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
Henning Cui
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. 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 ist 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:
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 beschrieben und deren Ergebnisse ausgewertet. Abschließend wird in Abschnitt 4 ein Fazit aus den Ergebnissen zusammengefasst, sowie ein Ausblick auf weitere mögliche Forschungsfragen gegeben.

2 Grundlagen

Für den praktischen Teil dieser Arbeit 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 eingeführt:

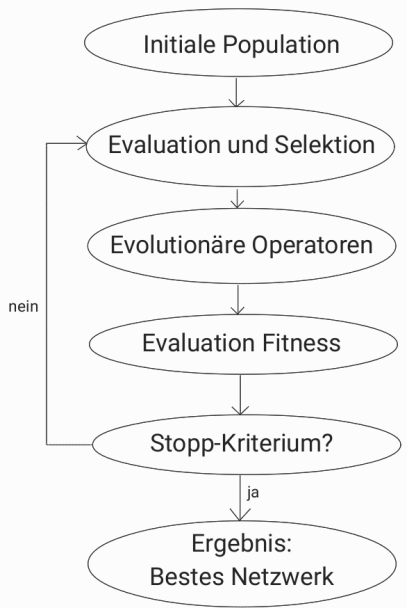

Abbildung 2.1: Aufbau CGP, angelehnt an [TST22]

Die Abbildung 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 [Cui24].
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 Abbildungen kann beispielhaft näher erläutert werden, wie die Chromosome in CGP aufgebaut sind und funktionieren. Dieses Beispiel wird in [TST22] eingeführt.

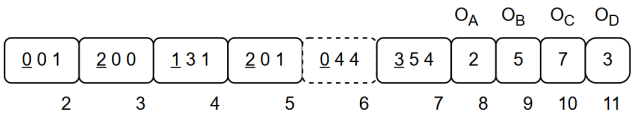

Abbildung 2.2: Beispiel Genotyp, angelehnt an [TST22]

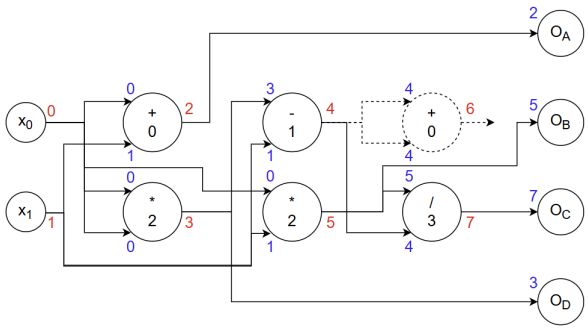

Abbildung 2.3: Beispiel Phänotyp, angelehnt an [TST22]

Abbildung 2.2 zeigt einen Genotyp und Abbildung 2.3 den daraus resultierenden Phänotyp. Anhand des Phänotyps 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 2.2 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 2.3 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.

Als letzter Punkt fällt innerhalb des Beispiels auf, dass ein Knoten und dessen Kanten mit gestrichelten Linien dargestellt sind. Dies ist ein sogenannter inaktiver Knoten und wird für die Berechnung des Endergebnisses nicht gebraucht. Ob ein Knoten für diese Berechnung 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. Alle Knoten, die bis dahin besucht wurden, werden aktive Knoten genannt und sind für die Berechnung des Endergebnisses eines Chromosoms relevant.

Schließlich soll anhand der eingeführten Abbildung 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:

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 Ausga-

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

ben 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. Im Quellcode von Cui wird dies erzielt, indem Knoten nur Ausgänge von denjenigen Knoten verwenden dürfen, deren Indices kleiner sind als der eigene Index.

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 im folgenden Abschnitt 2.3 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 erläutert werden:

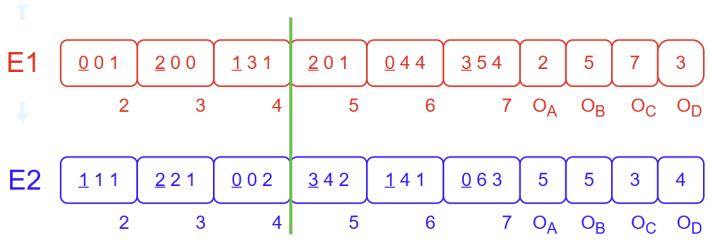

Abbildung 2.4: Eltern-Chromosomen One-Point Rekombination, angelehnt an [TST22]

Das erste Eltern-Chromosom (E1, rot) in diesem Beispiel wurde aus Abbildung **??** 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 2.4 grün markiert. Die Eltern-Chromosomen werden anschließend an dieser Stelle geteilt und überkreuzt zusammengesetzt. Für dieses Beispiel ergeben sich die beiden folgenden Nachwuchs-Chromosomen:

