR3 und KISS hingegen werden mithilfe der Bibliothek „Simplerandom“ implementiert [17].

Die statistische Analyse erfolgt mittels der „scipy“ Bibliothek, die eine Vielzahl von statistischen Methoden bietet, wie zum Beispiel dem „KWT“, um Daten zu analysieren [25].

3.3.2 Algorithmus

Für ein reibungsloses Durchführen der Tests müssen bestimmte Stellen des Algorithmus geändert werden, damit die PRNGs einfach gewechselt werden können. Dazu werden die Stellen bei der Initialisierung des Algorithmus, der Initialisierung eines Individuums, der Selektion und der Mutation angepasst.

Bei der Initialisierung des Algorithmus muss der RNG als Argument übergeben werden. Dies wird ermöglicht, indem beim __init__ RNG als Parameter festgelegt wird und anschließend gesetzt wird, siehe Quellcode 3.1 Zeile 5 und 8.

```
1  class EvoStrategyAlgorithm(object):
2      """description of class"""
3
4      def __init__(self, fn_cost, k, bounds, popsize, RNG):
5          self.fn_cost = fn_cost
6          self.bounds = bounds
7          self.RNG = RNG
8
9          self.k = k
10         self.NP = popsize
11         self.individuals = np.ndarray([popsize + k + 1, len(bounds)], dtype
       =np.float32)
12         self.individuals_cost = np.zeros(popsize + k + 1, dtype=np.float32)
13
14         self.__init_random()
```

Quellcode 3.1: Initialisierung des Algorithmus

Damit der festgelegte RNG in den einzelnen Schritten verwendet werden kann, wird jeweils der self.RNG verwendet, um die Methoden durchzuführen. Für die Initialisierung der Population, wäre das einmal bei der Methode __new_random_individual, siehe Quellcode 3.2 Zeile 4. Hier wird ein Individuum initialisiert, indem zufällige Zahlen innerhalb der Grenzen, die in Tabelle 3.1 dargestellt wurden, generiert werden.

```
1      def __new_random_individual(self):
2          individual = np.ndarray(shape=[len(self.bounds),], dtype=np.float32
       )
3          for k in range(0, len(self.bounds)):
4              individual[k] = self.RNG.uniform(self.bounds[k][0], self.bounds
       [k][1])
5          return individual
```

Quellcode 3.2: Erstellung eines Individuums

Wie vorhin schon erwähnt, wird nur ein Teil der Bevölkerung neu initialisiert. Dafür wird ϵ genutzt, um zu bestimmen, wie hoch die Rate der Neuinitalisierung ist. Um zu entscheiden, ob ein Individuum mittels der Methode __new_random_individual neu initialisiert wird, wird eine zufällige Zahl zwischen 0 und 1 generiert. Um den RNG verwenden zu können, muss bei der Methode extinct auch der self.RNG verwendet werden, siehe Quellcode 3.3 Zeile 3.

```
1    def extinct(self, EP):
2        for i in range(0, len(self.individuals)):
3            if self.RNG.uniform(0, 1) < EP:
4                self.individuals[i] = self.__new_random_individual()
```

Quellcode 3.3: Reinitialisieren eines Bruchteils der Populations

Die Selektion muss mehrere zufällige Individuen aus der Population auswählen, damit dann die Rekombination durchgeführt werden kann, um neue Individuen zu generieren. Dafür werden zufällige Zahlen benötigt, welche mittels des self.RNG generiert werden müssen. Diese Zahlen repräsentieren den Index des Individuums in der Population, in diesem Fall ein Array, siehe Quellcode 3.4 Zeile 5.

```
1    def __selection(self, exclude):
2        r = 0
3        sanity = 100
4        while True:
5            r = self.RNG.randint(0, self.NP) # This includes the random
     individual
6            sanity = sanity - 1
7            if not r in exclude or sanity < 0:
8                break
9
10       return self.individuals[r], r
```

Quellcode 3.4: Selektion zufälliger Individuen

Für die Mutation werden Zahlen zwischen M und $2 - M$ benötigt, um an einem Individuum die Feinabstimmung durchzuführen und den Mutantenvektor zu generieren. Hierfür wird auch ein RNG benötigt, um eine zufällige Zahl zwischen diesen Grenzen zu generieren. Hierbei wird der self.RNG in Zeile 4 des Quellcodes 3.5 verwendet.

```
1    def __mutate(self, individual, M = 0.999):
2        mutant = np.ndarray(individual.shape)
3        for k in range(0, len(individual)):
4            mutant[k] = self.RNG.uniform(M, 2 - M) * individual[k]
```

```
5         return mutant
```

Quellcode 3.5: Erstellung des Mutantenvektor

3.3.3 Zufallszahlengenerator

Um den Code übersichtlicher zu gestalten und die Lesbarkeit zu verbessern, wird eine Klasse namens 'RNG' implementiert. Diese Klasse dient dazu, mehrere PRNGs in einer einzigen Klasse zu Kapseln. Siehe Quellcode 3.6.

```
1  import random
2  import simplerandom.random as srr
3
4  class RNG():
5
6      def __init__(self, rngName):
7          self.counter = 0
8          self.rng = None
9
10         if (rngName == "MST"):
11             self.rng = random
12
13         elif (rngName == "KISS"):
14             self.rng = srr.KISS()
15
16         elif (rngName == "SHR3"):
17             self.rng = srr.SHR3()
18
19         else:
20             ValueError("Invalid RNG!")
21
22     def randint(self, low, high):
23         self.counter += 1
24         return self.rng.randint(low, high)
25
26     def uniform(self, low, high):
27         self.counter += 1
28         return self.rng.uniform(low, high)
```

Quellcode 3.6: Klasse zum Kapseln aller PRNGs

3.3.4 Datenspeicherung

Die Ergebnisse werden als zwei CSV-Dateien gespeichert, siehe Quellcode 3.7. Einmal als _pred_hand.csv und _cmp_hand.csv.

```
# Save Predictions
pred_hand.to_csv('results/thirdRun/prediction/'+ str(rngName) +
                 '/subj' + str(i) + '/' + '_pred_hand' + str(j) +'.csv')

anne = load_anne_file(base + 'anne.csv')
cmp_hand = compare(anne, pred_hand, Slen)

# Save Comparison
cmp_hand.to_csv('results/thirdRun/compare/' + str(rngName) +
                '/subj' + str(i) + '/' + '_cmp_hand_' + str(j) +'.csv')
```

Quellcode 3.7: Speichern der Ergebnisse

Die Datei _pred_hand.csv, wie in Abbildung 3.2 dargestellt, umfasst mehrere Spalten. Dazu gehören der Zeitstempel, die vorhergesagten Werte für CCF und CCD, der RMSE, der Vektor x, welcher die Parameterwerte enthält, sowie die Anzahl der benötigten Generationen.

	A	B	C	D	E	F	G
1		ts	pred_ccf	pred_ccd	fitting_error	x	generations
2	0	355	114.907861	0.03210387	[0.03384698	[0.03384698	1
3	1	356	111.542007	0.04542304	[0.02271152	[0.02271152	1
4	2	357	116.660423	0.04087629	[0.02043815	[0.02043815	1
5	3	358	117.548876	0.04619856	[0.02309928	[0.02309928	1
6	4	359	118.242495	0.04862343	[0.03604963	[0.03604963	1
7	5	360	117.79439	0.02333477	[0.02179735	[0.02179735	1
8	6	361	110.240437	0.05291247	[0.04801414	[0.04801414	1
9	7	362	113.762948	0.04490776	[0.02245388	[0.02245388	1
10	8	363	112.322361	0.02946574	[0.0268794	[0.0268794	1

Abbildung 3.2: Aufbau der _pred_hand.csv Datei

Die Datei _cmp_hand.csv, siehe Abbildung 3.3 enthält ebenfalls einen Zeitstempel sowie die CCF- und CCD-Werte sowohl aus der Vorhersage als auch aus dem Referenzmodell. Darüber hinaus sind die jeweiligen absoluten Fehler der CCF und CCD in dieser Datei aufgeführt.

	A	B	C	D	E	F	G	H
1		ts	ccf_model	ccd_model	ccf_anne	ccd_anne	ccf_error	ccd_error
2	0	0	117.787522	0.04631401	110	0.05	7.78752238	0.00368599
3	1	2.15	117.86192	0.03938559	116	0.048	1.86192038	0.00861441
4	2	2.63	117.521993	0.03969786	116	0.048	1.52199299	0.00830214
5	3	3.11	115.425774	0.04162356	120	0.046	4.57422595	0.00437644
6	4	3.63	115.108076	0.04136003	116	0.047	0.89192437	0.00563997
7	5	4.17	113.932591	0.040385	115	0.052	1.06740852	0.011615
8	6	4.69	113.365122	0.0410208	112	0.049	1.36512183	0.0079792

Abbildung 3.3: Aufbau der _cmp_hand.csv Datei

3.3.5 Datenverarbeitung

Damit die Daten analysiert werden können, müssen die wichtigsten Daten aus den gespeicherten CSV-Dateien extrahiert werden und berechnet werden. Dafür wird das Skript 3.8 verwendet.

```
class statisticalAnalysis():

    def __init__(self):
        self.base = "path_to_root_folder"

    def read_file(self, path, filetype):
        if (filetype == "csv"):
            fileContent = pd.read_csv(path)
        else:
            fileContent = pd.read_excel(path)

        return fileContent

    def calculate_pro_eintrag(self,calculate):
        folderNames = ["MST","KISS","SHR3"]

        for j in range(1, 7):
            data = {}

            for name in folderNames:
                ccf_avg = []
                ccd_avg = []

                for i in range(0, 50):
```

```
25                      ccf = []
26                      ccd = []
27
28                      fileContent = self.read_file(self.base + "results/
    compare/"+ str(name) +
29                                                   "/subj" + str(j) +
30                                                   "/_cmp_hand" + str(i) +
31                                                   ".csv","csv")
32
33                      ccf = fileContent['ccf_error'].array
34                      ccd = fileContent['ccd_error'].array
35
36                      ccf_mask = ~np.isnan(ccf)
37                      ccd_mask = ~np.isnan(ccd)
38
39                      # Apply the boolean mask
40                      ccf_avg.append(ccf[ccf_mask])
41                      ccd_avg.append(ccd[ccd_mask])
42
43                  if (calculate == "mean"):
44                      ccf_avg = np.mean(ccf_avg,axis=0, dtype=np.float64)
45                      ccd_avg = np.mean(ccd_avg,axis=0, dtype=np.float64)
46                  elif (calculate == "median"):
47                      ccf_avg = np.median(ccf_avg,axis=0)
48                      ccd_avg = np.median(ccd_avg,axis=0)
49                  elif (calculate == "variance"):
50                      ccf_avg = np.var(ccf_avg,axis=0)
51                      ccd_avg = np.var(ccd_avg,axis=0)
52
53
54                  data = {
55                      "ccf":ccf_avg,
56                      "ccd":ccd_avg,
57                  }
58
59                  frame=pd.DataFrame(data)
60                  frame.to_csv(self.base + "calculated/"+ calculate + "/" +
    str(name) + "/subj" + str(j) + '.csv')
61          print("Done")
```

Quellcode 3.8: Skript zum Berechnen des Durchschnitts, Median und Varianz der absoluten Fehler

Zuerst wird die Datei _cmp_hand.csv eingelesen. Die Spalten „ccf_error“ und „ccd_-error“ werden aus der Datei extrahiert und einem Array zugewiesen. Dadurch können mithilfe der numpy-Bibliothek [18] Durchschnitt, Median oder Varianz berechnet werden. Die Ergebnisse werden dann in einem Dictionary gespeichert, um damit in Zeile 59 ein DataFrame zu erstellen. Im Anschluss wird dieses DataFrame als die Datei „subj.csv“ gespeichert, wie in Abbildung 3.4 dargestellt. Diese Datei enthält die Werte für die absoluten Fehler von CCD und CCF pro Eintrag und wird einmal für den Durchschnitt, Median und Varianz erstellt.

