Christian-Albrechts-Universität
Institut für Informatik
Lehrstuhl Kognitive Systeme
Prof. Dr. Sommer
Olshausenstr. 40
24098 Kiel

WS 2009/2010

Diplomarbeit

Effektivitätssteigerung einer Methode für Neuroevolution zum Erlernen von Roboterreglern und Bildklassifikatoren

Sven Grünewald
(Matrikelnummer: 584940)

17. März 2010

Betreut durch Prof. Dr. Nils T. Siebel

Steinbergkirche, 17. März 2010

Hiermit versichere ich, dass ich die Arbeit selbständig verfasst und ausschließlich die angegebenen Hilfsmittel und Quellen verwendet habe.

Sven Grünewald

Kurzfassung

EANT2 ist ein Evolutionärer Algorithmus, der künstliche Neuronale Netze von Grund auf entwerfen und trainieren kann. Dabei hat EANT2 seine Leistungsfähigkeit bereits an einer Reihe von Problemen demonstrieren können. In dieser Arbeit wird EANT2 benutzt, um einen Kantendetektor zu generieren und es wird versucht, einen Regler zu erstellen, der einen Helikopter im Schwebeflug hält.

Um EANT2 weiter zu verbessern, werden erweiterte Protokollierungsfunktionen eingebaut. Außerdem wird versucht, die Leistung des Parameteroptimierers, der die Neuronalen Netze trainiert, durch Erhöhung der Populationsgröße zu steigern. Weiterhin wird untersucht, ob eine Beschleunigung durch Entfernen der Reoptimierung der Elterngeneration möglich und sinnvoll ist.

Auch wird untersucht, ob eine Beschleunigung von EANT2 durch Verringerung der Funktionsevaluationen der Fitnessfunktion durch Verwendung lokaler Modelle mittels LMM-CMAES sinnvoll ist.

In dieser Arbeit wird der Partikel-Schwarm-Optimierungsalgorithmus, der sich wachsender Beliebtheit erfreut, als Alternative zu CMAES in EANT2 etabliert.

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Einführung und Überblick

Mit **EANT2** (Evolutionary Acquisition of Neural Topologies) liegt ein Algorithmus zur Neuroevolution vor, der verschiedene Probleme lösen kann. Dafür werden künstliche Neuronale Netze automatisch und von Grund auf generiert und trainiert. EANT2 wurde hierbei schon an einer Reihe von Problemen angewandt [SS08, SK06, SGS08] und konnte gute Ergebnisse liefern. Es hat sich gezeigt, dass EANT2 anderen Methoden zur Neuroevolution überlegen ist [SKS07].

1.1.1 Gliederung

In diesem Kapitel werden die Grundlagen der Methoden und Probleme vermittelt. Dabei erklären die Abschnitte 1.2 - 1.4 die zugrundeliegenden Techniken bzw. Methoden.

Abschnitt 1.5 beschreibt kurz, welche Maßnahmen zur Verbesserung von EANT2 untersucht wurden und in 1.6 werden die Optimierungsprobleme vorgestellt, auf die EANT2 angewandt wurde.

Für eines dieser Probleme, die Kantendetektion, werden die Ergebnisse in Kapitel 2 vorgestellt. Die Ergebnisse des Helikopterflugs finden sich in Kapitel 3.

In Kapitel 4 sind alle durchgeführten Verbesserungen an EANT2 aufgeführt, die nicht die Parameteroptimierung betreffen.

Kapitel 5 beschreibt den in Kapitel 6 und 7 verwendeten Versuchsaufbau für Parameteroptimierer. Wobei sich Kapitel 6 damit beschäftigt, CMAES durch PSO zu ersetzen. Das Kapitel 7 hingegen untersucht die Auswirkungen einer Populationsvergrößerung der CMAES-Population.

Ob ein Geschwindigkeitsgewinn durch die Benutzung lokaler Metamodelle möglich ist, beschreibt Kapitel 8.

Eine Zusammenfassung der Ergebnisse und Ausblicke auf weitere Forschungsmöglichkeiten bietet Kapitel 9

1.2 Neuronale Netze

1.2.1 Einführung

Das menschliche Gehirn wird bis heute von keinem Computer in den Punkten Lernfähigkeit, Adaption und Robustheit übertroffen. Zwar ist es möglich, durch hochspezialisierte Algorithmen den Menschen vor allem bei mathematischen Berechnungen zu schlagen, aber auch hier erreichen z.B. Menschen mit sogenannten „Inselbegabungen“ Erstaunliches. Die besonders erstaunlichen Ergebnisse von Computern werden allerdings meist bei gut formalisierbaren und vor allem mathematisierbaren Problemen teilweise mit Hilfe von spezialisierter Hardware erreicht. Ein Computer kann in einer Sekunde π schneller berechnen, als ein Mensch in Tagen. Wer allerdings schon einmal mit einer Spracherkennungssoftware telefoniert hat, wird ohne Zweifel zustimmen, dass der Computer noch nicht auf jedem Bereich dem Menschen überlegen ist.

Für einen (erwachsenen) Menschen ist es z.B. kein Problem, eine Katze auf einem Bild als solche zu erkennen, selbst wenn er diese Katze noch niemals vorher gesehen hat und die Katze vor einem unruhigen Hintergrund steht. Außerdem fällt es ihm leicht, aufrecht auf zwei Beinen zu gehen oder mit einem anderen Menschen einen sinnvollen Dialog zu führen, und das alles sogar gleichzeitig. Aber auch ein Mensch ist kurz nach der Geburt zu wenig mehr fähig, als die lebensnotwendigen Funktionen aufrecht zu erhalten und vor allem zu lernen.

Betrachtet man nun noch die geringe Größe und den geringen „Energieverbrauch“ des Gehirns, erkennt man, wie effizient das Gehirn arbeitet. Es ist somit naheliegend, sich von der Funktionsweise des menschlichen Gehirns inspirieren zu lassen und zu versuchen, Teile davon auf Computer zu übertragen.

1.2.2 Biologische Neuronale Netze

Eine der wichtigsten Komponenten des menschlichen Gehirns sind die Nervenzellen, auch Neuronen genannt. Von diesen Nervenzellen besitzt das

menschliche Gehirn rund 100 Milliarden [FS04]. Diese Neuronen bestehen aus:

- **Dendriten** Sie dienen zur Aufnahme von Informationen anderer Neuronen mittels deren Synapsen.
- **Soma** (auch **Perikaryon**) Dem Zellkörper in dem die eintreffenden Signale verarbeitet und summiert werden.
- **Axon** Hier wird das Signal weitergeleitet, wenn das durch die Dendriten anliegende Signal im Soma ein gewisses Schwellenpotenzial überschreitet.
- **Synapsen** Die Schnittstelle zu anderen Neuronen. Sie können unterschiedlich stark auf das Neuron einwirken.

