Universität
Augsburg
University

Fakultät für Informatik
Organic Computing

Vergleich verschiedener Rekombinationsalgorithmen für Cartesian Genetic Programming

Masterarbeit

Cindy Ebertz

Abgabedatum 26. März 2025
Matrikelnummer 1542003
Studiengang Master Ingenieurinformatik

Gutachter Prof. Dr. Jörg Hähner
Prof. Dr.-Ing. Lars Mikelsons

Zusammenfassung

TODO

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Motivation und Aufbau

Das klassische Genetic Programming (GP) wird heutzutage für die Problemlösung in den unterschiedlichsten Domänen erforscht. Beispiele hierfür sind die Erstellung einer mathematischen Gleichung für einen industriellen Prozess [SB01], die Strukturanalyse von FGMs (funktionell gradierten Materialien) [Dem+22] und der Verarbeitung von natürlicher Sprache [Ara20].

Cartesian Genetic Programming (CGP) ist eine Methode des GPs, in der Lösungen für Probleme als Graphen dargestellt werden [Mil20]. Im Standard-CGP werde laut Miller der Rekombinationsschritt normalerweise nicht ausgeführt und in den meisten Arbeiten werde dieser Schritt gänzlich ignoriert. Er bezieht dieses Verhalten auf Forschungsergebnisse aus dem Jahr 1999, die nach Miller aufzeigen, dass der Rekombinationsschritt kaum einen Effekt auf die Effizienz von CPG hat. [Mil20] In mehreren weiteren Papern werden andere Rekombinationsalgorithmen mit dem Ziel vorgestellt, dass der Rekombinationsschritt neben der Mutation sinnvoll in CGP eingebaut werden kann. Dieser weitere Operator könnte die Effizienz von CGP-Modellen im Training steigern und somit komplexere Ausgangsprobleme lösbar machen. Innerhalb dieser Paper wird als Prämisse angenommen, dass wie von Miller geschildert in Standard-CGP keine Rekombination verwendet wird. [CWM07; Kal20; TST22]
Da die Aussage, dass der Rekombinationsschritt nicht zielführend in Standard-CGP sei auf den Papern von Miller aus dem Jahr 1999 und 2011 basiert, kommt die Frage auf, ob dies immer noch zutrifft. Die erste Forschungsfrage, die in dieser Arbeit beantwortet werden soll, ist demnach die folgende: „Kann mit heutigem Forschungsstand nachgewiesen werden, dass Rekombination in Standard-CGP sinnvoll eingesetzt werden kann im Vergleich zu CGP ohne Rekombinationsschritt?“

Da der Erfolg der Rekombination ebenfalls von der Rekombinationsrate abhängt, ist es sinnvoll diese näher zu betrachten. Clegg et al. und Torabi et al. beschreiben in ihren Papern unterschiedliche Herangehensweisen an die Rekombinationsrate [CWM07; TST22]: Clegg et al. führen eine variable Rekombinationsrate in ihrem Paper ein. Dabei wird eine

hohe Rekombinationsrate linear verringert, sodass in den letzten Lernschritten keine Rekombination mehr ausgeführt wird. [CWM07]
Torabi et al. fügen für ihre Rekombination ein Offset zu Beginn des CGP ein. Dieser Offset wird durch einen Hyperparameter bestimmt und gibt an, wie viele Iterationen die Rekombination ausbleibt. [TST22]
Diese beiden Ansätze sollen mit einem selbst entwickelten Ansatz verglichen werden. Die neu vorgestellte Rekombinationsrate bezieht sich auf die One-Fifth-Rule zur Berechnung der Mutationsrate. Die zweite Forschungsfrage, die in dieser Arbeit beantwortet werden soll, ist: „Hat die Art und Weise der Rekombinationsratenberechnung eine Auswirkung auf die Effektivität des CGP?“

Die beiden Forschungsfragen sollen anhand unterschiedlicher Experimente beantwortet werden. In Abschnitt 2 werden die theoretischen Grundlagen gelegt, die für das Verständnis des Experimentaufbaus und der Ergebnisse benötigt werden. Im Anschluss werden in Abschnitt 3 die jeweiligen Experimente, sowie die Evaluationsstrategien der Ergebnisse beschrieben. Darauf folgend werden in Abschnitt 4 die Ergebnisse der Experimente vorgestellt und ausgewertet. Abschließend wird in Abschnitt 5 ein Fazit aus den Ergebnissen zusammengefasst, sowie ein Ausblick auf weitere mögliche Forschungsfragen gegeben.

2 Grundlagen

Für den praktischen Teil dieser Arbeit in Abschnitt 3 werden mit Hilfe von CGP unterschiedliche Testprobleme gelöst. Für das Verständnis dieses Teil werden einige Grundkenntnisse vorausgesetzt, welche in diesem Abschnitt näher betrachtet werden.
CGP ist eine Art von GP, welches verwendet wird, um unterschiedliche Probleme zu lösen. Dabei wird der Maschine allerdings nicht beigebracht, wie diese Probleme zu lösen sind. Stattdessen lernt sie eigenständig über mehrere Iterationen hinweg das Ausgangsproblem zu lösen. Dies geschieht angelehnt an die Darwin'sche Theorie der biologischen Evolution. [Ahv+19]
Um den grundlegenden Aufbau von CGP zu erläutern, wird folgende Abbildung 2.1 eingeführt:

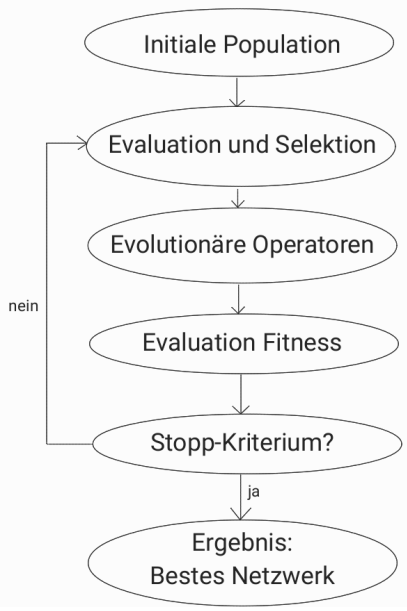
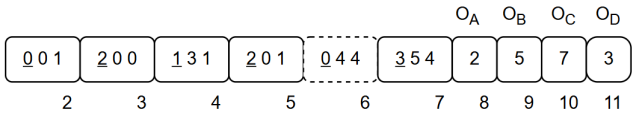


Abbildung 2.1: Aufbau CGP, angelehnt an [TST22]

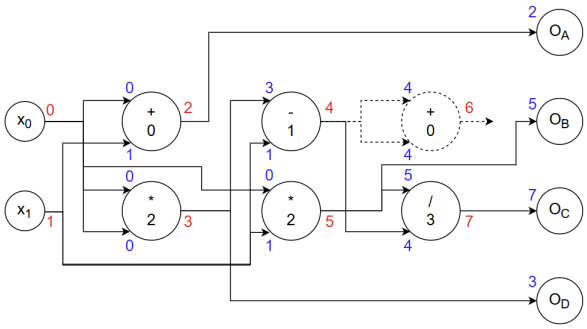
Die Abbildung 2.1 zeigt den grundlegenden Ablauf innerhalb von CGP. Dabei lernt das System über mehrere Iterationen hinweg die beste Lösung eines Problems. Die folgenden Unterkapitel beziehen sich jeweils auf einen Knoten des Graphen und erläutern diesen genauer.

2.1 Initiale Population

Die Initialisierung der Population bezieht sich auf die Paper [Mil20], [TST22] und [Ahv+19]. Außerdem werden Erkenntnisse aus dem Quellcode von Cui verwendet [Cui24b].
Die *Population* von CGP ist eine Menge von *Chromosomen*, auch Individuen genannt. Chromosome sind individuelle Möglichkeiten ein komplexes Ausgangsproblem zu lösen. Jedes Chromosom ist in CGP ein gerichteter, azyklischer Graph, bestehend aus *Eingangsknoten*, *Rechenknoten* und *Ausgangsknoten*. Dabei gibt es zwei Darstellungsmöglichkeiten für ein Chromosom: den *Genotyp* und den daraus resultierenden *Phänotyp*. Anhand der folgenden Abbildung 2.2 kann beispielhaft näher erläutert werden, wie die Chromosome in CGP aufgebaut sind und funktionieren.



(a) Beispiel Genotyp



(b) Beispiel Phänotyp

Abbildung 2.2: Darstellungsmöglichkeiten eines Chromosoms, angelehnt an [TST22]

Abbildung 2.2a zeigt einen Genotyp und Abbildung 2.2b den daraus resultierenden Phänotyp. Der Genotyp ist dabei das volle Individuum, während der Phänotyp die Dekodierung dessen ist. Die Lösung des CGPs ist dementsprechend ein Phänotyp, also die Dekodierung des besten Individuums des CGPs. Der Genotyp- und Phänotyp-Raum können sich dabei stark voneinander unterscheiden. [ES15] Dies lässt sich dadurch erklären, dass nicht alle Teile des Genotyps für die Berechnung des Endergebnisses verwendet werden. Diejenigen Anteile, die für die Berechnung der Lösung nicht gebraucht werden, werden *inaktive Knoten* genannt. So ein inaktiver Knoten wird in Abbildung 2.2a durch gestrichelte Linien dargestellt. Da der Phänotyp die Lösung des CGPs ist, müssen inaktive Knoten auch nicht im Phänotyp dargestellt werden, da diese für die Berechnung keine Verwendung finden. Der Vollständigkeit halber wird der inaktive Knoten aus Abbildung 2.2a auch in Abbildung 2.2b aufgenommen. So können also verschiedene Genotypen zum gleichen Phänotyp und somit zum gleichen Ergebnis des CGPs führen: indem die gleichen aktiven Knoten innerhalb mehrerer Genotypen, allerdings unterschiedliche inaktive Knoten vorhanden sind. Dies kann die Weiterentwicklung der Chromosomen stören, wie in einem Beispiel in Abschnitt 2.4 beschrieben wird.
Anhand des Phänotyps in Abbildung 2.2b lässt sich die klassische Struktur von CGP erkennen: Es handelt sich um einen gerichteten, azyklischen Graphen mit Ein- und Ausgängen. Bei den Eingängen (hier: x_0 und x_1) handelt es sich um die Systemeingänge, aus denen Ausgänge (hier: O_A – O_D) abgeleitet werden sollen. Zwischen den Ein- und Ausgangsknoten liegen die Rechenknoten. Diese werden verwendet, um verschiedene Rechenoperationen an den Eingangsknoten auszuführen, bis schließlich die Inhalte der Ausgangsknoten als Ergebnis resultieren. Die Rechenoperationen, die von den Rechenknoten ausgeführt werden, werden je nach Anwendungsfall definiert und codiert. In diesem Beispiel ergibt sich folgende Kodierung:

Rechenoperation	**Kodierung**
+	0
-	1
*	2
/	3

Tabelle 2.1: Kodierung der Rechenoperationen

Diese Kodierungen werden im Phänotyp innerhalb der Knoten angegeben und stellen die erste (unterstrichene) Zahl innerhalb der Genotyp-Arrays dar. Ein Genotyp-Array wird in Abbildung 2.2a umrandet dargestellt und steht jeweils für einen Knoten. Unterhalb dieser

Array-Blöcke wird jeweils der Index des zugehörigen Knotens angegeben. Im Phänotyp wird dieser Index am Ausgang des jeweiligen Knotens (rot) angezeigt.
Des Weiteren werden im Genotyp die Eingangskanten für jeden Knoten angegeben. In diesem Beispiel hat jeder Rechenknoten zwei Eingänge und jeder Ausgangsknoten hat jeweils nur einen Eingang, in dem das Ergebnis eines Rechenknoten weitergeleitet und ausgegeben wird. Die Knoteneingänge sind in Abbildung 2.2b blau markiert. Hier werden die Indices derjenigen Knoten angegeben, deren Ausgangswerte verwendet werden. Diese spiegeln sich ebenfalls im Genotyp wider: hier werden die Knoteneingänge innerhalb der Array-Blöcke als nicht-unterstrichene Indices angegeben. Damit ergibt sich eine vollständige Beschreibung eines Knotens: einem Index werden Eingänge und gegebenenfalls eine Rechenoperation zugeschrieben.
Durch dieses Beispiel wird ebenfalls ersichtlich, was die Eingangsknoten eines Chromosoms ausmacht: Sie geben die Systemeingänge wieder, ohne diese auf irgendeine Weise zu verarbeiten. Aus diesem Grund müssen die Eingangsknoten nicht im Genotyp aufgezeigt werden, um sie vollständig zu beschreiben, denn sie weisen weder Eingänge noch Rechenoperationen auf, die beschrieben werden müssten.