	A	B	C
1		ccf	ccd
2	0	7.21735756	0.00849548
3	1	1.72680016	0.00691218
4	2	1.68044421	0.00682433

Abbildung 3.4: Aufbau der subj.csv Datei

3.4 Durchführung

3.4.1 Aufbau

RNGs sind aufgrund ihrer stochastischen Natur unvorhersehbar. Daher ist es von entscheidender Bedeutung, mehrere Durchläufe durchzuführen, um die Entscheidungen nicht auf Einzelfälle zu stützen. In diesem Experiment werden PRNGs auf sechs verschiedene Datensätze (Subjekte) angewendet. Jeder Datensatz wird jeweils 50 Mal von allen PRNGs durchlaufen. Nach Abschluss der Berechnung eines Datensatzes erfolgt eine Neuinitialisierung des PRNG (neuer Seed). Nach Abschluss aller Durchläufe werden der Durchschnitt, Median und die Varianz der absoluten Fehler (CCD und CCF) für jeden Eintrag über alle Läufe hinweg für jeden PRNG berechnet. Zudem wird auch der Durchschnitt der

Anzahl an benötigten Generationen für jeden Eintrag berechnet, um zu untersuchen, ob Unterschiede in der Konvergenzgeschwindigkeit existieren.

3.4.2 Verwendete Ressourcen

Für die Durchführung sind leistungsfähige Hardware-Ressourcen erforderlich. Zu diesem Zweck wird das „Creative Space for Technical Innovations“ (CSTI) der HAW-Hamburg genutzt [1]. Durch die Verwendung eines VPNs [19] kann eine SSH-Verbindung zur virtuellen Umgebung des CSTI hergestellt werden, um Berechnungen durchzuführen. Damit der Code auf der Remote-Maschine ausgeführt werden kann, muss er von der lokalen Maschine auf die Remote-Maschine übertragen werden. Dies wird durch das Datentransferprotokoll „Secure File Transfer Protocol“ (SFTP) [3] ermöglicht.

Zum Ausführen wird anschließend eine Verbindung mittels PuTTY erstellt [2]. Damit die PuTTY-Verbindung nicht für die gesamte Dauer der Durchführung aufrechterhalten werden muss, wird das Skript mittels des „nohup“-Befehls als neuer Prozess gestartet. Auf diese Weise kann die Remote-Maschine nun Berechnungen durchführen, ohne dabei eine dauerhafte SSH-Verbindung aufrechtzuerhalten.

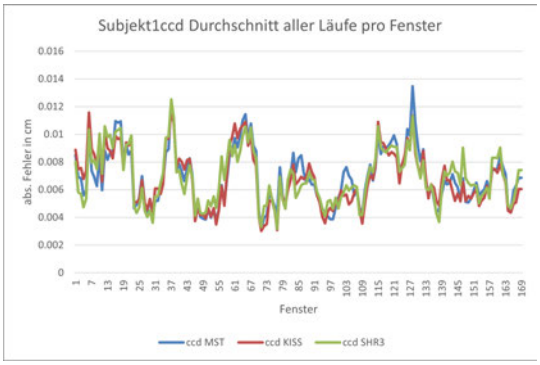
4 Ergebnisse

4.1 Grafische Darstellung der Ergebnisse

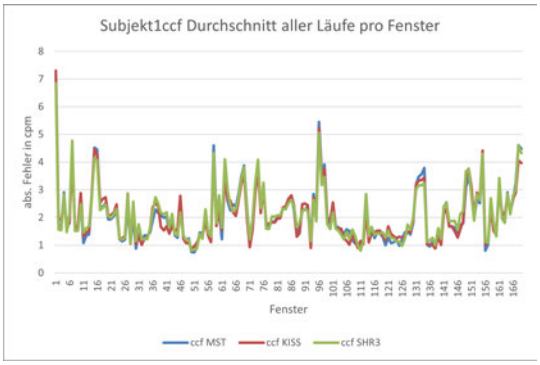
4.1.1 Absolute Fehler

Die nachfolgenden Abbildungen zeigen, dass für die Subjekte 1 bis 6 kleine, jedoch nicht signifikante Unterschiede in den absoluten Fehlern bezüglich Durchschnitt, Median und Varianz zwischen den PRNGs bestehen.