An den Dendriten eines Neurons liegen die Signale anderer Neuronen an, die diese mittels Synapsen an ihren Axionterminalen weiterleiten. Hierbei hat ein Neuron bis zu 1000 Dendriten, an denen zusammen bis zu 100000 Synapsen [LR08] Impulse weiterleiten. Diese empfangenen Signale werden an das Soma weitergeleitet. Erreichen die hier eintreffenden Signale einen bestimmten Schwellenwert, so wird ein Aktionspotential ausgelöst, das immer gleiche Form, Größe und Dauer hat („**Alles-oder-nichts-Regel**“). Dieses Signal wird dann über das Axon und dessen Synapsen an andere Neuronen oder z.B. Muskeln weitergegeben. Dieses Axon kann dabei eine Länge von bis zu einem Meter erreichen. Das Axon kann zum Schutz und als elektrische Isolierung von einer Markscheide bzw. Myelinscheide aus Schwannzellen umgeben sein, die in die Kategorie der Gliazellen[1] gehören. An sogenannten Ranvier-Schnürringen fehlt diese Isolierung und die Signale „springen“ von Schnürring zu Schnürring.

Eine schematische Darstellung eines Neuron zeigt Abbildung 1.1.

Neuronale Netze sind somit zum Teil extrem stark miteinander vernetzt und auch häufig zyklisch geschaltet. Auch wenn Neuronen (nach heutigen Maßstäben) nicht schnell arbeiten (Reaktionszeiten im Millisekundenbereich), erreichen sie doch eine sehr gute Echtzeitfähigkeit durch ihre große Anzahl und die starke Parallelität der Berechnungen. Hier liegt allerdings auch das Problem, wenn man Neuronale Netze am Computer simulieren möchte. Mit der heutigen Technik ist es nicht möglich, diesen hohen Grad an Parallelität zu erreichen und in Echtzeit zu simulieren, auch wenn im Moment mit modernen Grafikkarten (z.B. Radeon HD 5970 mit 1600 Streamprozessoren) schon extreme parallele Rechenkraft selbst im „Consumerbereich“ zur Verfügung steht.

[1] Vereinfacht: Gliazellen sind Nervenzellen, die keine Neuronen sind.

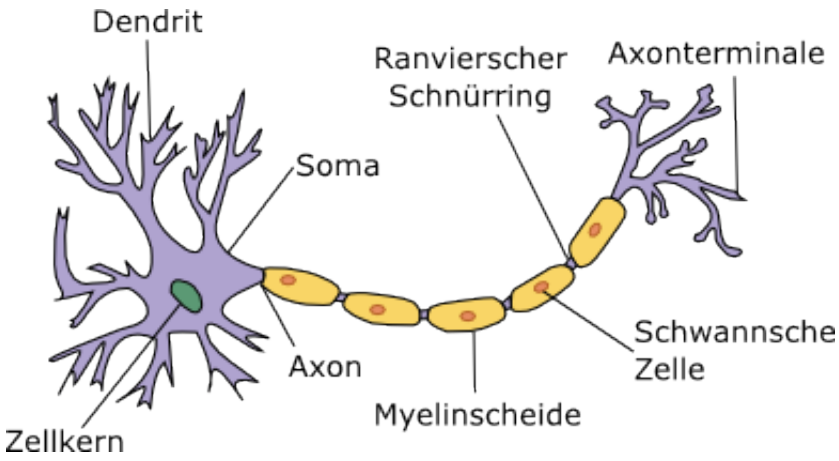


Abbildung 1.1: Schematische Darstellung eines Neurons. (Grafik aus [IN06])

1.2.3 Künstliche Neuronen

Wie oben dargelegt, sind Neuronale Netze in der Lage, komplexe Probleme zuverlässig zu lösen. Es ist also naheliegend, die Funktionsweise von Neuronen zu formalisieren und für Computer umzusetzen. Herkömmliche künstliche[2] Neuronale Netze sind dabei nur von biologischen Netzen inspiriert und versuchen nicht, deren Funktionsweise 1 zu 1 umzusetzen. Hierfür wäre auch eine zu starke Parallelisierung notwendig.

Die nachfolgenden Ausführungen basieren zum Großteil auf dem Vorlesungsskript [Som06] von Prof. Dr. Sommer bzw. auf [Bis95].

Ein (künstliches) Neuron i besteht aus $n \in \mathbb{N}$ **Eingängen** $x_1 ... x_n \in \mathbb{R}$, n **Gewichten** $w_{i1} ... w_{in} \in \mathbb{R}$, einer **Propagierfunktion** $g_i : \mathbb{R}^n \to \mathbb{R}$, einer **Aktivierungsfunktion** $f_i : \mathbb{R} \to \mathbb{R}$ sowie einem **Ausgang** $y \in \mathbb{R}$. Es ist weiterhin möglich, einen Schwellenwert θ durch Setzen von w_{i0} mit $w_{i0} = -\theta$ einzuführen, wobei $x_0 = 1$ konstant anliegt.

[2] Da sich die folgenden Kapitel praktisch nur noch mit künstlichen Neuronalen Netzen beschäftigen, wird im Folgenden meist auf diesen Zusatz verzichtet.

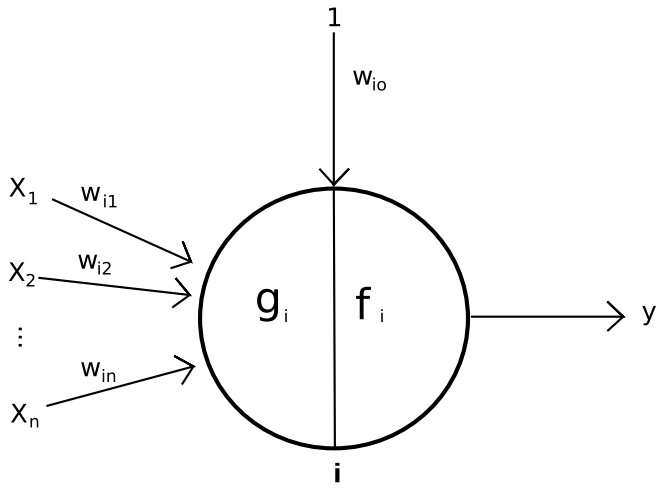


Abbildung 1.2: Künstliches Neuron

Propagierungsfunktion

Die Propagierungsfunktion (g_i) übernimmt in künstlichen Neuronalen Netzen die Aufgabe, die in biologischen Netzwerken die Dendriten übernehmen. Sie fassen die ankommenden Signale der Eingänge bzw. anderer Neuronen zusammen, um sie an das Axon bzw. die Aktivierungsfunktion weiter zu leiten. In biologischen Netzwerken haben hierbei die Neuronen unterschiedlich viele Verbindungen zueinander, so dass manche Neuronen größere Auswirkungen haben, andere kleinere. Dieser Mechanismus wird in künstlichen Netzen durch Gewichte realisiert, die mit den jeweiligen Eingängen multipliziert werden.