Ob ein Knoten für die Berechnung der Lösung verwendet wird oder nicht, hängt davon ab, ob ein nachfolgender Knoten auf dessen Ausgang zugreift. Um aktive Knoten von inaktiven Knoten zu unterscheiden, werden demnach zuerst die Ausgangsknoten betrachtet, die offensichtlich für die Auswertung der Ausgabe verwendet werden. Anschließend werden iterativ die Eingangskanten der Rechenknoten zurückverfolgt, bis man schließlich bei den Eingangsknoten des Graphen ankommt.

Schließlich soll anhand der eingeführten Abbildungen 2.2a und 2.2b ein Beispiel berechnet werden:
Angenommen werden die Systemeingänge x_0 = 10 und x_1 = 20. Demnach sind die Ausgänge der beiden Eingangsknoten mit den Indices 0 und 1 gleich den Werten 10 und 20. Der erste Rechenknoten (Index = 2) verwendet als Eingänge die beiden Eingangsknoten (Indices = 0 und 1) und besitzt die Rechenfunktion +. Demnach ist das Ergebnis des Rechenknotens (10 + 20 =) 30. Führt man dieses Vorgehen für die restlichen Rechenknoten aus, ergeben sich folgende Ergebnisse:

Knoten	Eingänge	Werte Eingänge	Rechenoperation	Ausgangswert
2	0; 1	10; 20	10 + 20	30
3	0; 0	10; 10	10 * 10	100
4	3; 1	100; 20	100 - 20	80
5	0; 1	10; 20	10 * 20	200
7	5; 4	200; 80	200 / 80	2,5

Tabelle 2.2: Ergebnisse Rechenknoten

Die Ausgangsknoten geben die jeweiligen Ergebnisse der Eingangs- oder Rechenknoten zurück. In diesem Beispiel werden durch das Chromosom die folgenden Ausgänge aus den beiden Eingängen (10; 20) berechnet:

Ausgangsknoten	Index Eingang	Wert des Ausgangsknotens
O_A	2	30
O_B	5	200
O_C	7	2,5
O_D	3	100

Tabelle 2.3: Ergebnisse Ausgangsknoten

Zusammengefasst hat das Beispielchromosom aus dem Eingangstupel (10; 20) ein Ausgangstupel (30; 200; 2,5; 100) berechnet.

Wie bereits erläutert, ist die Population in CGP eine Menge an Chromosomen, also eine Menge an gerichteten, azyklischen Graphen, die aus definierten Systemeingaben Ausgaben berechnen können. Diese Population wird zu Beginn zufällig initialisiert. Dabei werden folgende Angaben vorausgesetzt:

- Größe der Population
- Anzahl der Systemeingänge
- Anzahl der Systemausgänge
- Anzahl der Rechenknoten pro Chromosom

Im Initialisierungsprozess werden für jedes Chromosom pro Knoten zufällige Eingangskanten und gegebenenfalls Rechenoperationen bestimmt. Dabei muss beachtet werden, dass es sich anschließend um einen azyklischen Graphen handeln muss.

Nachdem die initiale Population erstellt wurde, erfolgt der erste *Evaluations-* und *Selektionsschritt*. Der folgende Abschnitt 2.2 erläutert, wie diese Schritte ausgeführt werden.

2.2 Evaluation und Selektion

Der erste Selektionsschritt in CGP startet mit einer zufällig initialisierten Population, wie sie in Abschnitt 2.1 beschrieben wird. In dieser initialen Population ist die Performance der einzelnen Individuen rein zufällig. Das Ziel von CGP ist es, die Performance über Generationen hinweg zu verbessern, bis schließlich eine (nahezu) perfekte Lösung eines Problems gefunden wird. GP im Allgemeinen richtet sich nach dem darwin'schen Prinzip des Überlebens der Stärkeren. Demnach werden die performantesten Chromosomen verwendet, um die nächste Generation der Population zu erzeugen. [Koz95]

Um zu bestimmen, welche Individuen die beste Performance aufweisen, muss eine numerische Bewertung erfolgen. Diese wird durch die *Fitness* der einzelnen Chromosomen bestimmt. [Koz95] Wie der Fitnesswert berechnet wird, hängt von den zu lösenden Problemen ab. Beispielsweise verwenden Cui et al. für symbolische Regressionsprobleme den mittleren absoluten Fehler zwischen korrekter Lösung und tatsächlicher Lösung für einen Evaluationsdatensatz. [CHH24]

Koza beschreibt in seinem Paper, dass für die Selektion der Eltern der nächsten Generation, jedem Chromosom ein Wahrscheinlichkeitswert zugewiesen wird. Dieser hängt von dessen Fitness ab. Anschließend wird eine definierte Anzahl an Eltern selektiert, wobei die Wahrscheinlichkeitswerte dafür sorgen, dass fittere Individuen die größere Chance haben, selektiert zu werden. Selektion heißt dabei, dass das Chromosom unverändert in die nächste Generation kopiert wird. [Koz95]
Dies ist ein Weg dafür zu sorgen, dass das Prinzip nach Darwin eingehalten wird und somit die fitteren Chromosomen „überleben". Eine andere Möglichkeit, dies zu erreichen, ist die Verwendung von sogenannten *Elitisten*. Die fittesten Individuen einer Generation werden dabei als Elitisten erwählt und werden für die folgende Generation selektiert. [Kra+13] Für die Auswahl der Elitisten wird in dieser Arbeit der *neutral search Algorithmus* verwendet.

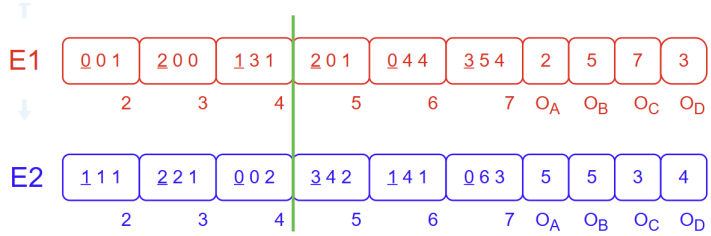
Dieser wird relevant, falls innerhalb einer Generation ein Elter-Chromosom und ein Kind-Chromosom die gleiche Fitness aufweisen. In diesem Fall wird stets das Kind-Chromosom als Elitist ausgewählt. Dies führt dazu, dass anschließend bessere Nachkommen erzeugt werden können. [CMH22]
Mit dem in dieser Arbeit verwendeten *(μ + λ)-Selektionsverfahren* wird dieser Ansatz verfolgt. Dabei werden jeweils μ-viele Eltern für die folgende Generation selektiert. Im Standard-CGP wird μ mit 1 belegt. Die restlichen Individuen der Population (λ-viele) werden anschließend durch *Mutation* aus dem Elter-Chromosom gebildet. [SB18] Da für den *Rekombinationsschritt* jeweils zwei Eltern-Chromosomen gekreuzt werden, muss, falls Rekombination ausgeführt wird, für μ ein Wert größer als 1 gewählt werden. Um eine bessere Vergleichbarkeit zu gewährleisten, werden im praktischen Teil auch unterschiedliche μ-Werte gewählt, selbst wenn keine Rekombination ausgeführt wird.
Wie Mutation und Rekombination ausgeführt werden, wird in folgenden Abschnitten 2.3 und 2.4 erläutert.

2.3 Evolutionärer Operator: Rekombination

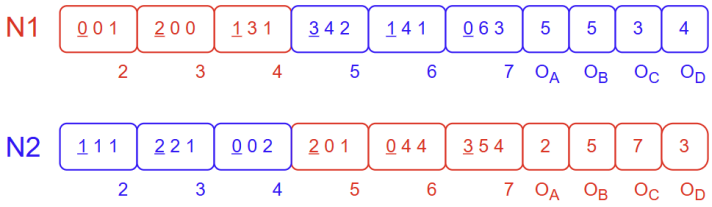
Nachdem die Selektion der Eltern-Chromosome erfolgt ist, wie in Abschnitt 2.2 beschrieben, können im nächsten Schritt die evolutionären Operationen ausgeführt werden, die die Nachkommen aus den Eltern erzeugen.
Der erste evolutionäre Operator, der verwendet wird, ist die Rekombination. Dabei werden jeweils zwei zufällig ausgewählte Eltern-Chromosomen kombiniert und somit zwei neue (Nachkommen-)Chromosomen erzeugt [Kal20]. Dieser Rekombinationsschritt wird so oft ausgeführt, bis die Population dieser Generation gefüllt ist.
Innerhalb dieser Arbeit werden unterschiedliche Standard-Rekombinationsalgorithmen für die Nachwuchserzeugung miteinander verglichen. Die folgenden Absätze erläutern diese näher.