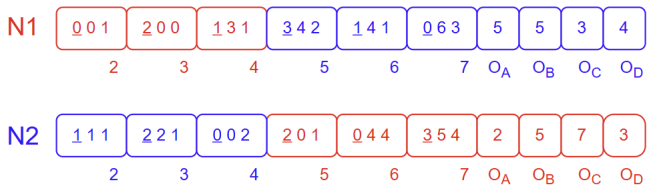

Abbildung 2.5: Nachwuchs-Chromosomen One-Point Rekombination, angelehnt an [TST22]

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:

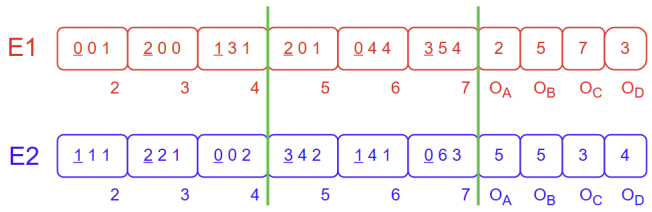

Abbildung 2.6: Eltern-Chromosomen Two-Point Rekombination, angelehnt an [TST22]

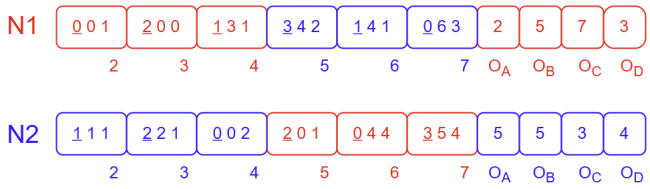

Abbildung 2.7: Nachwuchs-Chromosomen Two-Point Rekombination, angelehnt an [TST22]

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 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 Abbildungen 2.8 und 2.9 lässt sich die Uniform Rekombination beispielhaft erklären:

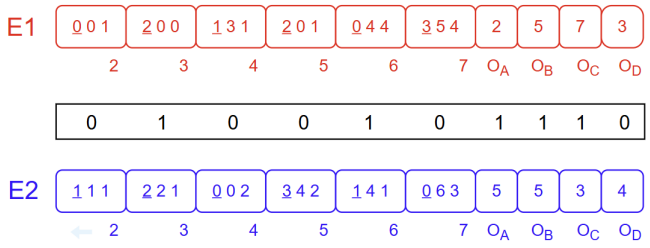

Abbildung 2.8: Eltern-Chromosomen Uniform Rekombination, angelehnt an [TST22]

Abbildung 2.8 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 folgende Abbildung zeigt die resultierenden Ergebnisse für dieses Beispiel der Uniform Rekombination.

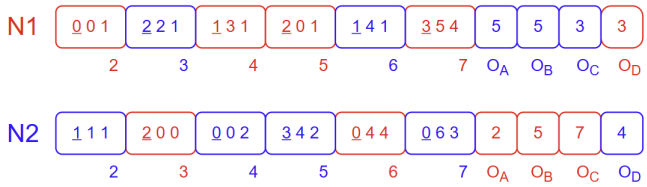

Abbildung 2.9: Nachwuchs-Chromosomen Uniform Rekombination, angelehnt an [TST22]

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. Diese 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“ erreicht. 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. 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, in dem das ausgewählte Zeichen des Genotyps zufällig geändert wird [Koz95]. Dies hat für unterschiedliche Knotenarten des entsprechenden Phänotyps andere Effekte.
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 2.2 erneut eingeführt. Es wird angenommen, dass dieses Chromosom durch vorherige Rekombination entstanden ist und somit im nächsten Schritt mutiert werden soll.

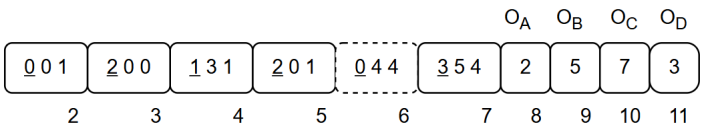

Abbildung 2.10: Beispiel Genotyp vor Mutation, angelehnt an [TST22]

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 zwischen einer dieser Parameter 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 folgende Abbildung zeigt das Ergebnis des Mutationsbeispiels (ohne Kennzeichnung der neuen aktiven Knoten). Die mutierten Teile des Chromosoms werden rot hervorgehoben.