In EANT2 wird als Propagierungsfunktion der lineare Assoziator verwendet:

Seien $n \in \mathbb{N}$, $i \in \mathbb{N}$ der Index des Neurons, $x = (x_1, ...x_n)$ das Eingangssignal und $w_{i1}...w_{in}$ die dazugehörigen Gewichte, dann ist $g_i : \mathbb{R}^n \to \mathbb{R}$ gegeben durch:

$$u_i = g_i(x) := \sum_{j=1}^{n} w_{ij} x_j$$

Aktivierungsfunktion

Die Aktivierungsfunktion verarbeitet die Ergebnisse der Propagierungsfunktion und bildet das Ergebnis des Neurons. Die bei biologischen Neuronen herrschende „Alles-oder-Nichts-Regel“[3], die auch in Perzeptron-Neuronen verwendet wird, findet bei den in EANT2 verwendeten Neuronen keine Anwendung. Hier werden Neuronen mit einer der folgenden Aktivierungsfunktionen verwendet, von denen manche mit einem zusätzlichem Parameter $p \in \mathbb{R}$ versehen sind:

- Linear

$$f_p(u_i) := u_i p$$

- Tangens Hyperbolicus

$$f_p(u_i) := \tanh(u_i p)$$

- Logistisch

$$f_p(u_i) := \frac{1}{1 + e^{-u_i p}}$$

- Gauss'sche

$$f(u_i) := e^{-u_i u_i}$$

- Sigmoidal

$$f(u_i) := \frac{u_i}{1 + |u_i|}$$

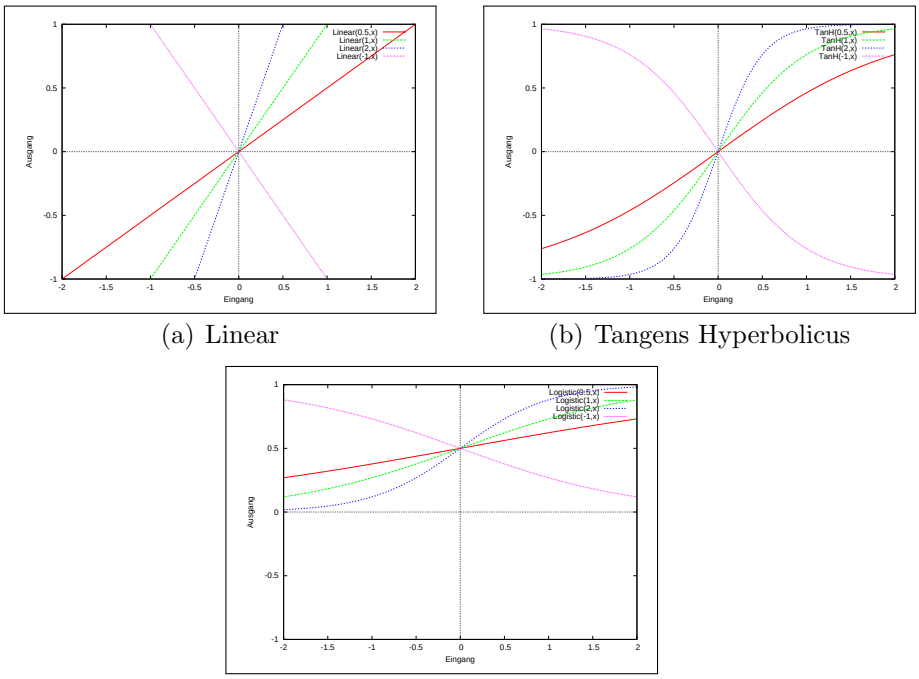
- Positiv Sigmoidal

$$f(u_i) := \left(\frac{1}{1 + |u_i|} + 1 \right) \cdot 0,5$$

- Signum-Quadrat

$$f(u_i) := u_i \cdot |u_i|$$

[3]Ein Neuron gibt kein Signal, bis die Eingänge einen Schwellenwert überschreiten. Dann geben sie ein immer gleich großes Signal.



(a) Linear

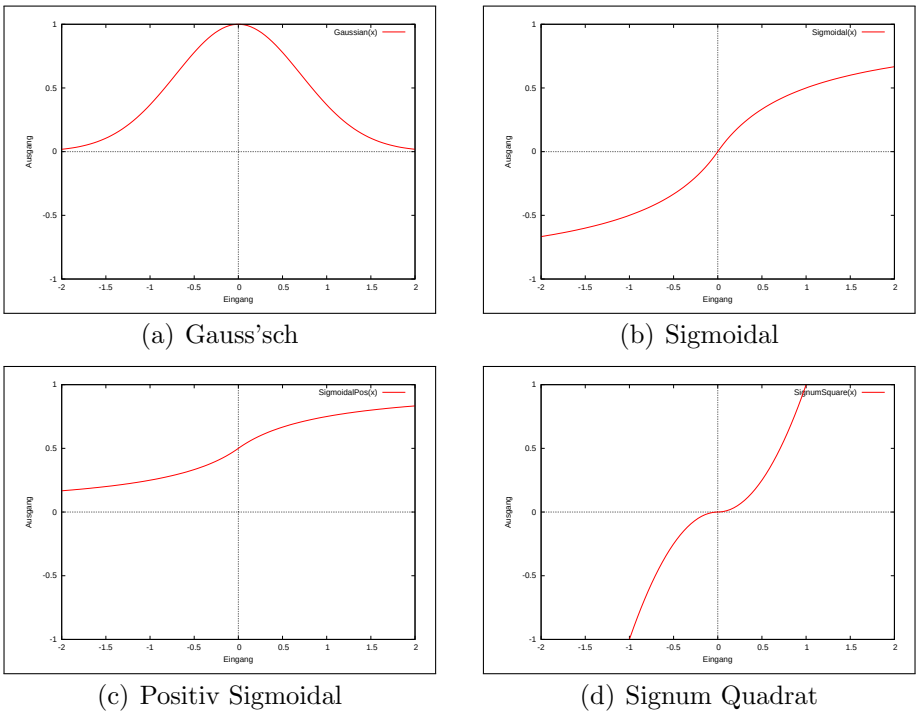
(b) Tangens Hyperbolicus

(c) Logistisch

Abbildung 1.3: EANT2 Aktivierungsfunktionen mit Parameter $p = 0,5$, $p = 1$, $p = 2$ und $p = -1$