2.3.1 One-Point Rekombination

Das erste Standard-Rekombinationsverfahren, das in dieser Arbeit verwendet wird, ist die *One-Point Rekombination*, auf die im Folgenden näher eingegangen wird. Für die Grundlagen, die in diesem Abschnitt erläutert werden, wurde folgende Quelle verwendet: [PG17]
Der One-Point Rekombinationsalgorithmus soll anhand folgender Beispielabbildung 2.3 erläutert werden:



(a) Eltern-Chromosomen One-Point Rekombination



(b) Nachwuchs-Chromosomen One-Point Rekombination

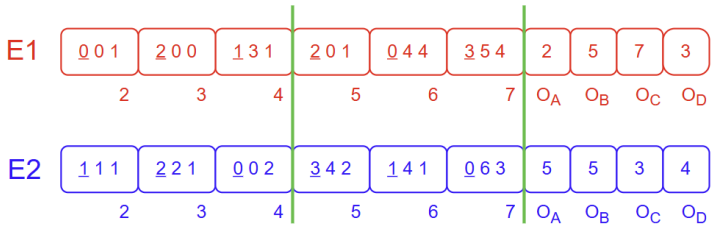
Abbildung 2.3: One-Point Rekombination, angelehnt an [TST22]

Das erste Eltern-Chromosom (E1, rot) in diesem Beispiel wurde aus Abbildung 2.2a entnommen. Das zweite Eltern-Chromosom (E2, blau) ist ein zufällig gewähltes Beispielchromosom.
Bei beiden Chromosomen wird vorerst nicht weiter betrachtet, ob die Knoten aktiv oder inaktiv sind, da sich dieses Merkmal mit der Rekombination und Mutation ändern kann. Angenommen wird, dass C1 und C2 aus der letzten Generation übernommen wurden, da sie die beste Fitness aufgewiesen haben.
Für die One-Point Rekombination wird zuerst eine zufällige Stelle innerhalb der Eltern-Chromosomen bestimmt. Diese ist in Abbildung 2.3a grün markiert. Die Eltern-Chromosomen werden anschließend an dieser Stelle geteilt und überkreuzt zusammengesetzt. Für dieses Beispiel ergeben sich die beiden Nachwuchs-Chromosomen aus Abbildung 2.3b.

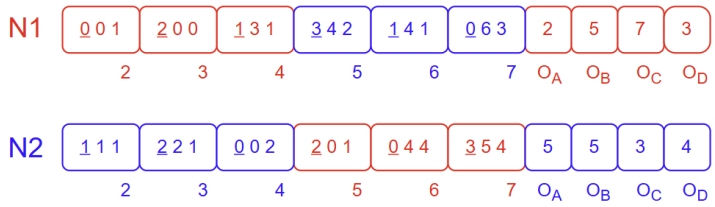
2.3.2 Two-Point Rekombination

Für die Grundlagen dieses Abschnitts wurde [PG17] als Quelle verwendet.
Das Verfahren der *Two-Point Rekombination* funktioniert identisch zur in Abschnitt 2.3.1

erläuterten One-Point Rekombination, mit dem Unterschied, dass zwei zufällige Stellen ausgewählt werden, an denen die Chromosomen geteilt werden.
Anhand des nachfolgenden Beispiels kann dies nachvollzogen werden:



(a) Eltern-Chromosomen Two-Point Rekombination



(b) Nachwuchs-Chromosomen Two-Point Rekombination

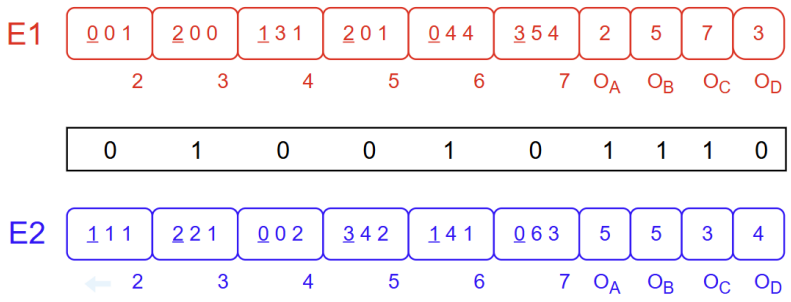
Abbildung 2.4: Two-Point Rekombination, angelehnt an [TST22]

Zu beobachten ist, dass die Eltern-Chromosomen an jeweils zwei Stellen aufgeteilt werden (grün). Die Nachwuchs-Chromosomen bilden sich anschließend abwechselnd aus den Teilstücken der beiden Eltern-Chromosomen. In der Abbildung 2.4 wird dieser Prozess deutlich, indem die Chromosomenteile des ersten Elternteils rot markiert sind und des zweiten blau.

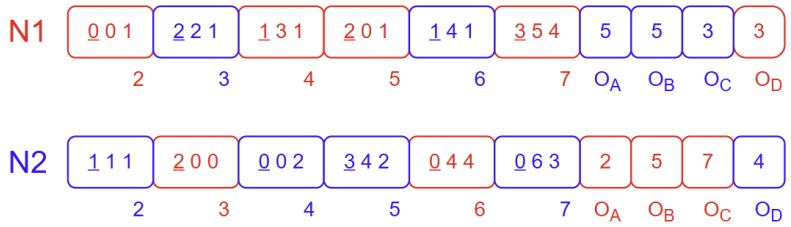
2.3.3 Uniform Rekombination

Die Erläuterung der *Uniform Rekombination* basiert auf [Sys89].
Was die Uniform Rekombination von der One-Point oder Two-Point Rekombination abhebt, ist die Verwendung einer Maske. Die Maske ist genauso lang wie die Eltern-Chromosomen und beinhaltet für jede Stelle binäre Werte. Diese Werte geben jeweils an, von welchem

Elternteil die jeweilige Stelle im Chromosom des Nachwuchses stammen soll. Der zweite gebildete Nachwuchs bekommt in diesem Prozess das Gen des jeweils anderen Elternteils. Anhand der Abbildung 2.5 lässt sich die Uniform Rekombination beispielhaft erklären:



(a) Eltern-Chromosomen Uniform Rekombination



(b) Nachwuchs-Chromosomen Uniform Rekombination

Abbildung 2.5: Uniform Rekombination, angelehnt an [TST22]

Abbildung 2.5a zeigt die Ausgangssituation mit den beiden Eltern-Chromosomen (E1 und E2) in rot und blau. Zwischen den beiden Eltern-Chromosomen wird schwarz die Maske angezeigt. Diese wird zufällig binär gefüllt, bis sie die Länge der beiden Eltern-Chromosomen erreicht.
Im folgenden Schritt werden die Nachwuchs-Chromosomen anhand der Maske erzeugt. Der entsprechende Index des Knotens kann jeweils unterhalb der Eltern-Chromosomen abgelesen werden. Für den ersten Wert der Maske ergibt das den Knotenindex 2. In diesem Beispiel enthält der erste Wert der Maske eine 0. Dementsprechend wird für den Knoten mit dem Index 2 des ersten Nachwuchs-Chromosoms (N1) der jeweilige Knoten des ersten Eltern-Chromosoms (E1, rot) verwendet. Das zweite Nachwuchs-Chromosom (N2)

bekommt demnach den entsprechenden Knoten aus dem zweiten Eltern-Chromosom (E2, blau).
Für den darauffolgenden Index hat die Maske den Wert 1. Dies bedeutet, dass die Vererbungen genau andersherum ablaufen: N1 bekommt den Knoten von E2 und N2 bekommt den Knoten von E1. Dieser Prozess wird für alle Indices ausgeführt. Die Abbildung 2.5b zeigt die resultierenden Ergebnisse für dieses Beispiel der Uniform Rekombination.

2.3.4 Rekombinationsraten

In den letzten Abschnitten wurden unterschiedliche Rekombinationsalgorithmen erläutert. Der Rekombinationsschritt wird allerdings nicht für jedes Nachwuchs-Chromosom verwendet. Dieser wird nur zu einer bestimmten Wahrscheinlichkeit ausgeführt, welche mit der *Rekombinationsrate* beschrieben wird. Die Effektivität des (C)GP-Systems hängt unter anderem von der richtigen Wahl der Rekombinationsrate ab. [Has+19]
Um das bestmögliche (C)GP-System zu erhalten, sollte das richtige Verhältnis aus *Exploration* (dt. Erforschung) und *Exploitation* (dt. Ausbeutung) des Lösungsraums erzielt werden. Exploration ist dabei der Prozess, neue Bereiche des Lösungsraums zu erkunden, während bei der Exploitation bereits erkundete Regionen des Lösungsraums näher betrachtet werden. Eine Stellschraube, um den eigenen Prozess dahingehend zu steuern, ist die Rekombinationsrate. [ČLM13] Wird die Rekombinationsrate sehr hoch eingestellt, wird zu einem hohen Maß Exploration betrieben. Dies führt allerdings dazu, dass die Exploitation niedrig gehalten wird und somit die optimalen Lösungen verfehlt werden. [PG17]
In unterschiedlichen Papern werden verschiedene Herangehensweisen vorgeschlagen, diesen Parameter zu wählen. Teilweise widersprechen sich diese Aussagen. Ziel dieser Arbeit ist es, unter anderem einen Einblick zu bekommen, wie man die Rekombinationsrate richtig wählen kann und welche Auswirkungen sie auf die Güte von CGP-Lösungen hat.

Die folgenden Abschnitte geben einen Einblick über die unterschiedlichen Möglichkeiten, die Rekombinationsrate zu belegen.

Konstante Rekombinationsrate

Wie von Hassanat et al. beschrieben, ist die konstante (statische) Rekombinationsrate die übliche Form. Als geläufiges Beispiel wird in ihrem Paper der Wert 0,9 für die Rekombinationsrate vergeben. [Has+19] Dies bedeutet, dass zur Initialisierung des CGP ein fester

Wert für die Rekombinationsrate gewählt wird. Dieser gilt für alle Generationen gleichermaßen und wird nicht verändert.
Diese „klassische“ Form der Rekombination kann in der Evaluation der praktischen Tests dazu verwendet werden, um die erste Forschungsfrage zu beantworten. Durch diese einfachste Form der Rekombinationsrate kann überprüft werden, ob CGPs ohne oder mit (klassischer) Rekombination effizienter sind.
Mit Hilfe der in den nächsten Abschnitten folgenden Anpassungen der Rekombinationsrate kann anschließend überprüft werden, ob die Effizienz von CGP mit Rekombination weiter verbessert werden kann.

Linear fallende Rekombinationsrate

Clegg et al. präsentieren in ihrem Paper aus 2007 eine neue Form der Rekombination. Dabei treffen sie auch einige Aussagen über die Rekombinationsrate, die in diesem Abschnitt näher betrachtet werden sollen.[CWM07]
Sie beobachten, dass sich für höhere Rekombinationsraten eine schnellere Konvergenz der Fitness innerhalb der ersten Generationen einstellt. Gleichzeitig stellen sie fest, dass in ihrem Beispiel ab der 200. Generation Rekombination keinen signifikanten Vorteil mehr in der Performance liefert.
Aus diesen beiden Beobachtungen konstruieren sie eine neue, dynamische Rekombinationsrate. Diese beginnt bei einem hohen Startwert von 0,9 und sinkt linear, bis eine Rekombinationsrate von 0,0 erreicht wird. In ihrem Beispiel legen sie über händische Analysen eine Generation fest, bis zu welcher die Rekombinationsrate auf 0,0 fallen soll.
Innerhalb des praktischen Teils dieser Arbeit wird 0,9 für den Startwert der Rekombinationsrate übernommen, um die Anzahl der Parameter in der Hyperparameteranalyse zu reduzieren. Ebenfalls ist es ein Vorteil, nur einen variablen Parameter pro Rekombinationsraten-Typ zu verwenden, da gegebenenfalls die Änderungen innerhalb der Ergebnisse für die verschiedenen Parameter besser miteinander verglichen werden können.
Der einzige variable Parameter für die linear fallende Rekombinationsrate ist in dieser Arbeit die Rate, die nach jeder Generation von der alten Rekombinationsrate abgezogen werden soll, um die neue Rekombinationsrate zu erhalten.