Subjekt 1

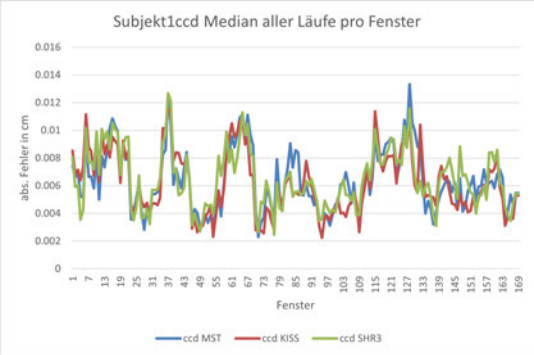


(a) CCD von Subjekt 1

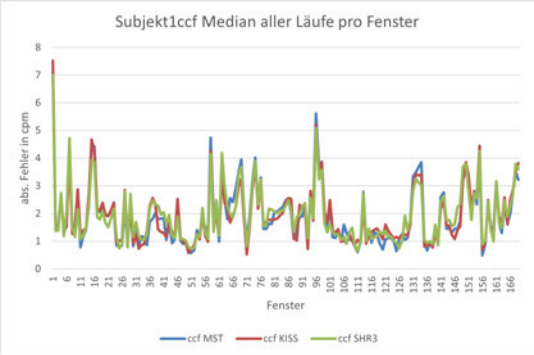


(b) CCF von Subjekt 1

Abbildung 4.1: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt1

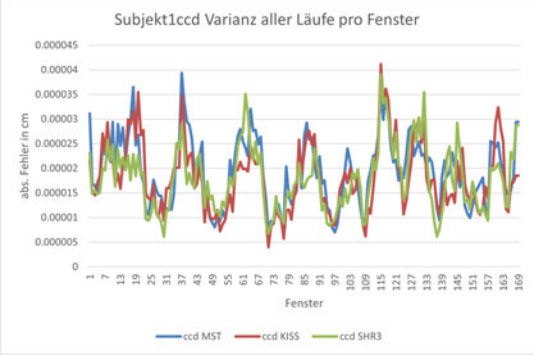


(a) CCD von Subjekt 1

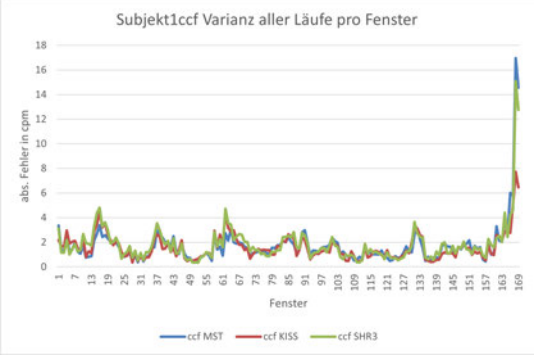


(b) CCF von Subjekt 1

Abbildung 4.2: Median des absoluten Fehlers von der CCD & CCF für Subjekt1



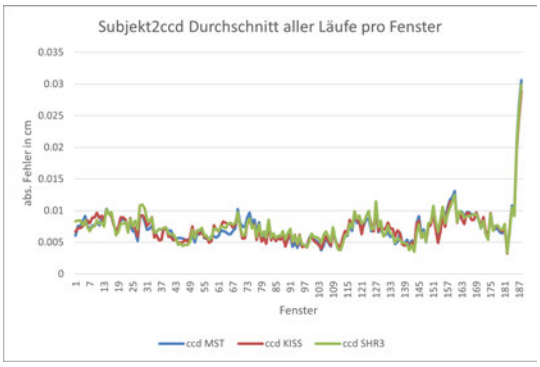
(a) CCD von Subjekt 1



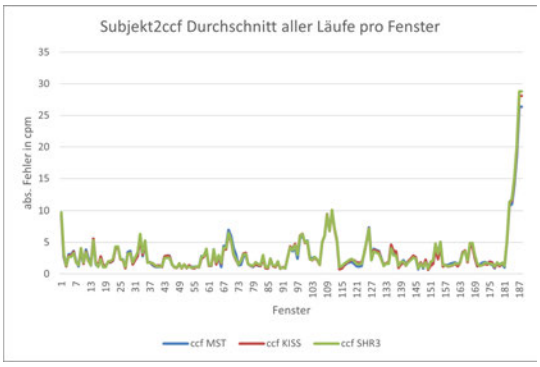
(b) CCF von Subjekt 1

Abbildung 4.3: Varianz des absoluten Fehlers von der CCD & CCF für Subjekt1

Subjekt 2

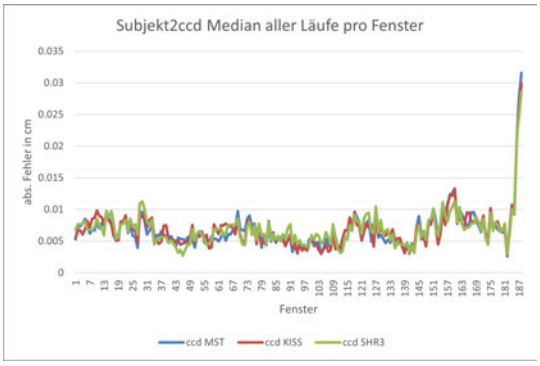


(a) CCD von Subjekt 2

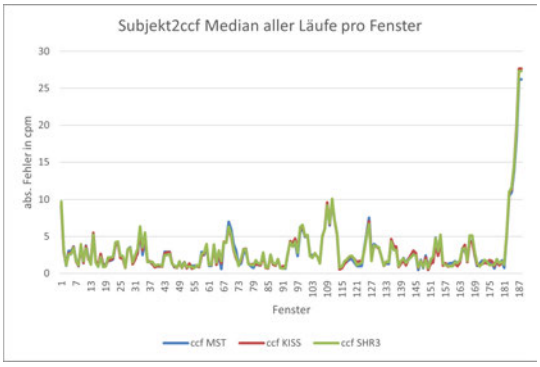


(b) CCF von Subjekt 2

Abbildung 4.4: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt2

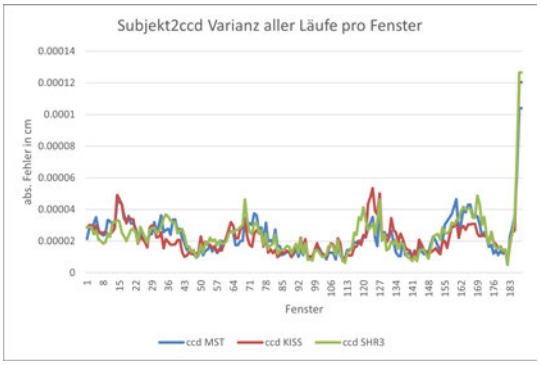


(a) CCD von Subjekt 2

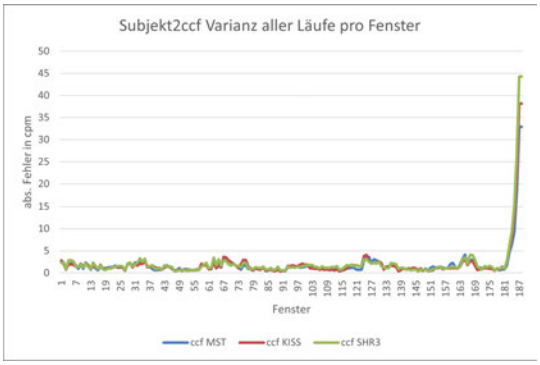


(b) CCF von Subjekt 2

Abbildung 4.5: Median des absoluten Fehlers von der CCD & CCF für Subjekt2



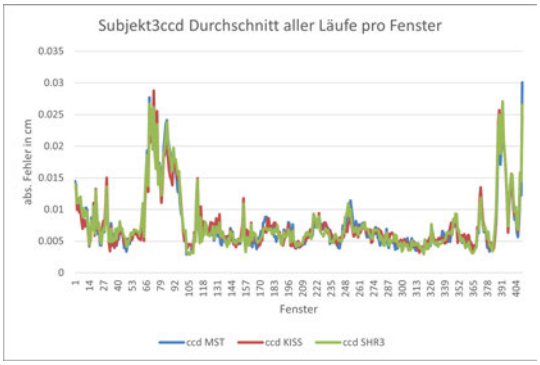
(a) CCD von Subjekt 2



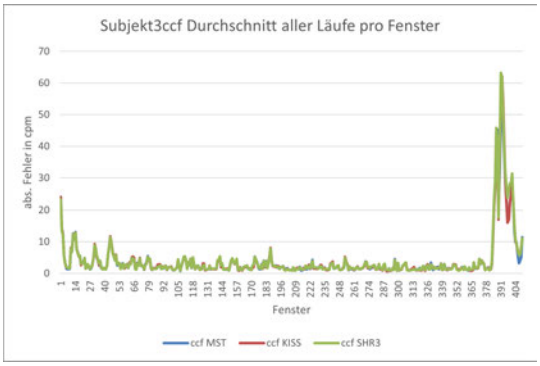
(b) CCF von Subjekt 2

Abbildung 4.6: Varianz des absoluten Fehlers von der CCD & CCF für Subjekt2

Subjekt 3

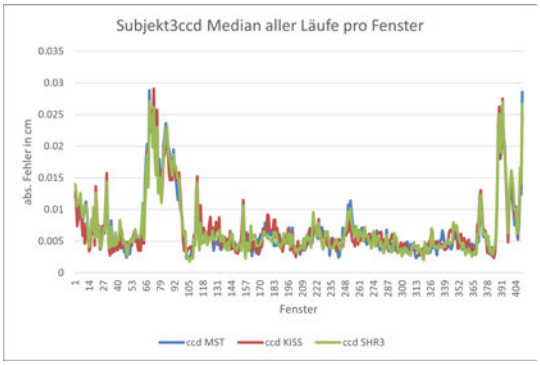


(a) CCD von Subjekt 3

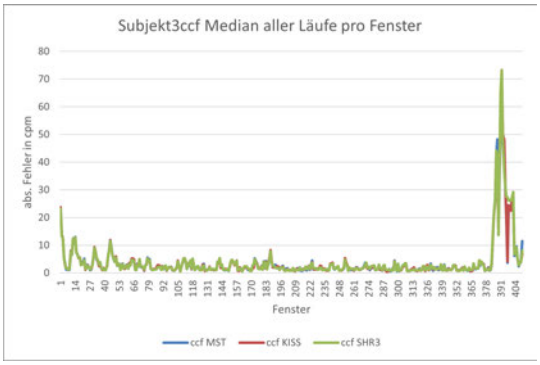


(b) CCF von Subjekt 3

Abbildung 4.7: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt3

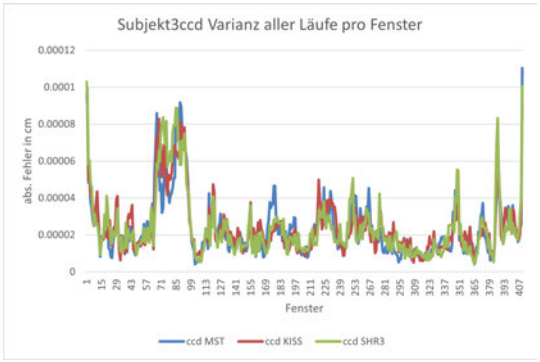


(a) CCD von Subjekt 3

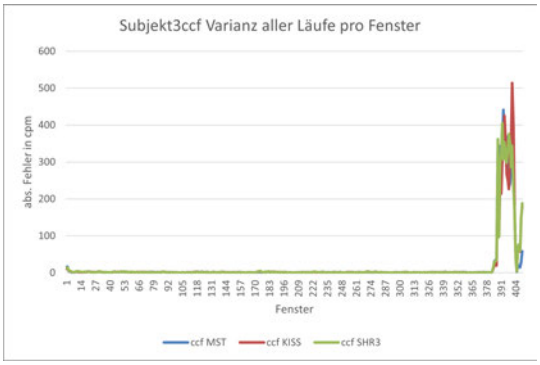


(b) CCF von Subjekt 3

Abbildung 4.8: Median des absoluten Fehlers von der CCD & CCF für Subjekt3



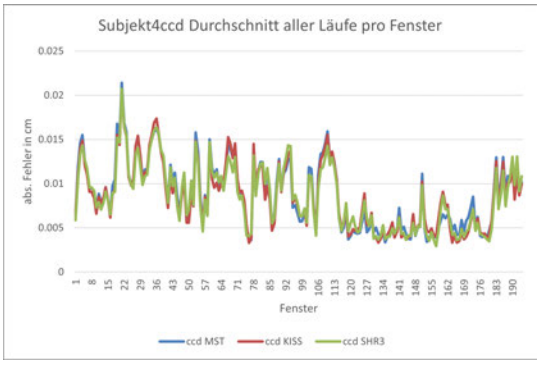
(a) CCD von Subjekt 3



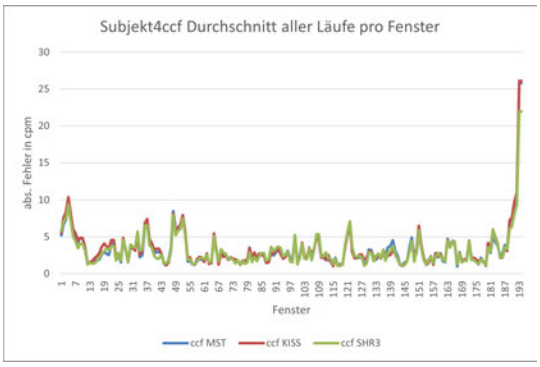
(b) CCF von Subjekt 3

Abbildung 4.9: Varianz des absoluten Fehlers von der CCD & CCF für Subjekt3