Radiale-Basis-Funktions-Neuronen

Radiale-Basis-Funktions-Neuronen (RBF-Neuronen) unterscheiden sich von den oben vorgestellten Neuronen. Sie verwenden nicht den linearen Assoziator als Aktivierungsfunktion. Stattdessen besitzt jedes Neuron ein sogenanntes Zentrum $c \in \mathbb{R}^n$, welches als Dimension die Anzahl $n \in \mathbb{N}$ der Eingänge des Neurons hat. Das Ergebnis der Aktivierungsfunktion ist dann der gewichtete (meist euklidische) Abstand des Eingabevektors zu dem Zentrum. Die Basisfunktion hat in der Regel dann ihren größten Ausschlag, wenn die Eingabe u_i dem Zentrum entspricht, also $||u_i - c|| = 0$.



(a) Gauss'sch (b) Sigmoidal

(c) Positiv Sigmoidal (d) Signum Quadrat

Abbildung 1.4: Parameterlose Aktivierungsfunktionen in EANT2

1.2.4 Künstliche Neuronale Netze

Werden mehrere Neuronen miteinander verschaltet, so spricht man von einem Neuronalen Netz. Dabei teilt man die Neuronen disjunkt in mehrere Schichten auf. Die Neuronen, welche mit dem Eingabesignal verbunden sind, werden Eingabeschicht genannt (grün in Abbildung 1.5). Die Neuronen, deren Ausgabe die Gesamtausgabe bilden (gelb), werden Ausgabeschicht genannt und die restlichen Neuronen bilden die Verdeckte(n) Schicht(en) (blau). Im einfachsten Fall (wie auch in der Abbildung) führen Ausgänge der i-ten Schicht ausschließlich zu den Eingängen der $i+1$-ten Schicht. Dabei spricht das Eingangsignal als 0-te Schicht die Eingänge der ersten Schicht an und die Ausgangssignale der letzten Schicht bilden den Gesamtausgang.

Bei erweiterten Modellen (die im Folgenden auch verwendet werden) sind zusätzliche Verbindungen innerhalb einer Ebene, rückwärtsgerichtete Verbindungen, sowie Verbindungen über mehrere Ebenen hinweg möglich. So ist es beispielsweise möglich, Eingabesignale direkt an Neuronen in einer verdeckten Schicht zu senden, unter Auslassung der Eingabeschicht.

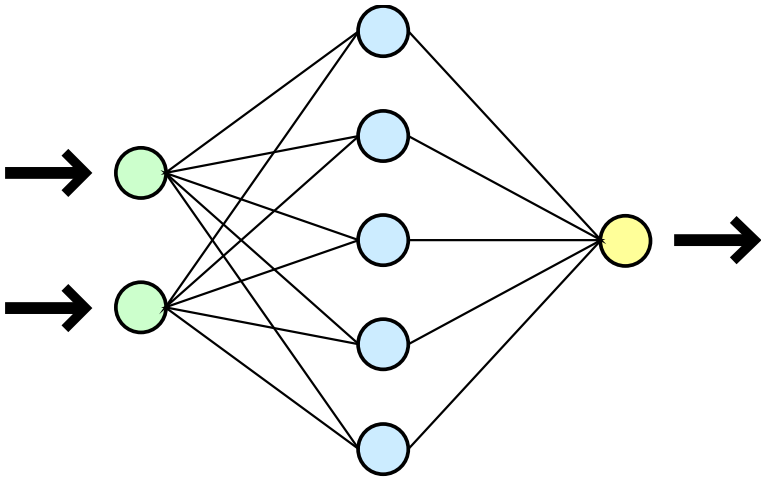


Abbildung 1.5: Einfaches, vollständig verschaltetes Neuronales Netz mit zwei Eingängen, fünf versteckten Knoten und einem Ausgang. Aus [DM06]

1.3 Evolutionäre Algorithmen

1.3.1 Einführung

Der Begriff **Evolutionäre Algorithmen** (**EA**) kann als Überbegriff für mehrere (nach Maßstäben der Informatik) schon ältere, randomisierte Verfahren angesehen werden. Evolutionäre Algorithmen zeigen besonders bei komplexen multimodalen Problemen ihre Stärke. Unimodale Probleme lösen sie hingegen schlechter als herkömmliche „Hillclimbing"-Algorithmen, die wiederum bei multimodalen Problemen häufiger in lokalen Maxima konvergieren und nicht das globale Maximum erreichen.

Einen ausführlichen Einblick in Evolutionäre Algorithmen gibt [ES03]. Im folgenden wird ein kurzer Überblick gegeben.

In den 60er und 70er Jahren entwickelten sich Unabhängig voneinander drei verschiedene Ansätze:

- „Evolutionary programming" (u.a.[FOW65])
- „Genetic algorithm" (u.a. [Hol73])
- „Evolutionsstrategie" (u.a. [Rec73])

Alle diese Verfahren sind inspiriert von Darwins Evolutionstheorie [Dar59], insbesondere von dem „Surival of the fittest". Dabei wird davon ausgegangen, dass eine gewisse Zahl an Individuen in einer Umgebung mit begrenzten Ressourcen existiert, sich fortpflanzt und je nach ihrer „Stärke" überlebt oder ausstirbt. Die obigen Verfahren benutzen dabei unterschiedliche Varianten der in 1.3.2 kurz angerissenen Schritte, wobei nicht alle Schritte von allen Verfahren benutzt werden.

1.3.2 Ablauf eines evolutionären Algorithmus

Gemein ist allen Verfahren, dass mögliche Kandidaten für die Lösung eines Problems in einem **Genom**[4] codiert werden müssen. Ebenso muss jedes Genom und somit die mögliche Lösung, die es repräsentiert, bewertet werden können. Diesen Wert bezeichnet man als „Fitness". Häufig auftretende Fitnessfunktionen sind z.B. die Negation der Kostenfunktion oder das negative Fehlermaß[5] des Kandidaten.

Außerdem brauchen alle Algorithmen eine Möglichkeit, die Population durch Mutation und/oder Rekombination zu verändern. Hierbei wird evtl.

[4] Welches am Ehesten einem biologischen haploidem Genom entspricht.

[5] Im Folgenden wird der Einfachheit halber häufig auf die Negation verzichtet.