One-Fifth-Regel angewandt auf die Rekombinationsrate

Die *One-Fifth-Regel* gibt es bereits andere Parameter von GP, wie beispielsweise für die *Mutationsrate*, die in Abschnitt 2.4 näher erläutert wird. Diese Regel wird ebenfalls im Paper

von Milano und Nolfi verwendet. Dabei handelt es sich um einen Weg, die Mutationsrate automatisch und dynamisch an die Problemcharakteristiken und die evolutionäre Phase anzupassen. [MN18]
Betrachtet wird bei dieser Regel das Fitness-Verhältnis der Elitisten und der neuen Chromosomen. In anderen Worten werden die Kinder also mit ihren Eltern verglichen. Erzielt werden soll ein Verhältnis von 20%. Das heißt, dass 20% der Nachwuchs-Chromosomen eine bessere Fitness aufweisen sollen als ihre Eltern. Wird dieses Verhältnis unter- oder übertroffen, wird der jeweilige Parameter verkleinert oder vergrößert. [DDL19]

In dieser Arbeit soll die One-Fifth-Regel für die dynamische Anpassung der Rekombinationsrate herangezogen werden. Um nur den Rekombinationsschritt zu bewerten und nicht den Mutationsschritt, muss die Bewertung der Fitness vor der Mutation geschehen. Dementsprechend wird für die One-Fifth-Regel in dieser Arbeit direkt nach dem Rekombinationsschritt die Fitness der Eltern- mit der Fitness der Nachwuchs-Chromosomen verglichen. Anschließend wird betrachtet, ob 20% der Kinder eine bessere Fitness aufweisen als ihre Eltern. Wird dieser Wert übertroffen, wird die Rekombinationsrate mit 1,1 multipliziert, um den Erfolg des Rekombinationsschritts auszuschöpfen. Andernfalls wird die Rekombinationsrate mit 0,9 multipliziert und somit verringert.

Rekombinationsrate mit Offset

Torabi et el. beschreiben in ihrem Paper eine neue Rekombinationsstrategie [TST22]. Dabei verwenden sie einen Hyperparameter, der den *Offset der Rekombination* definieren soll. Das heißt, dass die Rekombination in ihrer Strategie in den ersten Generationen nicht angewendet wird, sondern erst zu einer bestimmten Generation beginnt. Wie dieser Hyperparameter bestimmt wurde und welche Größenordnung dieser einhalten sollte, wird in dem Paper allerdings nicht erwähnt.

Torabi et al. behaupten außerdem, dass ihre Strategie „den richtigen Kompromiss aus Exploration und Exploitation“ erreiche. [TST22] Vergleicht man diese Aussage mit der Aussage von Clegg et al., stellt man fest, dass sich diese Aussagen widersprechen [CWM07]: Während Clegg et al. vor allem in den ersten Generationen auf Rekombination setzen, da der Rekombinationsschritt hier zu einer höheren Fitness-Konvergenz führen soll, meiden Torabi et al. in ihrer Strategie Rekombinationen in den ersten Generationen völlig.
Ein Ziel dieser Arbeit ist es herauszufinden, ob sich ein Offset in der Rekombination als

sinnvoll erweist oder ob sich dieser nur für die Rekombinationsstrategie eignet, die Torabi et al. in ihrem Paper eingeführt haben.

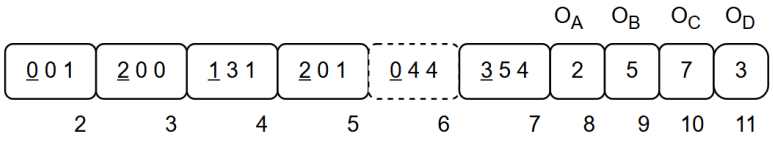
2.4 Evolutionärer Operator: Mutation

Der zweite evolutionäre Operator ist die Mutation. Sie wird nach der Rekombination ausgeführt. Anders als bei der Rekombination werden bei der Mutation nicht zwei, sondern nur ein Chromosom einbezogen und daraus ein neues Chromosom erstellt. In diesem Prozess werden Teile des Genotyps des Chromosoms zufällig verändert, um daraus einen neuen Genotyp zu erzeugen. [Ahv+19]
Für *probabilistische Mutation* wird, vergleichbar zur Rekombinationsrate, eine Mutationsrate verwendet, um die Wahrscheinlichkeit anzugeben, mit der ein Gen mutiert wird. Dadurch kann es allerdings dazu kommen, dass das Chromosom vor und nach der Mutation zwar unterschiedliche Genotypen aufweist, sich die aktiven Knoten allerdings nicht voneinander unterscheiden. Dies hat zur Folge, dass sich der Lösungsansatz des CGPs bezüglich des Ausgangsproblems nicht ändert, obwohl bereits eine Mutation ausgeführt wurde. Die Fitness des CGPs kann sich in so einem Fall demnach nicht verbessern. Um dieses Problem in den Griff zu bekommen, wird in dieser Arbeit die *Single (Active) Mutation* herangezogen. In diesem Algorithmus werden zufällige Gene eines Chromosoms verändert, bis ein aktiver Knoten mutiert wurde. Anschließend bricht der Mutationsalgorithmus ab. [Mil20] Dies hat zusätzlich den Vorteil, dass keine Mutationsrate angepasst werden muss, was die Hyperparameteranalyse weniger rechenintensiv macht.

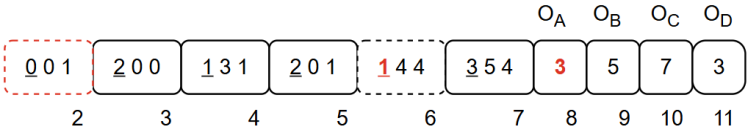
Die Veränderung eines Gens innerhalb der Mutation wird vorgenommen, indem das ausgewählte Zeichen des Genotyps zufällig geändert wird [Koz95]. Dies hat verschiedene Effekte für unterschiedliche Knotenarten des entsprechenden Phänotyps.
Ein Ausgangsknoten hat als relevanten Parameter nur seinen Vorgängerknoten. Wird also ein Ausgangsknoten mutiert, wird die eingehende Kante zufällig neu belegt. Dabei muss wie bei der Initialisierung beachtet werden, dass die Struktur von CGP erhalten bleibt. In dieser Arbeit werden dementsprechend nur Vorgängerknoten gewählt, deren Index kleiner ist als der mutierte Knoten.
Anders als bei Ausgangsknoten wird ein Rechenknoten durch mindestens zwei Parameter bestimmt: ein Rechenoperator und mindestens eine Eingangskante. Welcher dieser Parameter mutiert werden soll, ist ebenfalls Zufall. Wird eine Kante verändert, gelten die gleichen Regeln wie beim Mutieren eines Ausgangsknotens. Wird der Rechenoperator mutiert, wird dieser zufällig neu aus den kodierten Rechenfunktionen gewählt.

Da ein Eingangsknoten nur aus den Dateneingängen besteht und keine Parameter enthält, können diese nicht mutiert werden und werden im Mutationsschritt nicht betrachtet.

Um die Single Active Mutation anhand eines Beispiels näher zu erläutern wird folgender Gentyp eines Chromosoms aus Abbildung 2.2a erneut eingeführt. Es wird angenommen, dass dieses Chromosom durch vorherige Rekombination entstanden ist und somit im nächsten Schritt mutiert werden soll. Die folgende Abbildung 2.6 beschreibt dieses Mutationsbeispiel:



(a) Genotyp vor Mutation



(b) Genotyp nach Mutation

Abbildung 2.6: Single Active Mutation, angelehnt an [TST22]

Der Algorithmus beginnt und ein zufälliger Index wird für die Mutation ausgewählt. Angenommen dieser Index hat den Wert 6. Dabei handelt es sich um einen Rechenknoten. Da in diesem Beispiel die Rechenknoten jeweils 3 Parameter aufweisen, wird zufällig einer aus drei Parametern für die Mutation bestimmt. In diesem Beispiel wird der erste Parameter verändert, also der Rechenoperator. Wie bereits in Tabelle 2.1 aufgezeigt, werden die Rechenoperationen durch 0 bis 3 kodiert. Sei beispielsweise die zufällig gewählte Zahl für den Rechenoperator gleich 1. Da der in diesem Schritt mutierte Knoten inaktiv ist, muss nach der Single Active Mutation erneut mutiert werden.
Angenommen der zufällig gewählte Index ist 8, was dem Index von O_A entspricht. Da es sich, um einen Ausgangsknoten handelt, muss ein neuer Vorgängerknoten zufällig gewählt werden. Beispielsweise wird nun dieser Parameter mit dem Index 3 belegt. Da dieser Knoten Teil der aktiven Knoten ist, terminiert hier der Mutationsalgorithmus. Die Abbildung 2.6b zeigt das Ergebnis des Mutationsbeispiels (mit Kennzeichnung der neuen aktiven / in-

aktiven Knoten). Die mutierten Teile des Chromosoms, sowie der neu entstandene inaktive Knoten werden rot hervorgehoben.

2.5 Evaluation Fitness, Stopp-Kriterium und Ergebnis

Nachdem der Mutationsschritt für alle Chromosome der Population ausgeführt wurde, findet erneut ein Evaluationsschritt statt. Dabei wird für jedes Chromosom ein Fitness-Wert bestimmt, der bewertet wie exakt die Trainingsdaten durch das Chromosom beschrieben werden. [Ahv+19] Die Berechnung der Fitness hängt wie in Abschnitt 2.2 geschrieben vom zu lösenden Ausgangsproblem ab.
Sobald für alle Chromosome einer Population ein Fitness-Wert bestimmt wurde, werden diese auf das *Stopp-Kriterium* geprüft. Dieses gibt an, ab welcher Bedingung ein Algorithmus als konvergiert gilt. Im Beispiel von Cui et al. ist diese Bedingung bei symbolischen Regressionsbenchmarks für $Fitness < 0,01$ erfüllt. [CHH24]
Wird das Stopp-Kriterium erfüllt, wird der CGP-Algorithmus abgebrochen. Das Ergebnis ist das beste Chromosom der letzten Generation. Dieses löst das Ausgangsproblem hinreichend gut. Andernfalls wird eine weitere Iteration des CGP-Algorithmus gestartet. Dabei werden erneut Selektion, Rekombination und Mutation ausgeführt, um eine bessere Lösung des Ausgangsproblems zu finden.

2.6 Bayes'sche Analyse

Für die Evaluation der Ergebnisse werden in diesem Abschnitt statistische Grundlagen zur *Bayes'schen Analyse* erläutert.
Der Satz von Bayes (Bayes'sches Therorem) bietet Grundlagen zur Berechnung der bedingten Wahrscheinlichkeit. Zu Beginn wird eine Anfangshypothese über das Ergebnis eines Wahrscheinlichkeitsproblems angenommen. Mit zusätzlichen Informationen wird die Anfangshypothese schließlich korrigiert. Das Ergebnis ist eine neue Wahrscheinlichkeit, die alle vorhandenen Informationen einschließt. [Pey20]

Vereinfacht dargestellt funktioniert die Anwendung des Satzes von Bayes in der Bayes'schen Analyse wie in folgender Abbildung 2.7.