Subjekt 4

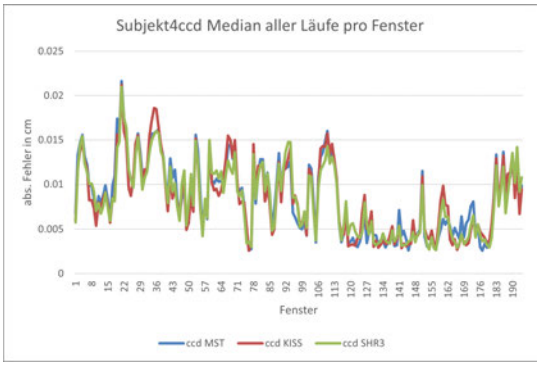


(a) CCD von Subjekt 4

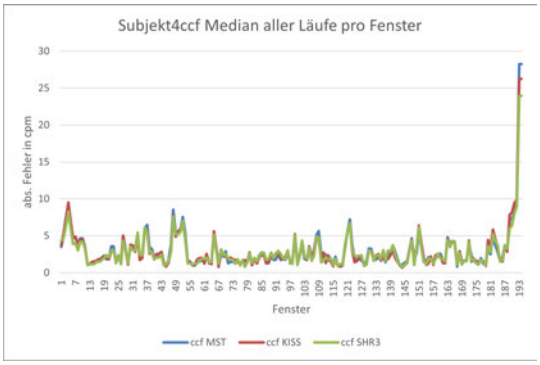


(b) CCF von Subjekt 4

Abbildung 4.10: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt4

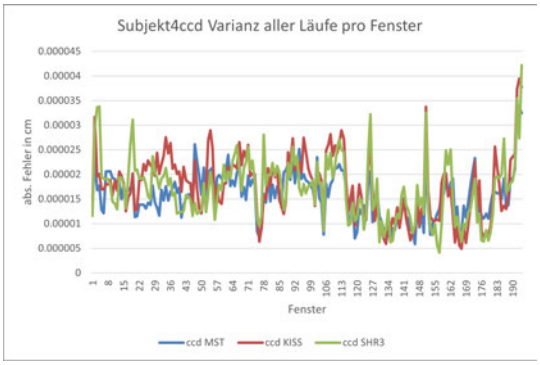


(a) CCD von Subjekt 4

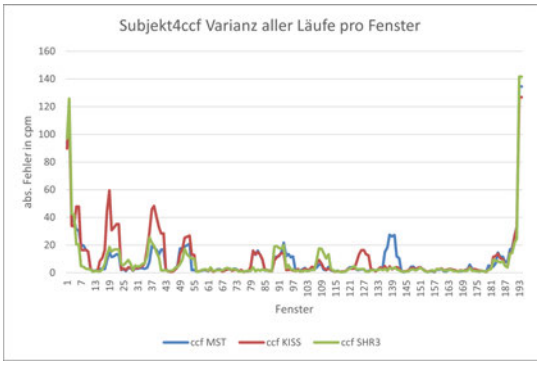


(b) CCF von Subjekt 4

Abbildung 4.11: Median des absoluten Fehlers von der CCD & CCF für Subjekt4



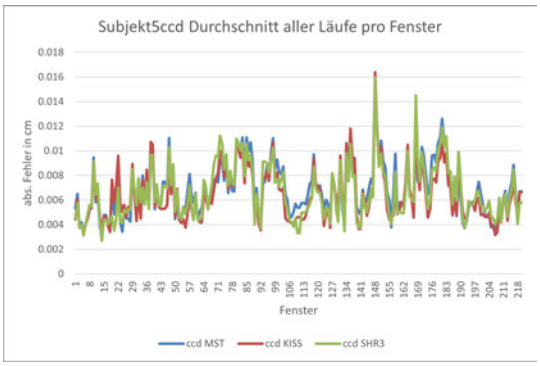
(a) CCD von Subjekt 4



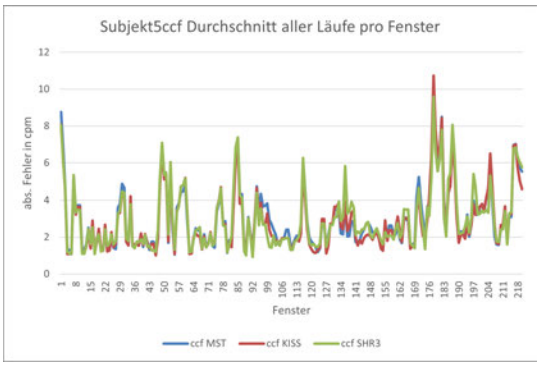
(b) CCF von Subjekt 4

Abbildung 4.12: Varianz des absoluten Fehlers von der CCD & CCF für Subjekt4

Subjekt 5

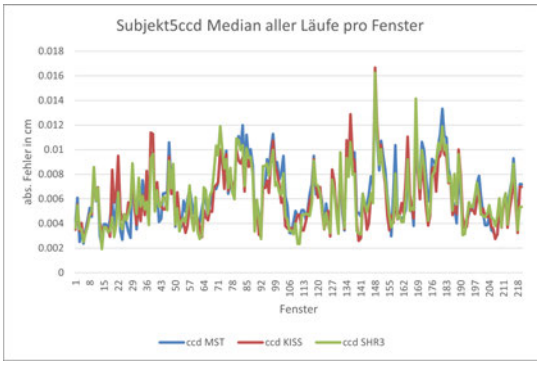


(a) CCD von Subjekt 5

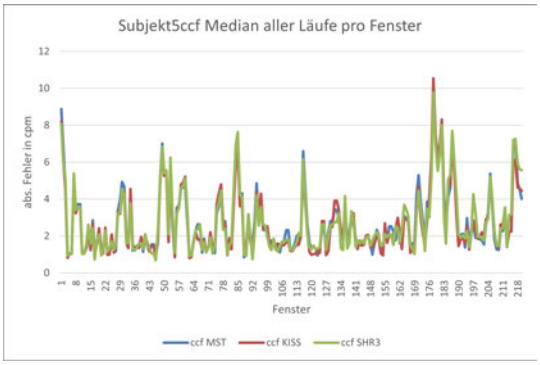


(b) CCF von Subjekt 5

Abbildung 4.13: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt5

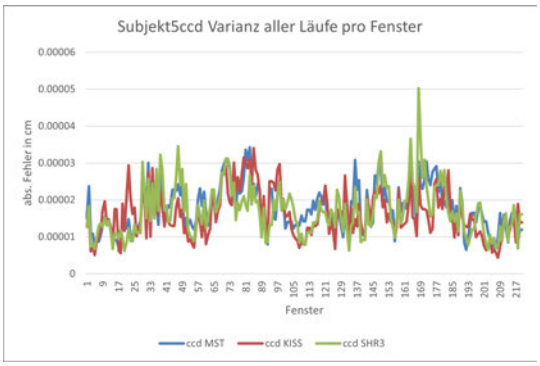


(a) CCD von Subjekt 5

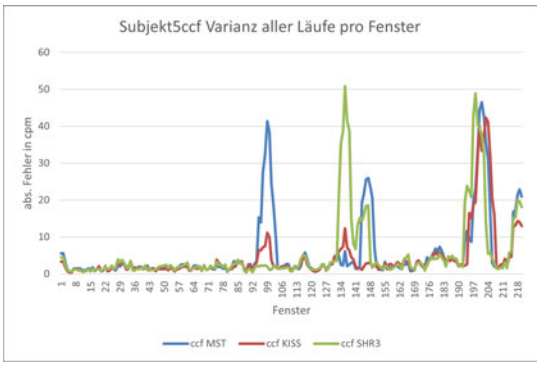


(b) CCF von Subjekt 5

Abbildung 4.14: Median des absoluten Fehlers von der CCD & CCF für Subjekt5



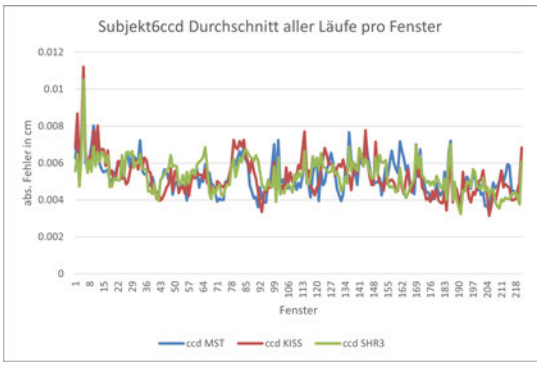
(a) CCD von Subjekt 5



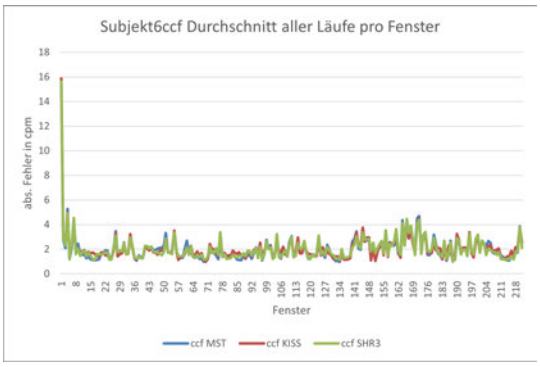
(b) CCF von Subjekt 5

Abbildung 4.15: Varianz des absoluten Fehlers von der CCD & CCF für Subjekt5

Subjekt 6

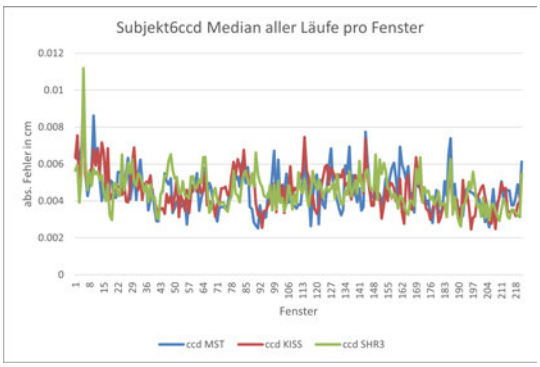


(a) CCD von Subjekt 6

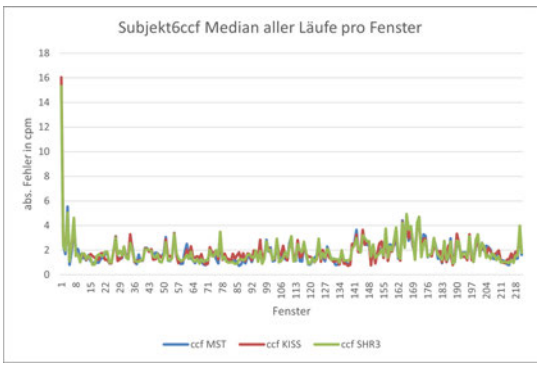


(b) CCF von Subjekt 6

Abbildung 4.16: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt6

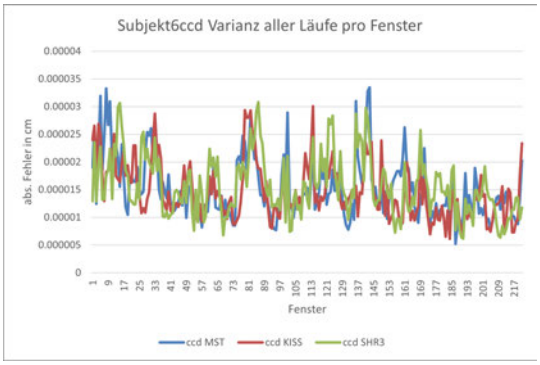


(a) CCD von Subjekt 6

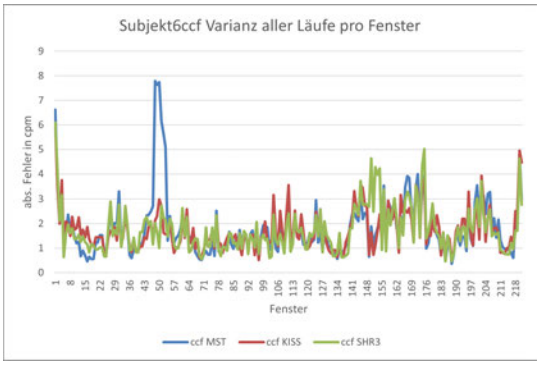


(b) CCF von Subjekt 6

Abbildung 4.17: Median des absoluten Fehlers von der CCD & CCF für Subjekt6



(a) CCD von Subjekt 6

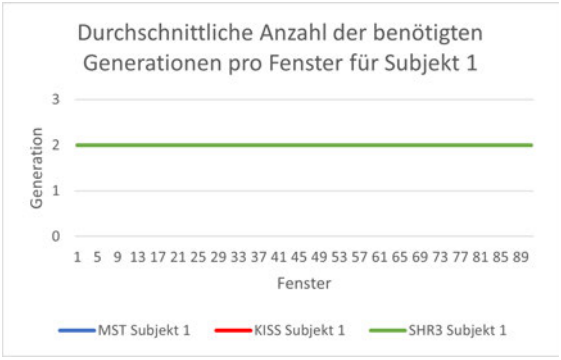


(b) CCF von Subjekt 6

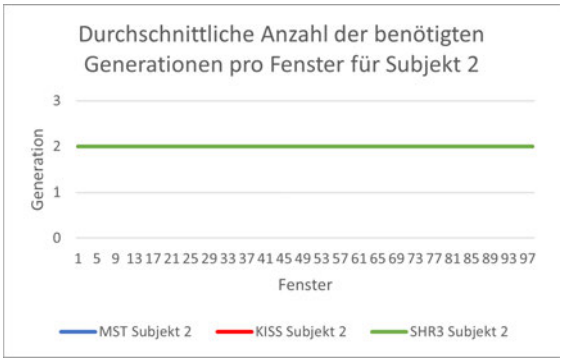
Abbildung 4.18: Varianz des absoluten Fehlers von der CCD & CCF für Subjekt6