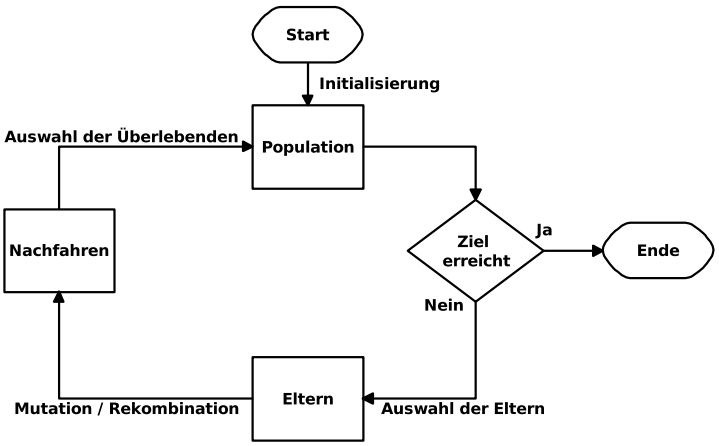


Abbildung 1.6: Allgemeines Schema eines Evolutionären Algorithmus

eine Auswahl getroffen, welche Individuen zur Fortpflanzung oder Mutation herangezogen werden. Ebenso müssen alle Verfahren auswählen können, welche Individuen „überleben“ und somit die nächste Generation bilden.

Codierung

Bei EAs muss das Genom nicht direkt mit der Lösung des Problems übereinstimmen, sondern nur eine mögliche **Codierung** selbiger sein. Man spricht vom Genotyp (Darstellung der Lösung als Genom) und Phänotyp (tatsächliche Lösung), wobei die Darstellung insofern eindeutig sein muss, als dass jedes Genom nur genau einer Lösung entspricht. Es dürfen allerdings durchaus mehrere unterschiedliche Genome den gleichen Phänotyp besitzen. Die unterschiedlichen Ausprägungen eines Gens bezeichnet man als Allel (Beispiel aus der Biologie: Augenfarbe).

Durch die Wahl der richtigen Codierung, die eine Verbindung zwischen dem Problemraum und dem Suchraum darstellt, kann der Suchraum stark verkleinert werden. Ein Beispiel dazu wäre das Damenproblem.

Dabei geht es darum, 8 Damen so auf einem Schachbrett zu platzieren, dass keine Dame eine beliebige andere in einem Zug schlagen kann. Eine mögliche Codierung wäre ein 2-dimensionales, binäres Array der Größe 8x8 als Codierung des Schachbretts, in dem eine 0 oder 1 anzeigt, ob auf dem Feld eine Dame steht oder nicht. Diese Darstellung wäre allerdings äußerst

ungeschickt, da so mehr als 8 Damen platziert werden können und wir einen Suchraum der Größe 2^{64} erhalten. Geschickter wäre also ein Genom der Größe 8 zu wählen, in dem die Zahlen[6] 1-64 eingetragen werden können. Selbst wenn man nicht überprüft, ob ein Feld mehrfach besetzt ist, erhält man somit schon eine Reduktion des Suchraums auf $64^8 = 2^{48}$. Codiert man zusätzlich noch das Wissen, dass alle Damen gleich sind und dass somit ihre Reihenfolge keine Rolle spielt, sowie, dass in jeder Zeile und Spalte nur eine Dame stehen darf bzw. genau eine Dame stehen muss, so kann man das Problem deutlich vereinfachen. Dies geschieht, indem man die Position der Damen als Permutationen der Zahlen von 1 bis 8 ansieht. Dann steht z.B. (8,7,6,5,4,3,2,1) für die (ungültige) Lösung, in der die Damen in einer Diagonalen stehen (die erste Dame in Zeile 1, Spalte 8 (also h1), die Zweite in Zeile 2, Spalte 7 in usw.). So ergibt sich eine drastische Verkleinerung des Problems auf $8! = 40320 << 2^{16}$ Möglichkeiten.

Initialisierung

Nachdem eine passende Codierung gefunden ist, werden zufällig verschiedene Lösungen generiert, die dann die „Population“ bilden, wobei die Größe der Population in fast allen Verfahren in jeder Generation konstant ist. Die initiale Population muss noch keine guten, teilweise nicht einmal gültige Lösungen enthalten.

Auswahl der Eltern

Hier wird bestimmt, welche Gene ausgewählt werden, um sich zu **kreuzen** bzw. zu **mutieren**. Bei vielen Verfahren ist diese Auswahl zufallsgesteuert, aber abhängig von der Fitness. Je besser die Fitness, desto höher die Wahrscheinlichkeit ausgewählt zu werden. Häufig haben so auch schlechtere Individuen eine (geringe) Chance, ausgewählt zu werden, um eine größere Artenvielfalt zu erhalten und um somit zu verhindern, dass man zu schnell in einem lokalen Optimum feststeckt.

Mutation

Mutation in EAs ist eine Methode, Veränderungen, und somit evtl. Fortschritt, an der Population hervorzurufen. Sie ist stark abhängig von der gewählten Repräsentation und ändert ein zufällig ausgewähltes Allel eines Gens in dem Genom des Individuums. Beispiel:

$$[0|1|\boldsymbol{0}|0|1|0] \rightarrow [0|1|\boldsymbol{1}|0|1|0]$$

[6] Die Position auf dem Spielbrett

Bei der Mutation in EAs wird meist allerdings nicht das ursprüngliche Individuum verändert, sondern eine Kopie, sodass durch Mutation zwei Individuen entstehen.

Rekombination

Bei der Kreuzung bzw. Rekombination werden zwei (wobei mehr als zwei ebenfalls möglich wären) Genome ausgewählt, um ihre Gene zu „vermischen“ und zwei Nachfahren zu generieren. Auch hier ist die genaue Methode abhängig von der gewählten Repräsentation. Außerdem gibt es verschiedenste (teilweise zufallsgesteuerte) Verfahren, um zu wählen, welcher Elternteil welche Gene weiter gibt. Ein Beispiel wäre es, das Genom in der Mitte zu teilen und dann jeweils die erste Hälfte des Einen an die zweite Hälfte des Anderen zu hängen (siehe Beispiel):

$$\begin{array}{ccc} [1|1|0|0|0|1] & & [0|1|0|0|0|1] \\ + & \rightarrow & \text{und} \\ [0|1|0|1|0|1] & & [1|1|0|0|0|1] \end{array}$$

Hierbei ergeben sich also vier Individuen, von denen zwei neue Eigenschaften besitzen.

Auswahl der Überlebenden

Die durch Mutation und/oder Kreuzung vergrößerte Population muss für den nächsten Durchlauf wieder auf die ursprüngliche Größe reduziert werden. Hierfür werden die Besten zum Überleben ausgewählt. Dies geschieht häufig ebenfalls zufallsgesteuert, abhängig von der jeweiligen Fitness. Bei Algorithmen, die direkt die Stärksten auswählen, sollte darauf geachtet werden, dass möglichst keine Individuen mehrfach vorkommen, da sonst wieder die Gefahr zu groß wäre, nur lokal zu suchen.