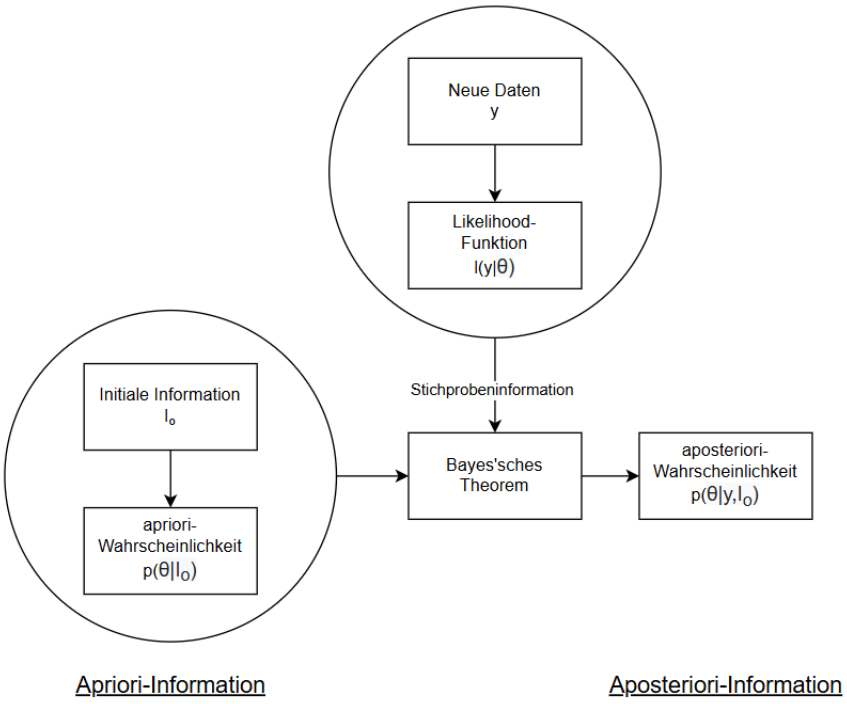


Abbildung 2.7: Bayes'sche Analyse, angelehnt an [Nen80]

Zu Beginn wird die Realität als Modell mit bestimmten Annahmen (θ) vereinfacht. Diese ursprünglichen Annahmen werden als Wahrscheinlichkeitsverteilung (*apriori-Wahrscheinlichkeit*) dargestellt. Aus den neuen Daten y kann die sogenannte *Likelihood-Funktion* errechnet werden, die eine Dichtefunktion für die Beobachtung y darstellt unter der Bedingung, dass θ zutrifft. Mit Hilfe des Bayes'schen Theorems können die apriori-Dichtefunktion und die Stichprobeninformationen miteinander verknüpft werden. Das Ergebnis ist eine neue Wahrscheinlichkeitsverteilung für θ (*aposteriori-Wahrscheinlichkeit*). [Nen80] Mit diesem Vorgehen kann also mit Hilfe von empirischen Daten eine Wahrscheinlichkeitsverteilung für ein Modell gefittet werden.

In dieser Arbeit werden basierend auf [CMH23] zwei Modelle verwendet: das *Plackett-Luce-Modell* und das *Gammaverteilung-basierte Modell*.
Mit Hilfe des Plackett-Luce-Modells können die Wahrscheinlichkeiten berechnet werden, mit denen die jeweiligen CGP-Konfigurationen besser als alle anderen Konfigurationen

sind. Demnach kann eine Reihenfolge bestimmt werden, die die Effizienz der Konfigurationen sortiert angibt. Das verwendete Modell basiert auf dem Plackett-Luce-Modell, das von Calvo et al. eingeführt wird. [CMH23; CCL18]
Für die Auswertung mit Hilfe des Gammaverteilung-basierten Modells wird der Code von Pätzel verwendet ([Pät24]). Dieser ermöglicht es nicht-negative Daten zu vergleichen und daraus die Wahrscheinlichkeitsverteilung von μ_{config} zu schätzen. μ_{config} ist dabei eine zufällige Variable, die dem jeweiligen Mittelwert der Iterationenzahl entspricht, die gebraucht werden, um eine CGP-Lösung konvergieren zu lassen. Für jede CPG-Konfiguration kann so ein 95% HPDI (*highest posterior density interval*) von μ_{config} bestimmt werden. Dies bedeutet, dass innerhalb dieses Intervalls 95% der Ergebnisse liegen. [CMH23] Dadurch können die Streuungen der Iterationszahlen für jede CGP-Konfiguration miteinander verglichen werden, ohne von einer gleichverteilten Dichtefunktion ausgehen zu müssen.

Pätzel verwendet in seinem Code das *Markov Chain Monte-Carlo (MCMC) Sampling* zur Berechnung der Verteilungen [Pät24; CMH23]. Bei MCMC werden mehrere Tausend Kombinationen von Parameterwerten (μ_1, σ_1, μ_2, σ_2, v) generiert. Jede Wertekombination ist repräsentativ für glaubwürdige Parameterwerte, die gleichzeitig die eingehenden Daten und die vorherigen Verteilungen berücksichtigen. Diese Parameterwerte werden anschließend in zusammengefasst. [Kru12]

3 Experimente und Evaluation

3.1 Aufbau der Experimente

Für die Beantwortung der Forschungsfragen werden unterschiedliche Experimente ausgeführt, deren Ergebnisse in dieser Arbeit ausgewertet und interpretiert werden sollen. Der Programmcode für die Experimente wurde in der Sprache Julia verfasst. Dabei fand eine Orientierung an folgendem Code von Henning Cui statt: [Cui24b]
Um Ergebnisse von Ausgangsproblemen aus unterschiedlichen Domänen für eine umfangreichere Bewertung zur Verfügung zu stellen, werden mehrere Benchmark-Testszenarien überprüft. Diese werden in den folgenden Abschnitten näher betrachtet.

3.1.1 Testszenarien

Boolesche Probleme

Nach der Aussage von Kalkreuth et al. spielen *Boolesche Probleme* eine wichtige Rolle in der Forschung zu GP. Grundsätzlich ist bei Booleschen Problemen das Ziel einen sinnvollen Zusammenhang zwischen Ein- und Ausgaben zu generieren, welcher von Booleschen Funktionen bestimmt wird. Diese können wiederum durch Boolesche Ausdrücke mathematisch beschrieben werden. Die verschiedenen Booleschen Funktionen können durch Wahrheitstabellen dargestellt werden, in denen die jeweiligen Ein- und Ausgaben miteinander verknüpft werden. [Kal+23] Das Ziel von CGP-Algorithmen ist es aus Eingängen die richtigen Ausgänge zu generieren, welche dem Mapping der Boolean Funktionen entsprechen.

Insgesamt werden in dieser Arbeit vier Boolesche Benchmarkprobleme für die Evaluation der CGP-Algorithmen betrachtet: 3-bit Parity, 16-4-bit Encode, 4-16-bit Decode und 3-bit Multiply (vereinfacht bezeichnet als Parity, Encode, Decode und Multiply).
Obwohl Parity als zu leichtes Ausgangsproblem für GP bezeichnet wird [Whi+13], wird

es häufig als Benchmarkproblem genutzt [YM01; KK17; KK20]. Um Benchmarkprobleme mit unterschiedlichen Ein- und Ausgangsgrößen miteinzubeziehen, werden Encode und Decode verwendet. Sinnvoll ist es ebenfalls Testprobleme verschiedener Schwierigkeitsstufen zu bewerten. Multiply ist dabei ein vergleichsweise schwer zu lösendes Testproblem und wird deswegen herangezogen [WM08].

Das verwendete Standardfunktionsset aller vier Testszenarien beinhaltet die Boolesche Rechenoperatoren AND, OR, NAND und NOR. Außerdem wird die Standardfitnessfunktion für Boolesche Benchmarkprobleme verwendet. Diese wird definiert durch den Anteil an korrekt zugeordneten Bits. [CMH23] Das Stopp-Kriterium ist erfüllt, wenn die Fitness den Wert 0 erreicht. Folgend werden die vier verwendeten Boolesche Benchmarkprobleme näher erläutert:

Parity: N-bit Parity ist eine Mapping-Funktion, die angibt, ob die Summe der Komponenten eines Binär-Vektors gerade oder ungerade ist. Bei 3-bit Parity handelt es sich dabei um Binär-Vektoren der Länge 3. Das Evaluationsset besteht aus $2^N = 2^3$ Testvektoren. [HLS99] Demnach gibt es für das CGP drei Eingaben (die Komponenten eines Binär-Vektors) und eine Ausgabe (Binärwert für „gerade“ / „ungerade“).

Encode: Beim 16-4-bit Encoding wird aus einer 16-stelligen One-Hot-Kodierung ein 4-bit Integer erstellt. Die One-Hot-Kodierung besteht dabei aus einem 16-stelligen Binär-Vektor, wobei nur eine Stelle mit einer 1 belegt ist, die restlichen Stellen sind 0. Ziel ist es diejenige Stelle zu finden, die die 1 hält und diese als üblichen 4-bit Integer zu kodieren. [Cui+23; GP15] Daraus ergibt sich eine Eingabegröße von 16 und eine Ausgabegröße von vier. Der Testdatensatz enthält 16 verschiedene One-Hot-Kodierungen, die umgewandelt werden sollen.

Decode: 4-16-bit Decode hat das genau umgekehrte Ziel als 16-4-bit Encode. Es wird ein 4-bit Integer-Wert angegeben, der durch eine 16-bit One-Hot-Kodierung dargestellt werden soll. [Cui+23] Demnach werden für 4-16-bit Decode vier Eingaben in 16 Ausgaben umgewandelt. Der Testdatensatz besteht aus 16 verschiedenen 4-bit Integerwerten.

Multiply: Ziel von 3-bit Multiply ist die Multiplikation von zwei 3-bit Integer-Werten. Das Ergebnis wird durch einen 6-bit Integer-Wert dargestellt. [Cui+23] Folglich ist die Anzahl der CGP-Eingaben gleich 6, da beide 3-bit Faktoren als Eingang in das CGP-Modell einfließen müssen. Für die Ausgabe werden ebenfalls 6 Binärausgaben benötigt. Der Testdatensatz besteht aus $2^6 = 64$ verschiedenen Kombinationen der möglichen Binär-Faktoren.