4.1.2 Benötigte Generationen

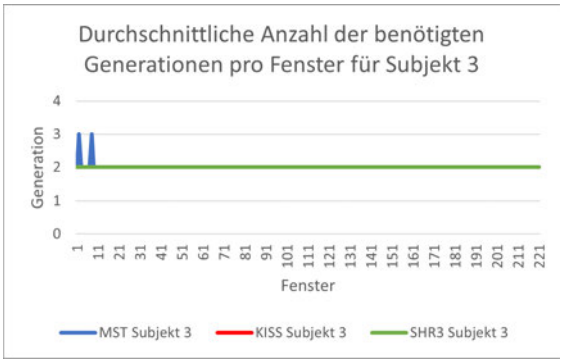
Bei den benötigten Generationen konnte kein signifikanter Unterschied festgestellt werden. Für alle Fenster unter allen Subjekten, war die benötigte Anzahl an Generationen für alle PRNGs identisch, siehe Abbildung 4.19a und 4.19a für Subjekt 1 und 2 als Beispiel. Es konnte jedoch ein Unterschied bei Subjekt 3 festgestellt werden, siehe Abbildung 4.19c. Hier hat der Mersenne Twister für zwei Fenster eine Generation mehr benötigt.



(a) Subjekt 1



(b) Subjekt 2



(c) Subjekt 3

Abbildung 4.19: Die maximale Generation die bei jedem Fenster im Durchschnitt erreicht wurde

4.2 Statistische Ergebnisse

4.2.1 Aggregierter Durchschnitt

Die Werte in Tabelle 4.1 repräsentieren den aggregierten Durchschnitt der durchschnittlichen absoluten Fehler der PRNGs über sämtliche Subjekte hinweg. Es sind kleine, aber nicht signifikante Unterschiede in der CCD sichtbar.

MST CCD	KISS CCD	SHR3 CCD
0,00714985 cm	0,007073895 cm	0,007145954 cm

Tabelle 4.1: Aggregierter Durchschnitt der durchschnittlichen absoluten Fehler für die CCD

Bei der CCF, siehe Tabelle 4.2, ist jedoch ein Unterschied bemerkbar. Hier hat der PRNG SHR3 einen höheren absoluten Fehler im Vergleich zu den anderen PRNGs.

MST CCF	KISS CCF	SHR3 CCF
3,05010817 cpm	3,08349125 cpm	3,111243172 cpm

Tabelle 4.2: Aggregierter Durchschnitt der durchschnittlichen absoluten Fehler für die CCF

4.2.2 p-Werte

In Tabelle 4.3 sind die p-Werte des KWT für die durchschnittlichen absoluten Fehler der CCD- und CCF-Werte zwischen den PRNGs dargestellt. Da alle Werte größer als das Signifikanzniveau von 0,05 sind, kann für jedes Subjekt angenommen werden, dass die absoluten Fehler der CCD- und CCF-Werte unter den PRNGs keine signifikanten Unterschiede aufzeigen.

Subjekt	p-Wert CCD	p-Wert CCF
1	0,78	0,78
2	0,48	0,88
3	0,48	0,94
4	0,81	0,81
5	0,11	0,75
6	0,50	0,44

Tabelle 4.3: p-Werte der PRNGs für Subjekt 1 bis 6 für die durchschnittlichen absoluten Fehler

Auf der Tabelle 4.4 werden die p-Werte für die Mediane der absoluten Fehler der CCD- und CCF-Werte zwischen den PRNGs gezeigt. Da alle Werte größer als das Signifikanzniveau von 0.05 sind, kann hier ebenfalls für jedes Subjekt angenommen werden, dass der Median der absoluten Fehler der CCD- und CCF- Werte unter den PRNGs, identisch sind.

Subjekt	p-Wert CCD	p-Wert CCF
1	0,43	0,79
2	0,61	0,89
3	0,65	0,98
4	0,87	0,99
5	0,23	0,88
6	0,80	0,31

Tabelle 4.4: p-Werte der PRNGs für Subjekt 1 bis 6 für den Median der absoluten Fehler

In Tabelle 4.5 sind die p-Werte für die Varianz der absoluten Fehler der CCD- und CCF-Werte zwischen den PRNGs aufgeführt. Der Wert für Subjekt 4 (CCD) und Subjekt 5 (CCD) liegt unterhalb des Signifikanzniveaus von 0,05. Daher wird die Nullhypothese verworfen und die Alternativhypothese akzeptiert, was darauf hinweist, dass bei der Varianz zwischen den PRNGs, ein signifikanter Unterschied besteht.

Subjekt	p-Wert CCD	p-Wert CCF
1	0,07	0,38
2	0,24	0,12
3	0,44	0,88
4	0,02	0,11
5	0,03	0,63
6	0,07	0,88

Tabelle 4.5: p-Werte der PRNGs für Subjekt 1 bis 6 für die Varianz der absoluten Fehler

4.2.3 Generierte Zahlen

Auf Tabelle 4.6 ist zu sehen, wie viele Zahlen durchschnittlich von den PRNGs generiert wurden.

Subjekt	MST	KISS	SHR3
1	4.001.963	4.003.885	4.008.208
2	4.310.333	4.309.494	4.309.727
3	9.716.639	9.720.869	9.715.872
4	4.571.644	4.571.636	4.571.528
5	4.663.712	4.660.309	4.659.028
6	4.754.198	4.750.614	4.750.934

Tabelle 4.6: Durchschnitt der generierten Zahlen pro PRNG je Subjekt

Auf Tabelle 4.7 ist der Median von der Anzahl erzeugter Zahlen pro PRNG zu erkennen.

Subjekt	MST	KISS	SHR3
1	3.999.784	3.999.610	4.000.078
2	4.306.751	4.306.980	4.307.377
3	9.711.868	9.712.339	9.711.960
4	4.571.545	4.571.697	4.571.724
5	4.658.985	4.658.835	4.658.205
6	4.747.635	4.747.002	4.747.914

Tabelle 4.7: Median der generierten Zahlen pro PRNG je Subjekt

Die Tabelle 4.8 zeigt die Standardabweichung von der Anzahl an generierten Zahlen je PRNG.

Subjekt	MST	KISS	SHR3
1	10.137	12.907	18.986
2	14.838	10.429	10.538
3	14.184	20.042	13.149
4	1.491	1.393	1.439
5	13.882	8.480	6.361
6	15.695	11.858	11.644

Tabelle 4.8: Standardabweichung der generierten Zahlen pro PRNG je Subjekt

5 Diskussion

5.1 Existiert ein signifikanter Unterschied?

Die grafischen und statistischen Ergebnisse deuten darauf hin, dass in diesem praktischen Anwendungsfall kein signifikanter Unterschied in den Ergebnissen festgestellt werden kann, wenn verschiedene PRNGs verwendet werden. Dies legt nahe, dass der Differential Evolution Algorithmus in bestimmten Szenarien nicht wesentlich von der Wahl des PRNGs beeinflusst wird.

Der aggregierte Durchschnitt zeigt geringfügige Unterschiede in der CCF, wie in Tabelle 4.2 dargestellt. Dies lässt darauf schließen, dass die Wahl des PRNGs insgesamt einen Einfluss auf die Ergebnisse hatte. Dieses Ergebnis ist jedoch mit Vorsicht zu interpretieren, da die unterschiedlichen Stichprobengrößen zu einer Verzerrung des Endergebnisses beigetragen haben könnten.

Die p-Werte auf Tabelle 4.5 zeigen, dass ein signifikanter Unterschied in der Varianz der CCD bei Subjekt 4 und 5 vorhanden ist. Beim Analysieren der Abbildungen 4.12a und 4.15a kann jedoch kein großer Unterschied in der Varianz zwischen den RNGs festgestellt werden.

Eine mögliche Erklärung für diese identischen Ergebnisse könnte sein, dass der Suchraum für die einzelnen Parameter der Individuen möglicherweise zu begrenzt war, was dazu führte, dass die verschiedenen PRNGs ähnliche Zahlen generierten. Es wäre auch denkbar, dass die Größe der Population und die Anzahl an durchlaufenen Generationen (siehe Tabelle 4.6) und Abbildung 4.19 in diesem Experiment nicht ausreicht, um die Periodenlänge der PRNGs vollständig auszunutzen. Es ist zu beachten, dass die beobachteten Ergebnisse auf diese speziellen experimentellen Bedingungen beschränkt sind.

5.2 Vergleich mit verwandten Arbeiten

Es kann helfen, die Ergebnisse mit den Ergebnissen verwandter Arbeiten zu vergleichen, um zu analysieren, ob Unterschiede oder Ähnlichkeiten in den Ergebnissen existieren.

Gemäß der Analyse von Cantú-Paz [4] lässt sich feststellen, dass in einigen Szenarien durch den Einsatz von suboptimalen PRNGs unter Umständen bessere Ergebnisse erzielt werden können. Dies konnte nicht festgestellt werden, da die Performance (absolute Fehler und Anzahl an benötigten Generationen) unter den PRNGs gleich war. Es ist jedoch zu beachten, dass nicht dieselben RNGs verwendet wurden, wie in der Arbeit, weshalb dies ein weiterer Faktor sein könnte, weshalb die Ergebnisse nicht gleich sind.

Die Untersuchung von Cárdenas et al. [5] legt nahe, dass der Differential Evolution Algorithmus besonders anfällig auf die Wahl des PRNGs ist. Diese Anfälligkeit konnte mittels des Tests, für diesen Anwendungsfall nicht nachgewiesen werden.