Terminierung

Da bei Optimierungsproblemen häufig nicht „die perfekte Lösung“ erreicht werden kann oder wird, terminieren EAs meist vorher. Mögliche Abbruchbedingungen sind:

- Erreichen einer ausreichend guten Lösung
- Keine oder nur geringe Veränderung seit der letzten Generation / den letzten Generationen

- Erreichen einer bestimmten Generationsanzahl
- Überschreiten einer bestimmten Berechnungsdauer

1.4 EANT2

„Evolutionary Acquisition of Neural Topologies“ (EANT) wurde von Kassahun und Sommer [Kas06] [KS05] entwickelt und von Siebel zu EANT2 weiterentwickelt [SK06]. EANT ist eine Methode zur Entwicklung künstlicher Neuronaler Netze.

Ähnlich wie in der Natur funktioniert die Lösungssuche mit EANT in zwei Phasen: Entwicklung der Spezies und Anpassung eines Individuums während seiner Lebenszeit. In der einen Phase werden neue Topologien der Netze (vergleichbar mit der Entwicklung einer neuen Spezies) entwickelt, deren Gewichte dann in der zweiten Phase verbessert werden, was mit der Anpassung (dem Lernen) eines Individuums an seine Umwelt verglichen werden kann. Einen Überblick über den Ablauf von EANT2 gibt der Pseudocode in Abschnitt 1.4.2 auf Seite 17.

Als Genom wird ein **Lineares Genom** eingesetzt, welches eine Auswertung des Neuronalen Netzes ohne Decodierung des Genoms erlaubt oder das neue **Binäre Genom** [SJS09]. Dieses ermöglicht es, das Neuronale Netz als compilierten Code auszuführen und somit zu beschleunigen.

1.4.1 Das Lineare Genom

EANT2 benutzt ein Lineares Genom variabler Länge, um die generierten Netze zu speichern. Dieses Lineare Genom ermöglicht eine Auswertung des dargestellten Neuronalen Netzes, ohne dass das Genom erst in ein neuronales Netz transformiert werden muss. Somit sind in diesem Fall Genotyp und Phänotyp identisch.

Jedes Gen in EANT entspricht entweder einem Eingangssignal, einem Neuron oder einer vorwärts oder rückwärts gerichteten Verbindung (forward bzw. recurrent Jumper), wobei rückwärts gerichtet bedeutet, dass der Ausgang eines Neurons an ein Neuron derselben Schicht oder einer früheren Schicht geleitet wird.

In jedem Gen wird der Typ des Knotens, sein synaptisches Gewicht und sein vorheriger Wert (für die Auswertung rückwärtsgerichteter Verbindungen) gespeichert. Neuronen speichern zusätzlich noch die Anzahl an Eingängen sowie ihre globalen IDs, die auch in den von ihnen ausgehenden **Jumper-Knoten** gespeichert werden, um deren Ursprung zu ermitteln. Beispiel siehe Abbildung 1.7.

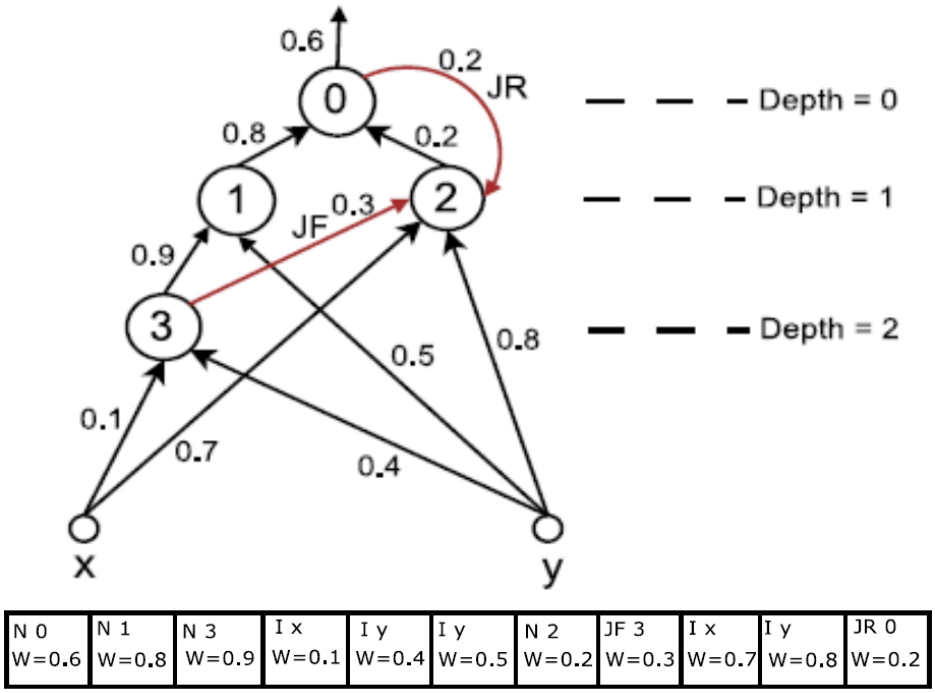


Abbildung 1.7: Beispiel eines Linearen Genoms (Grafik aus [Kas06])

Auswertung des Linearen Genoms

Ein großer Vorteil der gewählten Repräsentation eines neuronalen Netzes ist, dass das Lineare Genom direkt ausgewertet werden kann, ohne vorher transformiert werden zu müssen. Dies geschieht mittels eines Stapelspeichers durch Anwendung folgender Regeln, während das Genom von Rechts nach Links durchlaufen wird:

- Ist der aktuelle Knoten ein *Eingabesignal*, so wird sein aktueller Wert und sein Gewicht auf den Stack gelegt.
- Ist der aktuelle Knoten ein *Neuron* mit n Eingängen, so werden die obersten n Wert-/Gewicht-Paare (a, w) vom Stapel genommen, dann wird das Neuron nach Formel 1.1 ausgewertet, wobei $f(.)$ die gewählte Aktivierungsfunktion ist. Danach wird das Ergebnis O auf den Stapel geschrieben.

$$O = f\left(\sum_{i=1}^{n} w_i a_i\right) \tag{1.1}$$

- Ist der aktuelle Knoten ein ***Recurrent Jumper***, wird das gespeicherte Ergebnis der letzten Berechnung des Neurons, dessen globale ID mit dem Jumper übereinstimmt, gelesen und dieser Wert sowie das Gewicht des Jumpers auf den Stapel gelegt.