Symbolische Regression

Symbolische Regression (SR) zählt seit Beginn von GP als Grundlage für methodologische Forschung und als primäres Anwendungsgebiet [OLM18]. Das Ziel von SR ist das Erlernen einer Beziehung zwischen Ein- und Ausgängen nur aufgrund von gegebenen Daten. Diese Beziehung beruht auf interpretierbaren mathematischen Ausdrücken. Der Fehler von errechnetem und vorgegebenem Ausgang pro Eingang soll dabei minimiert werden. [MC24]

Die in dieser Arbeit verwendeten SR Probleme werden aus dem Paper von Cui et al. übernommen: Keijzer-6, Koza-3, Nguyen-7. Im folgenden werden diese durch Keijzer, Koza und Nguyen abgekürzt. Sie sind von der GP-Community empfohlen und wurden bereits in früheren Arbeiten verwendet [Whi+13; Kal20].
Der verwendete Funktionssatz besteht aus den folgenden acht mathematischen Funktionen: Addition, Subtraktion, Multiplikation, Division, Sinus, Cosinus, natürlicher Logarithmus und Exponentialfunktion. Bei der Division wird sichergestellt, dass eine Division durch null abgefangen wird. [CHH24] Dabei wird der Wert 1,0 ausgegeben, statt eine Division durch null auszuführen. Zur Absicherung des natürlichen Logarithmus werden für alle Eingaben nur die absoluten Werte in die Berechnung einbezogen. Für den Fall, dass die Eingabe gleich 0 ist, wird vergleichbar zur Division der Wert 1,0 zurückgegeben. Da es bei der Programmierung mit Julia zu Fehlern kommen kann, wenn die Eingaben von Sinus oder Cosinus zu groß sind, werden diese Fälle ebenfalls abgefangen. Dabei werden die Eingaben nicht weiter verrechnet sondern durchgereicht.
Für die Berechnung der Fitness wird der mittlere absolute Fehler zwischen vorhergesagter und tatsächlicher Ausgabe pro Eingabe berechnet. Das Stopp-Kriterium ist erfüllt, sobald die Fitness den Wert 0,01 unterschreitet. [CHH24]
Die folgende Tabelle 3.1 beschreibt die verwendeten SR Probleme näher.

Name	Variablen	Gleichung	Trainingsdaten	Testdaten
Keijzer	1	$\sum_{i}^{x} \frac{1}{i}$	E[1, 50, 1]	E[1, 120, 1]
Koza	1	$x^6 – 2 \cdot x^4 + x^2$	U[–1, 1, 20]	-
Nguyen	1	$ln(x+1) + ln(x^2+1)$	U[0, 2, 20]	-

Tabelle 3.1: Beschreibung SR Benchmarkprobleme nach [CHH24]

Zu beobachten ist, dass in unterschiedlichen Papern verschiedene Keijzer-6 Funktionen beschrieben werden [Oli+18; Li+24; Kom18]. In dieser Arbeit wurden alle SR Benchmarkprobleme auf das Paper von Cui et al. bezogen [CHH24].

3.1.2 CGP-Konfigurationen

In dieser Arbeit werden unterschiedliche *CGP-Konfigurationen* miteinander verglichen und evaluiert. Diese enthalten verschiedene Parametrierungen innerhalb eines CGPs und werden in diesem Abschnitt näher erläutert.
Ein Ziel der Arbeit ist es die Effizienz von Rekombination in CGP zu bewerten. Ebenfalls sollen unterschiedliche Rekombinationsalgorithmen und -konfigurationen miteinander verglichen werden, um eine Aussage über deren Effektivität zu treffen. Aus diesem Grund müssen für die unterschiedlichen Testszenarien mehrere Rekombinationskonfigurationen getestet werden. Um eine Vergleichbarkeit der Ergebnisse zu gewährleisten müssen Selektion und Mutation in allen Experimenten gleichen Algorithmen entsprechen. Diese werden im Folgenden beschrieben.
Selektion: Für die Selektion wird in allen Experimenten das (μ+λ)-Selektionsverfahren verwendet. Dabei wurde für die CGP-Konfiguration ohne Rekombinationsschritt nicht $\mu = 1$ festgelegt, obwohl dies den Standard-Einstellungen eines CGP ohne Rekombination entspricht. Dieser zusätzliche Freiheitsgrad soll einen ausgewogeneren Vergleich zwischen den CGP-Konfigurationen mit und ohne Rekombinationsschritt ermöglichen. Für die Auswahl der Elitisten wird das neutral search Verfahren herangezogen.
Mutation: In allen Experimenten dieser Arbeit wird die Single Active Mutation angewendet.

Die nachfolgende Tabelle 3.2 gibt alle Konfigurationen für die **Rekombination** an, die für jedes Testszenario aus Abschnitt 3.1.1 getestet werden.

Rekombinationsverfahren	**Art der Rekombinationsrate**	**Offset**
One-Point Rekombination	konstante Rekombinationsrate	aktiv
Two-Point Rekombination	linear fallende Rekombinationsrate	inaktiv
Uniform Rekombination	Rekombinationsrate mit One-Fifth Regel	-
keine Rekombination	-	-

Tabelle 3.2: Konfigurationen Rekombination

Zu beachten ist, dass diese einzelnen Einstellungen miteinander kombiniert werden, solange es die Verfahren zulassen. Falls keine Rekombination ausgeführt wird, wird selbstverständlich auch die Rekombinationsrate nicht angepasst und es kann kein Offset eingeführt werden. Grundsätzlich werden die Konfigurationskombinationen nach folgendem Muster erstellt:

- Auswahl Rekombinationsverfahren
- Auswahl Art der Rekombinationsrate
- Auswahl Offset aktiv / inaktiv

Dementsprechend ergeben sich 19 verschiedene Konfigurationen für den Rekombinationsschritt, die miteinander verglichen werden sollen.

3.1.3 Hyperparameteroptimierung

Da CGP mehrere Hyperparameter ausweist, die die Effektivität eines CGP-Modells beeinflussen, muss eine *Hyperparameteroptimierung* ausgeführt werden, die einen optimierten Parametersatz ausgibt. Für jede in Abschnitt 3.1.2 eingeführte CGP-Konfiguration wird eine eigene Hyperparameteroptimierung ausgeführt, da nicht von einem optimierten Datensatz auf den nächsten geschlussfolgert werden kann. Außerdem müssen die Hyperparameter auf das jeweilige Testszenario angepasst werden.
Für die Hyperparameteroptimierungen in dieser Arbeit wird das Julia-Paket HyperOpt.jl verwendet [Car25]. Mit Hilfe des Pakets können die Hyperparameter, sowie deren Wertebereiche angegeben werden, sodass eine automatisierte Optimierung stattfinden kann. Eben diese werden in folgender Tabelle 3.3 aufgelistet.

Parameter	**min**	**max**	**Schrittweite**
Anzahl Rechenknoten	50	2000	50
μ (Anzahl Elitisten)	2	20	2
λ (Anzahl Nachkommen)	10	60	2
Offset Rekombination (in Iterationen)	0	P:55 / K:300	P:5 / K:30
Konstante Rekombinationsrate	0,1	1,0	0,1
Fallende Rekombinationsrate (Abzug)	0,005	0,05	0,005
One-Fifth Regel Rekombinationsrate (Startwert)	0,3	0,05	0,75

Tabelle 3.3: Optimierte Hyperparameter und deren Wertebereiche

Zu beachten ist, dass $\mu <= \lambda$ gilt und die Hyperparameteranalysen nur für Parity und Keijzer (in Tabelle 3.3 durch P und K markiert) ausgeführt werden. Eine ausführliche Begründung liefert Abschnitt 3.1.4. Außerdem werden die Einträge der Tabelle 3.3 je nach CGP-Konfiguration verwendet. Zum Beispiel wird der Wert „Offset Rekombination“ nur verwendet, wenn der Rekombinationsoffset aktiv ist. Dieser Wert wird in Iterationen angegeben, in denen die Rekombination ausgesetzt wird. Um die obere Grenze des Offsets sinnvoll abzuschätzen wurden zuerst die Hyperparameteranalysen derjeniger CGP-Konfigurationen vorgenommen, die keinen Offset verwenden. Für jeweils ein Testszenario wurden anschließend die Mittelwerte der benötigten Iterationen berechnet. Auf Basis dieser Mittelwerte konnten die Offset Wertebereiche sinnvoller gewählt werden, sodass sich der Wert der wahrscheinlich benötigten Iterationen mit der oberen Grenze deckt. Die drei letzten Zeilen der Tabelle 3.3 geben die verschiedenen Arten an Rekombinationsraten an, die in dieser Arbeit verglichen werden. Für jede dieser Ratenarten wird gezielt nur ein Hyperparameter verwendet, um die Rechenzeit der Hyperparameteroptimierung zu reduzieren. Für die linear fallende Rekombinationsrate gibt dieser Parameter an, mit welcher Schrittweite die Rekombinationsrate pro Iteration sinkt. In der One-Fifth Regel wird der Startwert der Rekombinationsrate angegeben, also diejenige Rekombinationsrate, mit der das CGP-Modell initialisiert wird.

In einer Optimierungsschleife werden pro Parametersatz 10 Testdurchläufe durchgeführt, in denen das CGP trainiert wird. Für Boolesche Probleme wird die Effizienz die Parametersätze anhand der Iterationen gemessen, die der CGP-Algorithmus braucht, bis er konvergiert. Bei symbolischer Regression bezieht sich der Vergleich auf die berechnete Fitness. Es wurde eine Iterationsgrenze eingeführt, ab der der CGP-Algorithmus in der Hyperparameteroptimierung abbricht, um Rechenzeit zu sparen. Diese Grenze wurde durch vorhergehende Tests so gesetzt, dass nur wenige Ausreißer diese überschreiten. Trifft dies zu wird bei Booleschen Problemen die Anzahl an benötigten Iterationen verdoppelt, um so das gescheiterte Training härter zu bestrafen.
Nach 150 getesteten Parametersätzen bricht die Hyperparameteroptimierung ab und gibt den besten Parametersatz aus.

Das verwendete Julia-Paket bietet verschiedene Sampler an. Der als Standardeinstellung zur Verfügung gestellte Sampler ist ein Random-Sampler. Da der verwendete BHOB Sampler mit und ohne Hyperband keine Einsparungen in der Rechenzeit ergeben haben, wurde weiterhin der einfach zu konfigurierende Random-Sampler verwendet.

3.1.4 Teststruktur

Ursprünglich sollte für jedes Testproblem/Testszenario und für jede CGP-Konfiguration eine eigene Hyperparameteroptimierung durchgeführt werden. Mit den daraus resultierenden Parametersätzen hätten für jede Kombination aus Testszenario und CGP-Konfiguration eine annähernd optimale Lösung gefunden werden können. Diese hätten anschließend systematisch evaluiert und verglichen werden können. Angesichts der begrenzten Rechenkapazitäten konnten nur für die beiden leichtesten Testszenarien Parity und Keijzer sinnvolle Hyperparameteranalysen ausgeführt werden. Für die jeweiligen anderen Szenarien wurde außerdem versucht eine stark reduzierte Hyperparameteranalyse auszuführen, indem alle Parameter, die nichts mit der Rekombination zu tun haben aus den Ergebnissen von Cui et al. herangezogen werden [Cui24a]. So sollten nur die für die Rekombination relevanten Parameter optimiert werden, also die Rekombinationsrate und gegebenenfalls der Offset. Bei den Versuchen dieses Testverfahren für die umfangreicheren Testszenarien auszuführen, wurde ebenfalls festgestellt, dass die verfügbare Rechenkapazitäten nicht ausreichen. Diese Beobachtung wurde auch gemacht als der Offset-Parameter bei der Optimierung vollständig herausgenommen wurde.
Aus diesen Gründen wurde eine neue Methodik entwickelt, die in diesem Abschnitt näher erläutert werden soll. Die Teststruktur wird in zwei voneinander unabhängige Testblöcke geteilt.