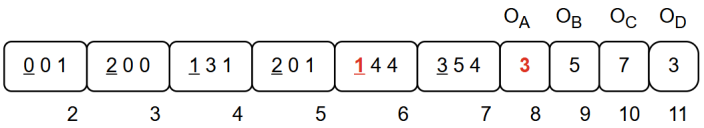

Abbildung 2.11: Beispiel Genotyp nach Mutation

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 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 Hyperparameteranalyse

2.7 statistische Analyse mit ANOVA

- Grundsätzliches Vorgehen über Generationen hinweg (anhand von GP erklären und dann Unterschied zu CPG erklären): Grafik -> „Genetic Programming Principles and Applications“
- Aufbau des Graphen
- Wozu CGP? Welche Probleme können damit gelöst werden? -> aus Eingabe(n) eine oder mehrere Ausgabe(n) generieren
- Arten von Knoten
- aktive und inaktive Knoten
- Exploration vs Exploitation
- Selektion
  - Ziel Selektion grundsätzlich
  - neutral search
  - Elitisten (Was ist das und Sinn)
  - mu lambda
  - ggf tournament
- Rekombination
  - Single-point
  - two-point
  - uniform
  - Rekombinationsraten
    - Ziel: Geeignetes Maß an Exploration vs Exploitation finden
    - Clegg et al
    - Torabi et al

        * One-Fifth Rule der Mutationsrate

- Mutation
    - Mutation grundsätzlich
    - single active mutation -> Mutationsrate nicht betrachtet (weniger Parameter in HPO)
- Hyperparameter analysis -> Sampler
- ANOVA

3 Praktischer Teil

3.1 Aufbau der Experimente

3.2 Ergebnisse

- Versuchsaufbau und Durchführung
  - an Hennings Code orientiert
  - Testszenarien -> welche Testdaten und Berechnung fitness
    - boolean Problems
    - symbolic regression
    - ggf. Ameisenoptimierungsproblem
  - weitere Einstellungen (Welche Selektionsverfahren, Mutationsverfahren, Rekomb…)
  - Hyperparameteroptimierung
    - Welche Bib benutzt
    - Welche Parameter optimiert; mit welchen Ranges
    - 10 Ausführungen pro Optimierungsschritt
    - händische Auswahl der besten Ergebnisse über Zeit
  - Ausführen mehrerer Durchläufe, um statistische Auswertung vorzunehmen
- Ergebnisse und Evaluation
  - Metriken zur Auswertung
  - händische Auswertung der Ergebnisse

- bayesian Auswertung der Ergebnisse
- Bewertung der statistischen Aussagekraft mit ANOVA

4 Fazit aus Ausblick

- Motivation nochmal aufgreifen und Forschungsfragen
- Sinn der einzelnen Kapitel in der Arbeit erläutern
- 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. URL: http://itiis.org/digital-library/manuscript/2311 (besucht am 18. 09. 2024).

[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. URL: http://link.springer.com/10.1007/s10710-019-09361-5 (besucht am 25. 11. 2024).

[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. URL: https://link.springer.com/10.1007/978-3-031-70055-2_10 (besucht am 18. 10. 2024).

[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. URL: https://dl.acm.org/doi/10.1145/2480741.2480752 (besucht am 29. 12. 2024).

[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. URL: https://link.springer.com/10.1007/978-3-031-21094-5_14 (besucht am 10. 12. 2024).

[Cui24] 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. URL: https://dl.acm.org/doi/10.1145/1276958.1277276 (besucht am 30. 05. 2023).

[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. URL: https://dl.acm.org/doi/10.1145/3321707.3321733 (besucht am 03. 01. 2025).

[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. URL: https://www.mdpi.com/2076-3417/12/16/8198 (besucht am 25. 11. 2024).

[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. URL: https://www.mdpi.com/2078-2489/10/12/390 (besucht am 10. 12. 2024).

[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. URL: https://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0010110700590070 (besucht am 11. 06. 2024).

[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. URL: http://ieeexplore.ieee.org/document/485447/ (besucht am 09. 08. 2024).

[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. URL: http://link.springer.com/10.1007/978-3-642-37207-0 (besucht am 01. 07. 2024).

[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. URL: http://link.springer.com/10.1007/s10710-019-09360-6 (besucht am 11. 06. 2024).

[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).

[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. URL: https://dl.acm.org/doi/10.1145/3009966 (besucht am 10. 12. 2024).

[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. URL: https://linkinghub.elsevier.com/retrieve/pii/S0952197602000131 (besucht am 09. 08. 2024).

[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. URL: https://link.springer.com/10.1007/s11063-022-11093-0 (besucht am 30. 05. 2023).

A Anhang