- Ist der aktuelle Knoten ein ***Forward Jumper***, wird das Subnetzwerk, welches von dem Neuron ausgeht, dessen ID mit dem Jumper übereinstimmt, kopiert und zuerst dieses nach denselben Regeln ausgewertet und dann das Ergebnis der Auswertung auf den Stapel geschrieben.

Wenn das Lineare Genom vollständig ausgewertet wurde, befinden sich noch so viele Ergebnisse auf dem Stapel, wie das Neuronale Netz Ausgänge hat. Diese Werte sind das Ergebnis des Netzes. Ein Beispiel für die Auswertung des linearen Genoms ohne Decodierung sieht man in Abbildung 1.8, wo das Netz aus Abbildung 1.7 ausgewertet wird.

Die Auswertung des linearen Genoms entspricht einer Auswertung eines decodierten Netzwerkes, wobei die Ausgabe des Neurons i ermittelt wird durch:

$$O_i(t) = f\left(\sum_{j=1}^{n_f} w_{ij}a_j(t) + \sum_{k=n_f+1}^{n} w_{ik}a_k(t-1)\right) \tag{1.2}$$

Wobei f eine Aktivierungsfunktion (siehe 1.2.3), n die Anzahl der Eingänge des Neurons, n_f die Anzahl der vorwärtsgerichteten Eingänge und $n - n_f$ die Anzahl der rückwärtsgerichteten Eingänge sind.

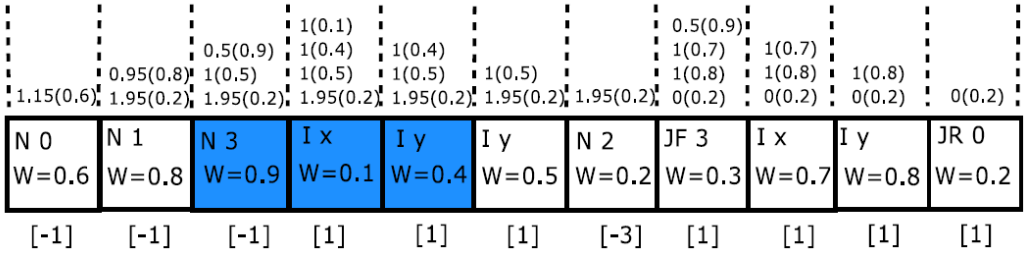


Abbildung 1.8: Auswertung eines Linearen Genoms (Grafik aus [KS05])

Zu Abbildung 1.8: An den Eingängen x und y liegt jeweils 1 an. Die Zahlen über dem Genom zeigen den Zustand des Speichers an, wobei die Zahlen in Klammern die Gewichte angeben. Hervorgehoben ist das Subnetz, das für die Auswertung des Forward Jumpers kopiert und extra ausgewertet werden

muss. Die Auswertung erfolgt von Rechts nach Links nach den Regeln aus 1.4.1. Als Aktivierungsfunktion fungiert hier die Identität. Die Zahlen in den eckigen Klammern geben Aufschluss über die Differenz der Anzahl der Ein- und Ausgänge eines Knotens. Da jeder Knoten genau einen Ausgang besitzt, steht [1] für einen Knoten ohne Eingang, [0] für ein Neuron mit genau einem Eingang usw. So ergibt sich, dass die Summe der Werte in einem vollständigen Netz oder Subnetz genau die Anzahl der Ausgänge ergibt.

Algorithmus 1 Pseudocode EANT2-Ablauf

```
procedure EANT2
    Generiere initiale Population (Initialisierung)
    Optimiere Parameter mittels CMA-ES (Exploitation)
    while Stoppkriterium nicht erreicht do
        Vergrößere Population mittels Mutation (Variation)
        Optimiere Parameter mittels CMA-ES (Exploitation)
        Wähle überlebende Population aus (Selektion)
    end while
end procedure
```

1.4.2 Initialisierung

Um mit dem Evolutionären Algorithmus beginnen zu können, muss zuerst eine Anfangspopulation der wählbaren Größe λ an Strukturen erschaffen werden. Hierfür generiert EANT zufällig Netze mit passender Anzahl an Ein- und Ausgängen. Dabei hat der Anwender die Wahl, wie viele Individuen erschaffen werden sollen, ob und wie viele versteckte Knoten in den anfänglichen Netzen enthalten sein sollen und ob die Netze vollständig verknüpft sein sollen. Dabei versucht EANT (soweit möglich) unterschiedliche Netze zu erstellen.

1.4.3 Variation

In EANT ist der einzige Operator zur strukturellen Variation die Mutation. Dabei wird für jedes Neuron eine Zufallszahl aus dem Intervall [0,1] ermittelt. Ist sie kleiner als die Mutationsrate p_m, die typischerweise zwischen 0.05 und 0.1 liegt, so wird dieses Neuron[7] mutiert. Wurde ein Knoten zur Mutation ausgewählt, so wird zufällig und gleichverteilt eine der folgenden Mutationen ausgeführt:

[7] Bzw. die dazugehörigen Verbindungen.

- Hinzufügen oder Entfernen eines Recurrent Jumpers

- Hinzufügen oder Entfernen eines Forward Jumpers

- Hinzufügen oder Entfernen eines Subnetzwerks

Dabei werden die Gewichte der aus dem neuen Netz herausführenden Verbindungen bzw. das Gewicht des neuen Jumpers initial auf 0 gesetzt, um die Leistung des Neuronalen Netzes nicht negativ zu beeinflussen. In Abbildung 1.9 sieht man, wie durch Mutation die Verbindung vom Eingangssignal zu Neuron 1 durch eine selbstrekurrente Verbindung ersetzt wird.