1. Testblock: einfache Testszenarien

Für die einfachen Testszenarien Parity und Keijzer kann eine Hyperparameteranalyse trotz eingeschränkter Rechenzeit ausgeführt werden, indem die Iterationenzahl bis zum Abbruch des CGP-Algorithmus heruntergesetzt werden. Dieser wird schrittweise reduziert, bis die Hyperparameteranalyse ausgeführt werden kann.
Um zu Überprüfen, ob die angepassten Iterationsgrenzen für Parity sinnvoll sind, werden die Ergebnisse von Cui et al. herangezogen. Dabei wurden nur die Ergebnisse mit $(\mu + \lambda)$-Selektion verwendet, die mit den CGP-Konfigurationen dieser Arbeit entsprechen [Cui24a]. Es ergibt sich ein HPDI-Maximalwert von ca. 495 Iterationen. Mit einer Iterationsgrenze von 500 ist es wahrscheinlich, dass vor allem schlechte Parametersätze zu einer Überschreitung dieses Werts führen. Durch die erhöhte Bestrafung dieser Überschreitung, wird sichergestellt, dass die sichere Konvergenz des CGP-Algorithmus bis zur Iterationsgrenze priorisiert wird.
Da für Keijzer die Hyperparameteranalyse auf Basis der Fitness ausgewertet wird, können

Güte-Aussagen ebenfalls getroffen werden, auch wenn der CGP-Algorithmus nicht vollständig konvergiert ist, da bereits vor Erreichen des Stopp-Kriteriums eine Aussage über das Konvergenzverhalten getroffen werden kann. Demnach wird für Keijzer die unter den Rechenzeit-Umständen höchste Iterationsgrenze von 500 Iterationen als genügend für die Hyperparameteroptimierung angesehen.

Mit Hilfe der optimierten Parametersätze können anschließend jeweils 50 CGP-Modelle pro Testszenario und CGP-Konfiguration ausgeführt werden. Die Ergebnisse dieser Modelle können verwendet werden, um die Güte der CGP-Konfigurationen auf einfache Testszenarien zu evaluieren. Das Evaluierungsverfahren wird in Abschnitt 3.2 näher erläutert.

2. Testblock: komplexe Testszenarien

Da durch die erhöhte Rechenzeit der komplexeren Testszenarien keine sinnvolle Hyperparameteroptimierung ausführen lässt, wird für diese Fälle eine andere Teststrategie entwickelt.
Diejenigen Hyperparameter eines CGP-Modells, die nicht mit Rekombination in Verbindung stehen, werden aus den Ergebnissen von Cui et al. herangezogen [Cui24a]. Die Rekombinationsrate wird variiert und mit diesen Parametern in CGP-Modelle eingepflegt. Für jede CGP-Konfiguration werden 50 Testdurchläufe ausgeführt, um eine statistische Auswertung ausführen zu können. Dabei werden die CGP-Modelle gegebenenfalls nicht bis zur vollständigen Konvergenz trainiert, um die Rechenzeit zu reduzieren. Für Boolesche Probleme werden den CGP-Modelle dabei weniger Iterationen Trainingszeit zur Verfügung gestellt als für SR Probleme, da diese in der Hyperparameteranalyse von Cui et al. mehr Rechenknoten brauchen und somit für jeweils eine Iteration mehr Rechenzeit benötigen als bei SR. [Cui24a]
Da sich die Wirkung von Rekombinationsrate und Offset gegenseitig beeinflussen kann, ist es sinnvoller die Bewertung der beiden Parameter einzeln zu betrachten. Aufgrund der hohen Anzahl der zu bewertenden Testergebnisse, die sich bereits für einen variierenden Parameter ergeben, soll nur einer dieser Parameter in dieser Arbeit näher betrachtet werden. Dieser Parameter ist wie bereits beschrieben die Rekombinationsrate. Dies ergibt sich daraus, dass die Ergebnisse der Hyperparameteranalyse der einfachen Tests bereits vermuten lassen, dass der Offset keinen deutlichen Mehrwert für das CGP-Training mit sich bringt. Details zu den jeweiligen Ergebnissen können in den Abschnitten 4.1.1 und 4.2.1 nachgelesen werden. Außerdem können mit Hilfe der Rekombinationsrate nähere Erkenntnisse zu

den unterschiedlichen Rekombinationsarten gewonnen werden, die in dieser Arbeit verglichen werden.
Für die Evaluation stehen nach dem Testdurchlauf die Ergebnisse bis zur 10000. Iteration mehrerer CGP-Konfigurationen zur Verfügung, die allerdings nicht die Ergebnisse der CGPs mit den jeweils besten Parametersätzen darstellen, da keine Hyperparameteroptimierung ausgeführt wurde. Trotzdem können die Ergebnisse verwendet werden, um einen näheren Einblick zum Verhalten des CGP-Modells zu erhalten, wenn die Rekombinationsrate variiert wird.
Nähere Informationen zur Evaluierung werden in Abschnitt 3.2 gegeben.

3.2 Evaluation

Für die Auswertung der Ergebnisse werden für jedes CGP-Training verschiedene Metriken aufgezeichnet, die einen Einblick in die Effizienz der Modelle geben sollen. Diese werden für jede Trainingsiteration gespeichert, um den Verlauf beobachten und bewerten zu können. Die folgende Liste gibt die Metriken an, die in dieser Arbeit näher betrachtet werden:

- Fitness nach Rekombination
- Fitness nach Mutation
- Anzahl aktiver Knoten
- Anteil aktiver Knoten

Die Evaluation umfasst unterschiedliche Techniken zur Bewertung der ausgezeichneten Daten. Diese werden im Folgenden eingeführt und erläutert.

Analyse der Rohdaten: Der erste Evaluationsschritt umfasst eine händische Analyse der Rohdaten. Dabei werden die Ergebnisse der Hyperparameteranalyse näher betrachtet und bewertet. Es werden erste Erkenntnisse über die Effizienz der verschiedenen Rekombinationstypen gesammelt, indem beispielsweise die Anzahl der Rechenknoten verglichen wird. Außerdem wird beobachtet wie die unterschiedlichen Rekombinationsparameter gesetzt werden.
Des weiteren werden die Ergebnisse des CGP-Trainings näher analysiert. Der Trainingserfolg zwischen Rekombination und Mutation können anhand der jeweiligen Fitness-Werte

miteinander verglichen werden.

Bayes'sche Analyse: Die Bayes'sche Analyse wird einerseits für die Sortierung der Effizienz der CGP-Konfigurationen verwendet. Dafür wird wie in Abschnitt 2.6 beschrieben das Plackett-Luce-Modell verwendet. Für die einfacheren Testszenarien Parity und Keijzer können so die Ergebnisse, der aus der Hyperparameteranalyse erhaltenen besten CGP-Konfigurationen miteinander verglichen werden.
Für weitere Bewertungen wird das Gammaverteilung-basierte genutzt. Durch die Berechnung des Mittelwerts und HPDI der Iterationen für jede CPG-Konfiguration können sowohl die einfacheren als auch die komplexeren Testszenarien ausgewertet werden. Für Parity und Keijzer kann so eingeschätzt werden, welche CGP-Konfiguration die Ausgangsprobleme schneller lösen kann als andere. Außerdem kann bewertet werden wie hoch die Streuung dieser Ergebnisse ist. Für die komplexeren Ausgangsprobleme kann beobachtet werden, welche Auswirkungen die Änderung der Rekombinationsparameter auf die Effizienz der CGP-Modelle hat.
Die *Prior-Sensitivitätsanalyse*, die Cui et al. in ihrem Paper untersuchen, wird aus Gründen des Umfangs in dieser Arbeit nicht näher betrachtet [CMH23].

Graphische Evaluation: Für alle Testszenarien kann neben der bayes'schen Analyse eine graphische Evaluation ausgeführt werden. Dafür werden nicht die Endergebnisse des CGP-Trainings betrachtet, sondern deren Verlauf. Es werden die Fitnesswerte und Anteile der aktiven Knoten über die Iterationen hinweg geplottet. Dabei werden die Mittelwerte und Standardabweichungen der Metriken verwendet. Im Anschluss können die Plots visuell bewertet werden.

4 Ergebnisse

Ergebnisse

TODO: Argumentation, dass Offset nicht so viel bringt (damit Kapitel „praktischer Teil“ darauf verweisen kann)
- HPO: mehrere Durchläufe wählen garkeinen Offset
- restliche Tests vergleichen zwischen mit Offset und ohne

4.1 Ergebnisse: Boolesche Probleme

4.1.1 Ergebnisse: Parity

Analyse der Rohdaten

Bayes'sche Analyse

Graphische Evaluation

4.1.2 Ergebnisse: Encode

Analyse der Rohdaten

Bayes'sche Analyse

Graphische Evaluation

4.1.3 Ergebnisse: Decode

Analyse der Rohdaten

Bayes'sche Analyse

Graphische Evaluation

TODO: ggf mehr Testprobleme einfügen

4.2 Ergebnisse: Symbolische Regression

4.2.1 Ergebnisse: Keijzer

Analyse der Rohdaten

Bayes'sche Analyse

Graphische Evaluation

4.2.2 Ergebnisse: Koza

Analyse der Rohdaten

Bayes'sche Analyse

Graphische Evaluation

TODO: ggf mehr Testprobleme einfügen

4.3 Zusammenfassung der Ergebnisse

5 Fazit aus Ausblick

- Motivation nochmal aufgreifen und Forschungsfragen
- Forschungsfragen abschließend erneut beantworten
- Ausblick auf weitere Themen oder offene Fragen

Literatur

[Ahv+19] Milad Taleby Ahvanooey u. a. „A Survey of Genetic Programming and Its Applications“. en. In: *KSII Transactions on Internet and Information Systems* 13.4 (Apr. 2019). ISSN: 19767277. DOI: 10.3837/tiis.2019.04.002.

[Ara20] Lourdes Araujo. „Genetic programming for natural language processing“. en. In: *Genetic Programming and Evolvable Machines* 21.1-2 (Juni 2020), S. 11–32. ISSN: 1389-2576, 1573-7632. DOI: 10.1007/s10710-019-09361-5.

[Car25] Fredrik Bagge Carlson. *baggepinnen/Hyperopt.jl*. original-date: 2018-08-04T11:25:52Z. Jan. 2025. URL: https://github.com/baggepinnen/Hyperopt.jl (besucht am 02. 02. 2025).

[CCL18] Borja Calvo, Josu Ceberio und Jose A. Lozano. „Bayesian inference for algorithm ranking analysis“. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '18. Kyoto, Japan: Association for Computing Machinery, 2018, S. 324–325. ISBN: 9781450357647. DOI: 10.1145/3205651.3205658.