5.3 Extremfall 1

Basierend auf den Ergebnissen könnte man vermuten, dass jede beliebige Zahlensequenz verwendet werden könnte, um dieses Optimierungsproblem zu lösen. Um diese Annahme zu überprüfen, wurde ein einfacher Zähler-PRNG, als NON-RNG bezeichnet, implementiert (siehe Quellcode 5.1). Dieser PRNG hat eine Periodenlänge von 2^8 und generiert Zahlen, indem er von 0 bis 255 hochzählt und dann von vorne beginnt. Es handelt sich um einen qualitativ schlechteren PRNG, da seine Periodenlänge im Vergleich zu den zuvor verwendeten PRNGs deutlich kürzer ist und die generierten Zahlen vorhersehbar sind. Da sich die Sequenz der Zahlen nicht ändert, muss der Algorithmus mittels des NON-RNG nur einmal für jedes Subjekt durchlaufen werden.

```
1  class NON_RNG:
2      def __init__(self, period_length):
3          self.period_length = period_length
4          self.counter = 0
5
6      def randint(self, low, high):
7          if low > high:
8              raise ValueError("Low value must be less than high value")
9          self.counter = (self.counter + 1) % self.period_length
10         return low + self.counter % (high - low)
```

```
11
12      def uniform(self, low, high):
13          if low > high:
14              raise ValueError("Low value must be less than high value")
15          self.counter = (self.counter + 1) % self.period_length
16          return low + (self.counter / self.period_length) * (high - low)
```

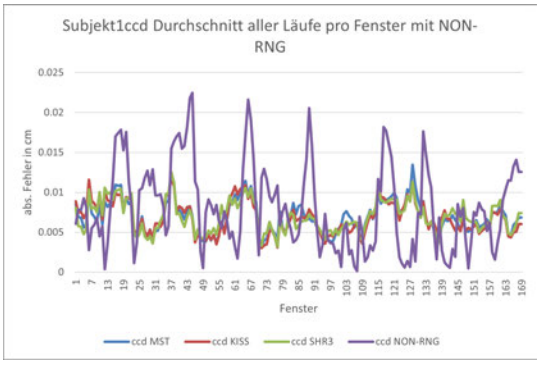
Quellcode 5.1: Implementierung des NON-RNG

Beide Methoden „randint“ und „uniform“, siehe Quellcode 5.1, generieren Zahlen mittels des Counters. Dabei generiert die Methode „randint“ einen Integer und „uniform“ einen Float.

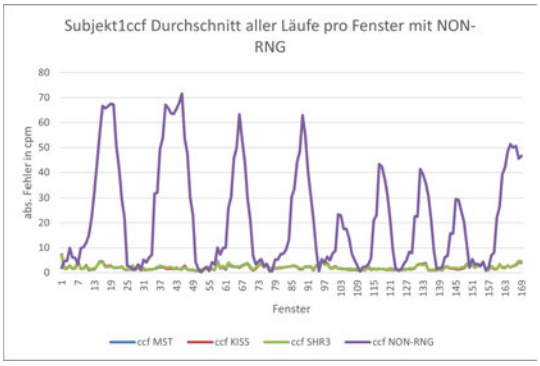
5.3.1 Ergebnisse

Die Ergebnisse, dargestellt in den Abbildungen 5.1a, 5.2a, 5.3a, 5.4a, 5.5a und 5.6a deuten darauf hin, dass der Algorithmus, der den NON-RNG verwendet, bei einigen Fenstern ähnliche Verhaltensmuster aufweist wie andere PRNGs im Kontext der CCD. Es können aber auch Unterschiede beobachtet werden, denn bei manchen Fenster, verhält sich der Algorithmus besser aber auch schlechter. Bei der CCF ist ein signifikanter Unterschied erkennbar, siehe Abbildung 5.1b, 5.2b, 5.3b, 5.4b, 5.5b und 5.6b. Hier hat sich die Performance des Algorithmus verschlechtert, da die Werte der absoluten Fehler größer sind, was darauf hindeutet, dass die Vorhersage nicht nah am tatsächlichen Wert liegt.

Subjekt 1



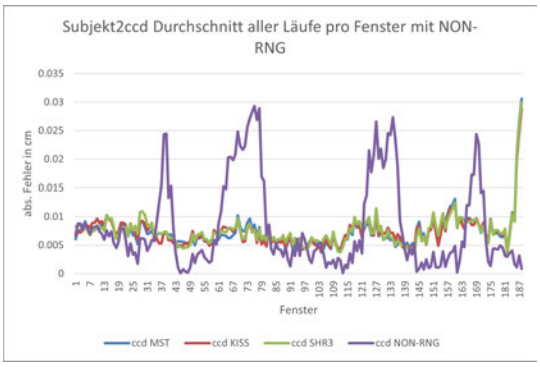
(a) CCD von Subjekt 1



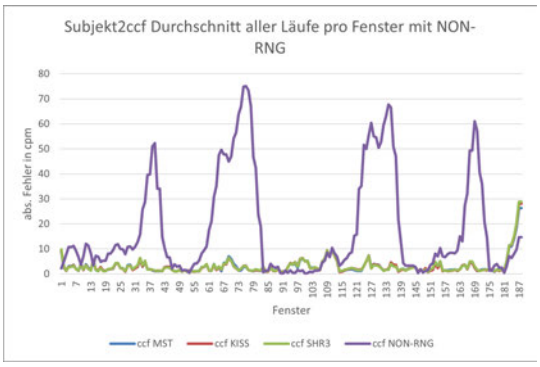
(b) CCF von Subjekt 1

Abbildung 5.1: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt1 mit NON-RNG

Subjekt 2



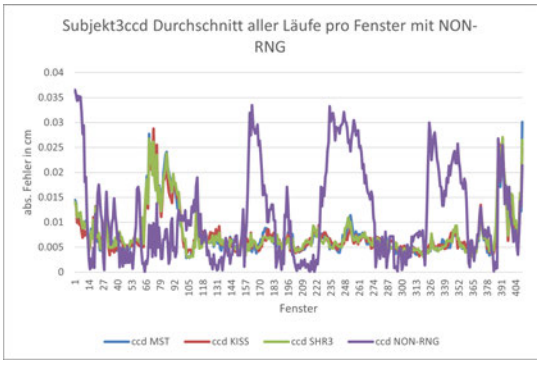
(a) CCD von Subjekt 2



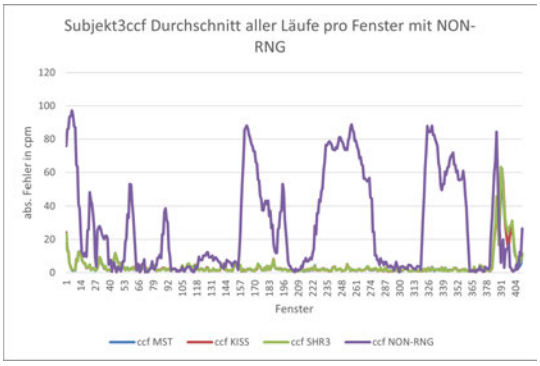
(b) CCF von Subjekt 2

Abbildung 5.2: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt2 mit NON-RNG

Subjekt 3



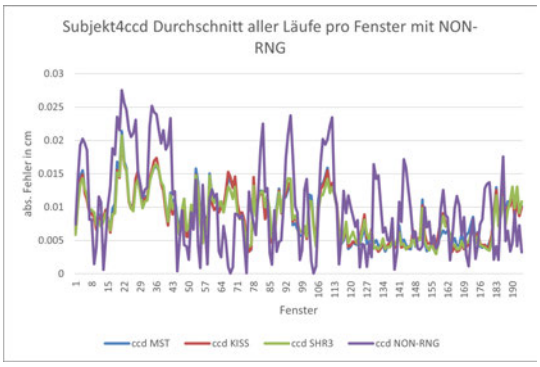
(a) CCD von Subjekt 3



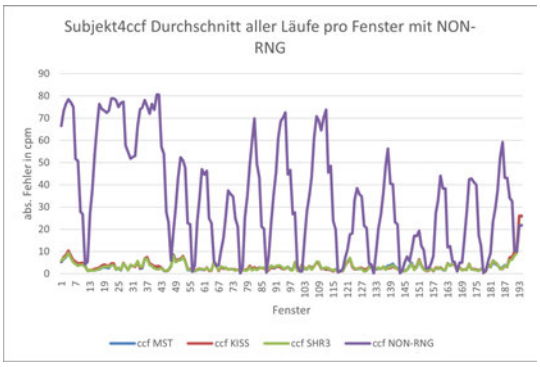
(b) CCF von Subjekt 3

Abbildung 5.3: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt3 mit NON-RNG

Subjekt 4



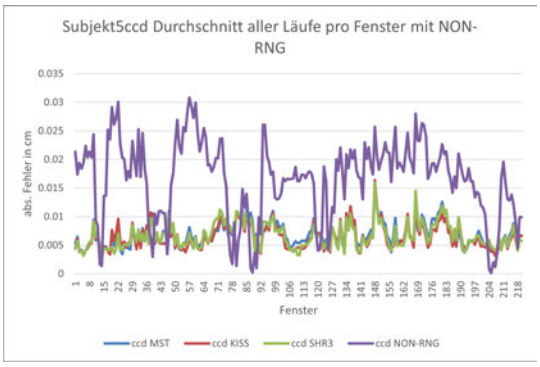
(a) CCD von Subjekt 4



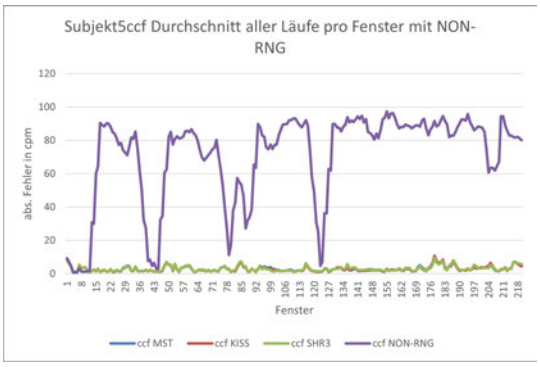
(b) CCF von Subjekt 4

Abbildung 5.4: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt4 mit NON-RNG

Subjekt 5



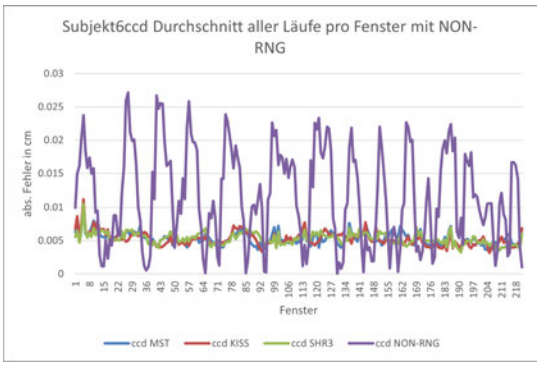
(a) CCD von Subjekt 5



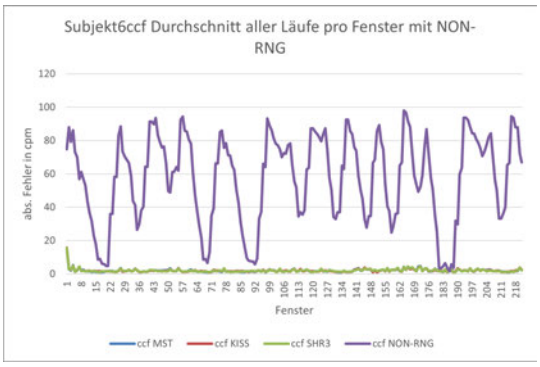
(b) CCF von Subjekt 5

Abbildung 5.5: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt5 mit NON-RNG

Subjekt 6



(a) CCD von Subjekt 6



(b) CCF von Subjekt 6

Abbildung 5.6: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt6 mit NON-RNG

p-Werte

Die in Tabelle 5.1 aufgeführten p-Werte deuten darauf hin, dass bei allen Subjekten (CCD und CCF) mit Ausnahme von Subjekt 1 (CCD) und Subjekt 4 (CCD) ein signifikanter Unterschied in den durchschnittlichen absoluten Fehlern zwischen den verschiedenen PRNGs besteht, da die p-Werte unter dem Signifikanzniveau sind.