[CHH24] Henning Cui, Michael Heider und Jörg Hähner. „Positional Bias Does Not Influence Cartesian Genetic Programming with Crossover“. en. In: *Parallel Problem Solving from Nature – PPSN XVIII*. Hrsg. von Michael Affenzeller u. a. Bd. 15148. Series Title: Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2024, S. 151–167. ISBN: 978-3-031-70054-5 978-3-031-70055-2. DOI: 10.1007/978-3-031-70055-2_10.

[ČLM13] Matej Črepinšek, Shih-Hsi Liu und Marjan Mernik. „Exploration and exploitation in evolutionary algorithms: A survey“. en. In: *ACM Computing Surveys* 45.3 (Juni 2013), S. 1–33. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/2480741.2480752.

[CMH22] Henning Cui, Andreas Margraf und Jörg Hähner. „Refining Mutation Variants in Cartesian Genetic Programming“. en. In: *Bioinspired Optimization Methods and Their Applications*. Hrsg. von Marjan Mernik, Tome Eftimov und Matej Črepinšek. Bd. 13627. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, S. 185–200. ISBN: 978-3-031-21094-5. DOI: 10.1007/978-3-031-21094-5_14.

[CMH23] Henning Cui, Andreas Margraf und Jörg Hähner. „Equidistant Reorder Operator for Cartesian Genetic Programming:“ en. In: *Proceedings of the 15th International Joint Conference on Computational Intelligence*. Rome, Italy: SCITEPRESS - Science und Technology Publications, 2023, S. 64–74. ISBN: 978-989-758-674-3. DOI: 10.5220/0012174100003595.

[Cui+23] Henning Cui u. a. „Weighted Mutation of Connections To Mitigate Search Space Limitations in Cartesian Genetic Programming“. In: *Proceedings of the 17th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*. FOGA '23. Potsdam, Germany: Association for Computing Machinery, 2023, S. 50–60. ISBN: 9798400702020. DOI: 10.1145/3594805.3607130.

[Cui24a] Henning Cui. *The Positional Bias might not Influence Cartesian Genetic Programming with Crossover*. März 2024. DOI: 10.5281/zenodo.10830014.

[Cui24b] CuiHen. *CuiHen/CGP_with_Crossover_Strategies*. original-date: 2024-03-15T09:54:31Z. Apr. 2024. URL: https://github.com/CuiHen/CGP_with_Crossover_Strategies (besucht am 11. 06. 2024).

[CWM07] Janet Clegg, James Alfred Walker und Julian Frances Miller. „A new crossover technique for Cartesian genetic programming“. en. In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. London England: ACM, Juli 2007, S. 1580–1587. ISBN: 978-1-59593-697-4. DOI: 10.1145/1276958.1277276.

[DDL19] Benjamin Doerr, Carola Doerr und Johannes Lengler. „Self-adjusting mutation rates with provably optimal success rules“. en. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Prague Czech Republic: ACM, Juli 2019, S. 1479–1487. ISBN: 978-1-4503-6111-8. DOI: 10.1145/3321707.3321733.

[Dem+22] Munise Didem Demirbas u. a. „Stress Analysis of 2D-FG Rectangular Plates with Multi-Gene Genetic Programming“. en. In: *Applied Sciences* 12.16 (Aug. 2022), S. 8198. ISSN: 2076-3417. DOI: 10.3390/app12168198.

[ES15] A.E. Eiben und J.E. Smith. *Introduction to Evolutionary Computing*. en. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. ISBN: 978-3-662-44873-1 978-3-662-44874-8. DOI: 10.1007/978-3-662-44874-8.

[GP15] Brian W. Goldman und William F. Punch. „Analysis of Cartesian Genetic Programming's Evolutionary Mechanisms". In: *IEEE Transactions on Evolutionary Computation* 19.3 (2015), S. 359–373. DOI: 10.1109/TEVC.2014.2324539.

[Has+19] Ahmad Hassanat u. a. „Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach". en. In: *Information* 10.12 (Dez. 2019), S. 390. ISSN: 2078-2489. DOI: 10.3390/info10120390.

[HLS99] Myron E Hohil, Derong Liu und Stanley H Smith. „Solving the N-bit parity problem using neural networks". In: *Neural Networks* 12.9 (1999), S. 1321–1323. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(99)00069-6.

[Kal+23] Roman Kalkreuth u. a. „Towards a General Boolean Function Benchmark Suite". en. In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. Lisbon Portugal: ACM, Juli 2023, S. 591–594. ISBN: 9798400701207. DOI: 10.1145/3583133.3590685.

[Kal20] Roman Kalkreuth. „A Comprehensive Study on Subgraph Crossover in Cartesian Genetic Programming:" en. In: *Proceedings of the 12th International Joint Conference on Computational Intelligence*. Budapest, Hungary: SCITEPRESS - Science und Technology Publications, 2020, S. 59–70. ISBN: 978-989-758-475-6. DOI: 10.5220/0010110700590070.

[KK17] Paul Kaufmann und Roman Kalkreuth. „An empirical study on the parametrization of cartesian genetic programming". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '17. Berlin, Germany: Association for Computing Machinery, 2017, S. 231–232. ISBN: 9781450349390. DOI: 10.1145/3067695.3075980.

[KK20] Paul Kaufmann und Roman Kalkreuth. „On the Parameterization of Cartesian Genetic Programming". In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. 2020, S. 1–8. DOI: 10.1109/CEC48606.2020.9185492.

[Kom18] Michael Kommenda. „Local Optimization and Complexity Control for Symbolic Regression". Dissertation. Linz, Austria: Johannes Kepler University Linz, 2018. URL: https://resolver.obvsg.at/urn:nbn:at:at-ubl:1-21036 (besucht am 31. 01. 2025).

[Koz95] J. R. Koza. „Survey of genetic algorithms and genetic programming“. en. In: *Proceedings of WESCON'95*. San Francisco, CA, USA: IEEE, 1995, S. 589. ISBN: 978-0-7803-2636-1. DOI: 10.1109/WESCON.1995.485447.

[Kra+13] Krzysztof Krawiec u. a., Hrsg. *Genetic Programming: 16th European Conference, EuroGP 2013, Vienna, Austria, April 3-5, 2013. Proceedings*. en. Bd. 7831. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-37206-3. DOI: 10.1007/978-3-642-37207-0.

[Kru12] John Kruschke. „Bayesian Estimation Supersedes the t Test“. In: *Journal of experimental psychology. General* 142 (Juli 2012). DOI: 10.1037/a0029146.

[Li+24] Yanjie Li u. a. *Generative Pre-Trained Transformer for Symbolic Regression Base In-Context Reinforcement Learning*. _eprint: 2404.06330. 2024. URL: https://arxiv.org/abs/2404.06330.

[MC24] Nour Makke und Sanjay Chawla. „Interpretable scientific discovery with symbolic regression: a review“. In: *Artificial Intelligence Review* 57.1 (Jan. 2024), S. 2. ISSN: 1573-7462. DOI: 10.1007/s10462-023-10622-0.

[Mil20] Julian Francis Miller. „Cartesian genetic programming: its status and future“. en. In: *Genetic Programming and Evolvable Machines* 21.1-2 (Juni 2020), S. 129–168. ISSN: 1389-2576, 1573-7632. DOI: 10.1007/s10710-019-09360-6.

[MN18] Nicola Milano und Stefano Nolfi. *Scaling Up Cartesian Genetic Programming through Preferential Selection of Larger Solutions*. arXiv:1810.09485. Okt. 2018. URL: http://arxiv.org/abs/1810.09485 (besucht am 14. 10. 2024).

[Nen80] M. Nenning. „Bayes'sche Inferenz bei der statistischen Auswertung von Marktdaten“. In: *Zeitschrift für Operations Research* 24.8 (Dez. 1980), B259–B273. ISSN: 1432-5217. DOI: 10.1007/BF01918729.

[Oli+18] Luiz Otavio Vilas Boas Oliveira u. a. „Analysing Symbolic Regression Benchmarks under a Meta-Learning Approach“. en. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. arXiv:1805.10365 [cs]. Juli 2018, S. 1342–1349. DOI: 10.1145/3205651.3208293.

[OLM18] Patryk Orzechowski, William La Cava und Jason H. Moore. „Where are we now? a large benchmark study of recent symbolic regression methods“. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. Kyoto, Japan: Association for Computing Machinery, 2018, S. 1183–1190. ISBN: 9781450356183. DOI: 10.1145/3205455.3205539.

[Pät24] David Pätzel. *dpaetzel/cmpbayes*. original-date: 2022-03-24T10:52:44Z. Juli 2024. URL: https://github.com/dpaetzel/cmpbayes (besucht am 27. 01. 2025).

[Pey20] Pablo Peyrolón. „Definition des Satzes von Bayes oder das Bayes-Theorem“. de. In: *Der Satz von Bayes*. Series Title: essentials. Wiesbaden: Springer Fachmedien Wiesbaden, 2020, S. 13–21. ISBN: 978-3-658-31022-6 978-3-658-31023-3. DOI: 10.1007/978-3-658-31023-3_2.

[PG17] G. Pavai und T. V. Geetha. „A Survey on Crossover Operators“. en. In: *ACM Computing Surveys* 49.4 (Dez. 2017), S. 1–43. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3009966.

[SB01] S. Sette und L. Boullart. „Genetic programming: principles and applications“. en. In: *Engineering Applications of Artificial Intelligence* 14.6 (Dez. 2001), S. 727–736. ISSN: 09521976. DOI: 10.1016/S0952-1976(02)00013-1.

[SB18] José Eduardo da Silva und Heder S. Bernardino. „Cartesian Genetic Programming with Crossover for Designing Combinational Logic Circuits“. In: *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*. Okt. 2018, S. 145–150. DOI: 10.1109/BRACIS.2018.00033.

[Sys89] Gilbert Syswerda. „Uniform Crossover in Genetic Algorithms“. en. In: *Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA, June 1989*. Fairfax, Virginia, USA, Jan. 1989. ISBN: 1-55860-066-3. URL: https://www.researchgate.net/publication/201976488_Uniform_Crossover_in_Genetic_Algorithms (besucht am 27. 12. 2024).

[TST22] Ali Torabi, Arash Sharifi und Mohammad Teshnehlab. „Using Cartesian Genetic Programming Approach with New Crossover Technique to Design Convolutional Neural Networks“. en. In: *Neural Processing Letters* (Dez. 2022). ISSN: 1370-4621, 1573-773X. DOI: 10.1007/s11063-022-11093-0.

[Whi+13] David R. White u. a. „Better GP benchmarks: community survey results and proposals“. en. In: *Genetic Programming and Evolvable Machines* 14.1 (März 2013), S. 3–29. ISSN: 1389-2576, 1573-7632. DOI: 10.1007/s10710-012-9177-2.

[WM08] James Alfred Walker und Julian Francis Miller. „The Automatic Acquisition, Evolution and Reuse of Modules in Cartesian Genetic Programming“. In: *IEEE Transactions on Evolutionary Computation* 12.4 (2008), S. 397–417. DOI: 10.1109/TEVC.2007.903549.

[YM01] Tina Yu und Julian Miller. „Neutrality and the Evolvability of Boolean Function Landscape“. In: *Genetic Programming*. Hrsg. von Julian Miller u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, S. 204–217. ISBN: 978-3-540-45355-0.

A Anhang