Subjekt	p-Wert CCD	p-Wert CCF
1	0.53	$<$ 0,01
2	$<$ 0.01	$<$ 0,01
3	$<$ 0,01	$<$ 0,01
4	0.61	$<$ 0,01
5	$<$ 0,01	$<$ 0.01
6	$<$ 0,01	$<$ 0,01

Tabelle 5.1: p-Werte der PRNGs für Subjekt 1 bis 6 für die durchschnittlichen absoluten Fehler mit NON-RNG

5.3.2 Feststellung

Der Extremfall verdeutlicht, dass die Verwendung einer simplen und vorhersehbaren Sequenz zu schlechteren und inkonsistenteren Ergebnissen führt. Daher lässt sich nicht behaupten, dass jede wahllose Zahlensequenz für diesen Anwendungsfall ausreichend ist. Es wäre interessant zu untersuchen, wie zufällig die Zahlensequenzen sein müssten, um ähnliche oder sogar bessere Ergebnisse im Vergleich zu den anderen PRNGs zu erzielen. Allerdings ist das nicht das Hauptziel dieser Arbeit, weswegen diese Fragestellung hier nicht weiterverfolgt wird.

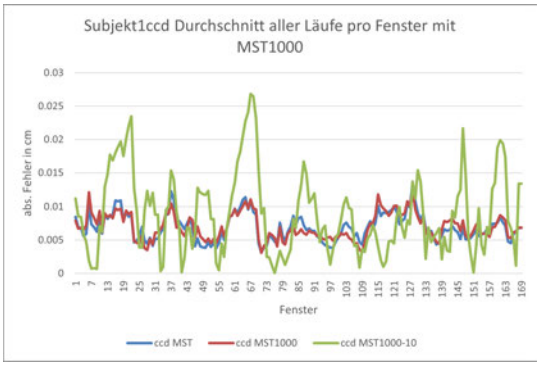
5.4 Extremfall 2

Wie im Kapitel 2.5 erwähnt wurde, hat Cantú-Paz in seiner Arbeit [4] den Mersenne Twister in verschiedenen Konfigurationen getestet. Dabei wurde der Mersenne Twister PRNG auf 1000 Zahlen begrenzt und mittels zufälligen und einem festen Seed (10) getestet. Um zu prüfen, ob die Erkenntnisse aus dieser verwandten Arbeit auf diesen spezifischen Anwendungsfall übertragbar sind, wird ein ähnlicher Test durchgeführt. Dies ermöglicht eine detaillierte Analyse, ob die in der vorherigen Studie gewonnenen Erkenntnisse auch für diese Untersuchung zutreffend sind. Da der MST1000-10 PRNG eine konstante Sequenz an Zahlen hat, wird dieser einmal für jedes Subjekt durchlaufen und wird als ein qualitativ schlechter PRNG eingestuft, während der MST1000 50 Mal durch jedes Subjekt durchlaufen wird, da dieser unterschiedliche Seeds verwendet.

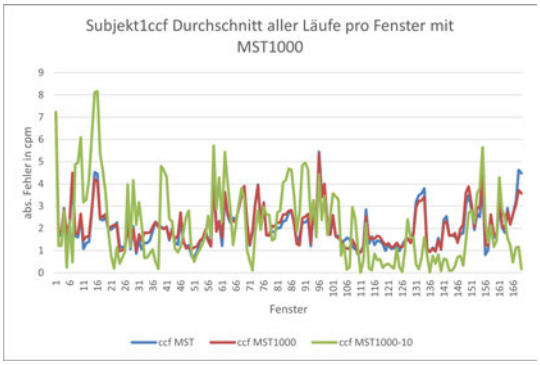
5.4.1 Ergebnisse

Die grafischen Darstellungen in den Abbildungen 5.7 und 5.8 zeigen die absoluten Fehler für die CCD und die CCF der PRNGs MST, MST1000 und MST1000-10 für Subjekt 1 und 2. Es lassen sich Unterschiede in den absoluten Fehlern zwischen den PRNGs feststellen. Der PRNG MST1000-10 zeigt in einigen Fenstern verbesserte Ergebnisse, weist jedoch sowohl für CCD als auch CCF im Vergleich zu MST und MST1000 auch schlechtere Ergebnisse auf. Zwischen MST und MST1000 sind lediglich geringfügige Unterschiede erkennbar.

Subjekt 1



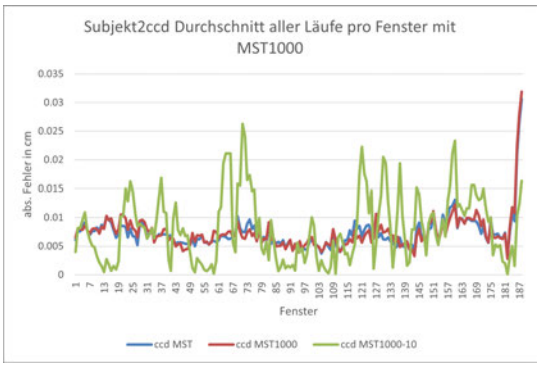
(a) CCD von Subjekt 1



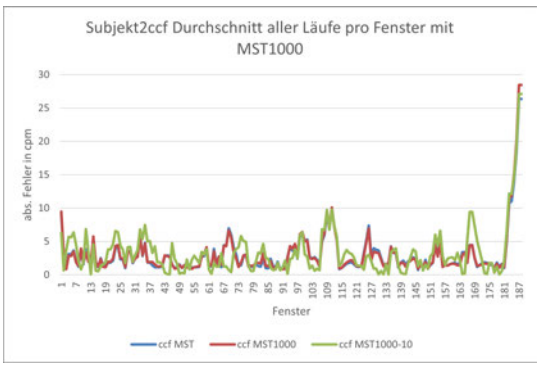
(b) CCF von Subjekt 1

Abbildung 5.7: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt1 mit MST1000

Subjekt 2



(a) CCD von Subjekt 2



(b) CCF von Subjekt 2

Abbildung 5.8: Durchschnitt des absoluten Fehlers von der CCD & CCF für Subjekt2 mit MST1000

p-Werte

Die p-Werte in Tabelle 5.2 verdeutlichen, dass keine signifikanten Unterschiede zwischen den Ergebnisse der PRNGs, aufgezeichnet werden kann, da die p-Werte über dem Signifikanzniveau liegen.

Subjekt	p-Wert CCD	p-Wert CCF
1	0.08	0,16
2	0,92	0,95

Tabelle 5.2: p-Werte der PRNGs für Subjekt 1 bis 2 für die durchschnittlichen absoluten Fehler mit MST1000

5.4.2 Feststellung

Ähnlichkeiten mit den Ergebnissen von Cantú-Paz [4] lassen sich feststellen. Diese zeigten, dass qualitativ schlechtere PRNGs bessere Ergebnisse erzielen können. Diese Beobachtung konnte auch für diesen Anwendungsfall bestätigt werden. Der qualitativ schlechtere PRNG MST1000-10 erzielt bei einigen Fenstern bessere Ergebnisse. Zudem waren die Ergebnisse von MST und MST1000 bei Cantú-Paz [4] ähnlich, was auch hier beobachtet

werden kann. Die zuvor genannten grafischen Darstellungen verdeutlichen einen geringen Unterschied zwischen den beiden PRNGs.

6 Fazit

6.0.1 Zusammenfassung

In dieser Arbeit wurde mittels des Differential Evolution Algorithmus untersucht, ob es einen signifikanten Unterschied gibt, wenn verschiedene PRNGs verwendet werden. Dazu wurde ein anwendungsspezifischer Fall ausgewählt, um zu analysieren, ob die theoretischen Erkenntnisse in der Praxis bestätigt werden können. Die Ergebnisse zeigen, dass die Wahl des PRNGs in manchen Anwendungsfällen nur einen geringen Einfluss auf die Resultate hat. Obwohl die Art und Weise, wie die Zufallszahlen der PRNGs erzeugt werden, unterschiedlich ist, führt dies nicht notwendigerweise zu unterschiedlichen Ergebnissen.

Die ergänzenden Tests in Kapitel 5 haben verdeutlicht, dass die Anwendung suboptimaler Sequenzen oder eines einzigen Seeds zu inkonsistenten Ergebnissen führen kann. Es wurde festgestellt, dass in einigen Fällen suboptimale Sequenzen bessere Lösungen erzielen können als Sequenzen aus hochwertigen PRNGs, jedoch sollte dies als Ausnahme betrachtet werden. Daher wird von der Verwendung simpler Zahlensequenzen abgeraten, stattdessen wird empfohlen, hochwertige PRNGs auszuwählen und die Seed-Auswahl sorgfältig zu treffen, um konsistente und gute Ergebnisse zu erzielen.

6.0.2 Reflexion

Betrachtet man die Durchführung dieser Arbeit, so kann festgestellt werden, dass die Anzahl an Durchläufen hätte höher sein können, um Ausreißer aus den Endergebnissen zu unterdrücken und damit genauere Ergebnisse zu erzielen. Es wäre zudem vorteilhafter gewesen, die PRNGs zunächst an Benchmark-Funktionen zu testen. Auf diese Weise hätte man mehr als einen Anwendungsfall haben können, um zu analysieren, wie sich die Ergebnisse des Differential Evolution Algorithmus theoretisch durch die Anwendung der PRNGs unterscheiden.

6.0.3 Ausblick

Die Erkenntnisse dieser Untersuchung sind ausschließlich auf den spezifischen Anwendungsfall sowie die entsprechenden Parameter und PRNGs beschränkt. Daher ist es von entscheidender Bedeutung, weitere Forschung in diesem Bereich zu betreiben. Als Ausblick könnte festgehalten werden, dass weitere Tests mit denselben PRNGs durchgeführt werden können, um zu überprüfen, ob die Ergebnisse auch in anderen Anwendungsfällen reproduzierbar sind. Zusätzlich bietet es sich an, weitere Tests unter Verwendung unterschiedlicher Parameter durchzuführen, um die Robustheit des Algorithmus zu testen. Es wäre auch interessant zu analysieren, wie robust die PRNGs auf die Wahl der Seeds sind und wie sich dies auf den Algorithmus auswirkt. Zudem kann untersucht werden, wie zufällig die Zahlensequenzen sein müssten, um den Algorithmus dazu zu bringen, ähnliche oder bessere Ergebnisse zu erzielen wie die der PRNGs MST, KISS und SHR3. Dafür könnte der NON-RNG wiederverwendet werden. Diese Empfehlungen könnten als Grundlage für künftige Forschungsarbeiten dienen, die sich mit dem Thema „Einfluss von RNGs auf die Effizienz von evolutionären Algorithmen“ befassen.

Literaturverzeichnis

[1] : *CSTI Creative Space for Technical Innovations*. – URL https://csti.haw-hamburg.de/. – Zugriffsdatum: 2023-11-20

[2] : *Putty*. – URL https://www.putty.org/. – Zugriffsdatum: 2023-11-20

[3] : *WinSCP :: Official Site :: Download*. – URL https://winscp.net/eng/index.php. – Zugriffsdatum: 2023-11-20

[4] Cantú-Paz, Erick: *On random numbers and the performance of genetic algorithms*. 10 2002. – URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3125461

[5] Cárdenas-Montes, Miguel ; Vega-Rodríguez, Miguel A. ; Gómez-Iglesias, Antonio: Sensitiveness of Evolutionary Algorithms to the Random Number Generator. In: Dobnikar, Andrej (Hrsg.) ; Lotrič, Uroš (Hrsg.) ; Šter, Branko (Hrsg.): *Adaptive and Natural Computing Algorithms*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, S. 371–380. – URL https://link.springer.com/chapter/10.1007/978-3-642-20282-7_38. – ISBN 978-3-642-20282-7

[6] Crocetti, Luca ; Nannipieri, Pietro ; Di Matteo, Stefano ; Fanucci, Luca ; Saponara, Sergio: Review of Methodologies and Metrics for Assessing the Quality of Random Number Generators. In: *Electronics* 12 (2023), Nr. 3. – URL https://www.mdpi.com/2079-9292/12/3/723. – ISSN 2079-9292

[7] Dasgupta, Dipankar ; Michalewicz, Zbigniew: *Evolutionary Algorithms — An Overview*. S. 3–28. In: Dasgupta, Dipankar (Hrsg.) ; Michalewicz, Zbigniew (Hrsg.): *Evolutionary Algorithms in Engineering Applications*. Berlin, Heidelberg : Springer Berlin Heidelberg, 1997. – URL https://doi.org/10.1007/978-3-662-03423-1_1. – ISBN 978-3-662-03423-1

[8] Eiben, A. E. ; Smith, James E.: *Introduction to Evolutionary Computing*. 2nd. Springer Publishing Company, Incorporated, 2015. – URL https://doi.org/10.1007/978-3-662-44874-8. – ISBN 3662448734

[9] Feoktistov, Vitaliy: *Differential Evolution – In Search of Solutions*. Bd. 5. Springer, 01 2006. – ISBN 978-0-387-36895-5

[10] Gerdes, I. ; Klawonn, F. ; Kruse, R.: *Evolutionäre Algorithmen: Genetische Algorithmen — Strategien und Optimierungsverfahren — Beispielanwendungen*. Vieweg+Teubner Verlag, 2013 (Computational Intelligence). – URL https://books.google.de/books?id=TBd6rWMLCqcC. – ISBN 9783322868398

[11] Karunasingha, Dulakshi Santhusitha K.: Root mean square error or mean absolute error? Use their ratio as well. In: *Information Sciences* 585 (2022), S. 609–629. – URL https://www.sciencedirect.com/science/article/pii/S0020025521011567. – ISSN 0020-0255

[12] Katzgraber, Helmut G.: Random numbers in scientific computing: An introduction. In: *arXiv preprint arXiv:1005.4117* (2010). – URL https://arxiv.org/abs/1005.4117

[13] Lins, Christian ; Friedrich, Björn ; Hein, Andreas ; Fudickar, Sebastian: An evolutionary approach to continuously estimate CPR quality parameters from a wrist-worn inertial sensor. In: *Health and Technology* 12 (2022), Jan, Nr. 1, S. 161–173. – URL https://doi.org/10.1007/s12553-021-00618-7. – ISSN 2190-7196

[14] Marsaglia, George: Xorshift RNGs. In: *Journal of Statistical Software* 8 (2003), Nr. 14, S. 1–6. – URL https://www.jstatsoft.org/index.php/jss/article/view/v008i14

[15] Mart, Rafael ; Pardalos, Panos M. ; Resende, Mauricio G. C.: *Handbook of Heuristics*. 1st. Springer Publishing Company, Incorporated, 2018. – ISBN 3319071254

[16] Matsumoto, Makoto ; Nishimura, Takuji: Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. In: *ACM Trans. Model. Comput. Simul.* 8 (1998), jan, Nr. 1, S. 3–30. – URL https://doi.org/10.1145/272991.272995. – ISSN 1049-3301

[17] McQueen, Craig: *SimpleRandom*. – URL https://pypi.org/project/simplerandom/. – Zugriffsdatum: 2023-09-19

[18] NUMPY.ORG: *NUMPY Documentation — NUMPY V1.26 Manual*. – URL https://numpy.org/doc/stable/index.html. – Zugriffsdatum: 2023-03-10

[19] OPENVPN: *Business VPN for secure networking | OpenVPN*. – URL https://openvpn.net/. – Zugriffsdatum: 2023-11-20

[20] PREL, Jean-Baptist d. ; RÖHRIG, Bernd ; HOMMEL, Gerhard ; BLETTNER, Maria: Auswahl statistischer Testverfahren. In: *Deutsches Ärzteblatt International* 107 (2010), Nr. 19, S. 343–348. – URL https://www.aerzteblatt.de/int/article.asp?id=74880. – Zugriffsdatum: 2023-11-21

[21] PYTHON.ORG: *Random — Generate pseudo-random numbers*. – URL https://docs.python.org/3/library/random.html. – Zugriffsdatum: 2023-09-19

[22] RAJASHEKHARAN, Lekshmi ; SHUNMUGA VELAYUTHAM, C.: Is Differential Evolution Sensitive to Pseudo Random Number Generator Quality? – An Investigation. In: BERRETTI, Stefano (Hrsg.) ; THAMPI, Sabu M. (Hrsg.) ; SRIVASTAVA, Praveen R. (Hrsg.): *Intelligent Systems Technologies and Applications*. Cham : Springer International Publishing, 2016, S. 305–313. – URL https://link.springer.com/chapter/10.1007/978-3-319-23036-8_26. – ISBN 978-3-319-23036-8

[23] ROSE, Greg: *KISS: A Bit Too Simple*. Cryptology ePrint Archive, Paper 2011/007. 2011. – URL https://eprint.iacr.org/2011/007. – https://eprint.iacr.org/2011/007

[24] SAMANTHA, Lomuscio: *Getting started with the Kruskal-Wallis Test | UVA Library*. 2021. – URL https://library.virginia.edu/data/articles/getting-started-with-the-kruskal-wallis-test. – Zugriffsdatum: 2023-11-21

[25] SCIPY.ORG: *SciPy*. – URL https://scipy.org/. – Zugriffsdatum: 2023-09-25

[26] SKANDEROVA, Lenka ; ŘEHOŘ, Adam: Comparison of Pseudorandom Numbers Generators and Chaotic Numbers Generators used in Differential Evolution. In: ZELINKA, Ivan (Hrsg.) ; SUGANTHAN, Ponnuthurai N. (Hrsg.) ; CHEN, Guanrong (Hrsg.) ; SNASEL, Vaclav (Hrsg.) ; ABRAHAM, Ajith (Hrsg.) ; RÖSSLER, Otto (Hrsg.): *Nostradamus 2014: Prediction, Modeling and Analysis of Complex Systems*. Cham : Springer International Publishing, 2014, S. 111–121. – ISBN 978-3-319-07401-6

[27] STATISTIK-NACHHILFE: *Kruskal-Wallis-Test / H-Test - Statistik Wiki Ratgeber Lexikon*. – URL https://www.statistik-

nachhilfe.de/ratgeber/statistik/induktive-statistik/signifikanztests-hypothesentests/verteilungsunabhaengige-tests-nichtparametrische-tests/kruskal-wallis-test-h-test. – Zugriffsdatum: 2023-11-21

[28] Tharwat, Alaa ; Schenck, Wolfram: Population Initialization techniques for evolutionary algorithms for single-objective constrained optimization problems: Deterministic vs. stochastic techniques. In: *Swarm and Evolutionary Computation* 67 (2021), 12, S. 100952. – URL https://doi.org/10.1016/j.swevo.2021.100952. – Zugriffsdatum: 2023-10-29

[29] Tharwat, Alaa ; Schenck, Wolfram: Population initialization techniques for evolutionary algorithms for single-objective constrained optimization problems: Deterministic vs. stochastic techniques. In: *Swarm and Evolutionary Computation* 67 (2021), S. 100952. – URL https://www.sciencedirect.com/science/article/pii/S2210650221001140. – ISSN 2210-6502

[30] Tirronen, Ville ; Äyrämö, Sami ; Weber, Matthieu: Study on the Effects of Pseudorandom Generation Quality on the Performance of Differential Evolution. In: Dobnikar, Andrej (Hrsg.) ; Lotrič, Uroš (Hrsg.) ; Šter, Branko (Hrsg.): *Adaptive and Natural Computing Algorithms*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, S. 361–370. – ISBN 978-3-642-20282-7

[31] Zürich, Universität: *Kruskal-Wallis-Test*. – URL https://www.methodenberatung.uzh.ch/de/datenanalyse_spss/unterschiede/zentral/kruskal.html. – Zugriffsdatum: 2023-11-21

A Anhang

A.1 Verwendete Hilfsmittel

In der Tabelle A.1 sind die im Rahmen der Bearbeitung des Themas der Bachelorarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tool	Verwendung
ChatGPT	Umformulierung von Sätzen
Excel	Erstellung von Tabellen & Grafiken
TableConvert	Konvertierung von Excel-Tabellen in Latex-Tabellen
Draw.io	Erstellung von Abbildungen

Tabelle A.1: Verwendete Hilfsmittel und Werkzeuge

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht

Ort | Datum | Unterschrift im Original