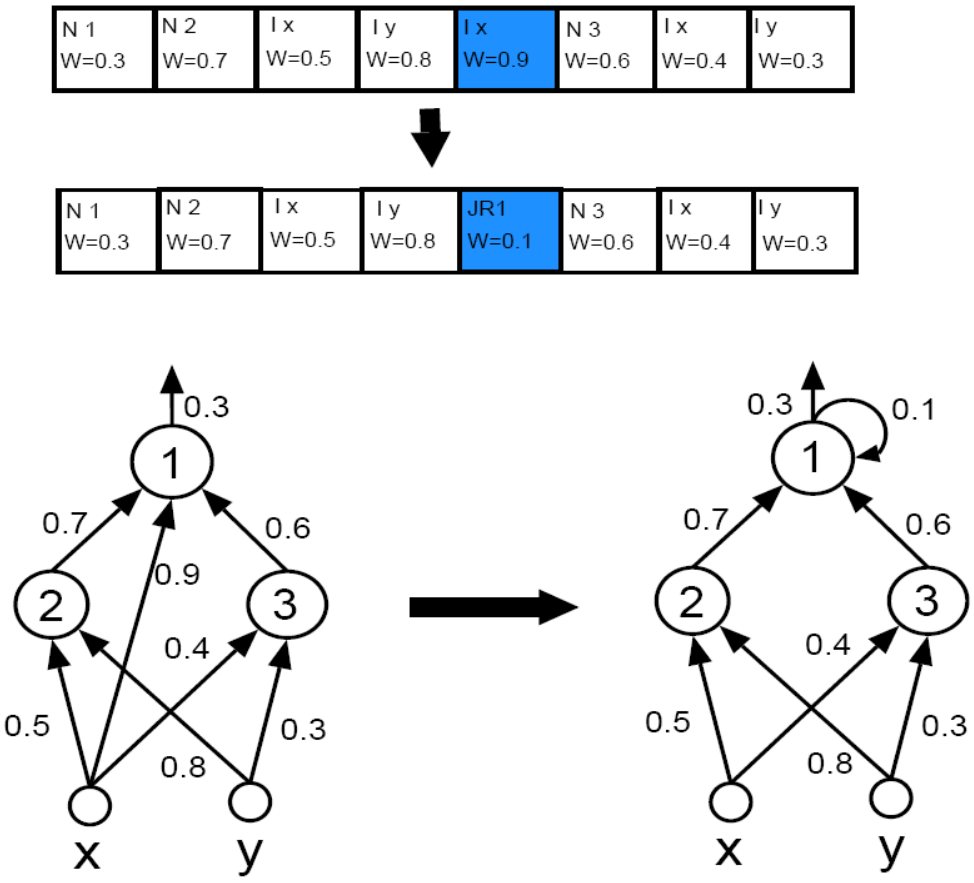


Abbildung 1.9: Beispiel für Mutation in EANT (Grafik aus [Kas06])

1.4.4 Exploitation

Ebenso wichtig wie die strukturelle Optimierung der Netze (**Exploration**) ist die Optimierung der Gewichte (**Exploitation**), um den Wert einer Struktur ermitteln zu können. Hierfür wird in EANT2 CMAES [HansOste01] verwendet. Zur Optimierung bildet jede Struktur mehrere Individuen, von denen allerdings nur ein Teil „überleben“ darf, um zu verhindern, dass sich eine einzelne Struktur zu stark ausbreitet und alle anderen verdrängt.

Zwischen je zwei Strukturgenerationen steht dabei immer die Exploitation (oder auch Parameteroptimierung), bei der versucht wird, das optimale Ergebnis für jede Struktur zu ermitteln. Dies geschieht bevor eine weitere Strukturgeneration entsteht, die häufig mit größeren Netzen verbunden ist und somit den Suchraum vergrößert. Dabei bedeutet jedes zusätzliche Gewicht eine weitere Dimension des Suchraums, somit ist es klar, dass es sinnvoll ist, die Optimierung einer bestehenden Struktur durchzuführen, bevor die Netze evtl. unnötig groß und komplex werden.

1.4.5 CMAES

CMAES wird in EANT2 eingesetzt, um die Gewichte der Netzwerke zu optimieren. Wie in [SK06] zu sehen ist, konnte durch den Einsatz von CMAES EANT signifikant verbessert werden. Um zwischen der Version mit CMA-ES und der ohne zu unterscheiden, wurde EANT2 als Bezeichnung für die neue Version eingeführt.

CMAES (Covariance Matrix Adaptation Evolution Strategy) ist ein sehr gutes, evolutionäres Minimierungsverfahren zur Lösung mehrdimensionaler, multimodaler Probleme. Für eine ausführliche Vorstellung von CMAES siehe [HO01].

Ausgehend von einer zu minimierenden Funktion[8] $f : \mathbb{R}^n \to \mathbb{R}$ mit n Parametern[9] wird eine Stichprobenmenge[10] mit Größe λ der multivariaten Normalverteilung $N(x_o, I_{n\times n})$ folgend, mit n-ter Einheitsmatrix $I_{n\times n}$, für einen Startpunkt x_0 gezogen. Mittels der gezogenen Stichprobenelemente ändert sich sowohl der Mittelwert als auch die Kovarianzmatrix der neuen Normalverteilung ($N(m_1, C_1)$), mit welcher die nächste Stichprobe gezogen wird, bis ein Abschlusskriterium erfüllt ist.

Der neue Mittelwert errechnet sich aus den besten μ Stichproben der vorherigen (i-ten) Generation. Diese werden sortiert, sodass $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_\mu)$ ist. Der neue Mittelwert m_{i+1} ergibt sich aus

[8] hier die negative der Fitnessfuntkion

[9] hier die Gewichte des neuronalen Netzes

[10] also Population

$$m_{i+1} = \sum_{i=1}^{\mu} x_i w_i$$

wobei für die Gewichte $w_1 \geq w_2 \geq \cdots \geq w_\mu$ gilt, dass $\sum_{i=1}^{\mu} w_i = 1$ ist.

Die neue Kovarianzmatrix C_{i+1} ergibt sich aus der empirischen Kovarianzmatrix $C_\mu = \langle x \rangle_{sel} \langle x \rangle_{sel}^T$ mit $\langle x \rangle_{sel} = \sum_{i=1}^{\mu} x_i w_i$ der μ besten, gewichteten Stichproben der vorherigen Generation durch

$$C_{i+1} = (1 - c_{cov}) C_i + c_{cov} C_\mu \ ,$$

wobei der Adaptionskoeffizient $c_{cov} \in [0, 1]$ die Geschwindigkeit der Adaption festlegt.

1.4.6 Selektion

Zuerst werden für Parameteroptimierung angefertigte Kopien eines Netzwerks entfernt. Als Elterngeneration für die nächste Generation werden dann die λ (Größe der Anfangspopulation) besten unterschiedlichen Netzwerke ausgewählt, wobei die Auswahl aufgrund der Fitnesswerte erfolgt. Allerdings werden bei gleicher oder ähnlicher Fitness kleinere Netzwerke bevorzugt, so dass einem „Survival of the Fattest“ entgegengewirkt wird.

1.5 Verbesserung von EANT2

Im Folgenden sollen Möglichkeiten betrachtet werden, um EANT2 (Evolutionary Acquisition of Neural Topologies 2) zu verbessern. Eine Verbesserung von EANT2 wäre, wenn die Geschwindigkeit erhöht wird oder wenn eine bessere Fitness erreicht wird. Um diese Verbesserungen zu erreichen werden mehrere Ansatzpunkte betrachtet:

- die Hauptschleife von EANT2
- die Parameteroptimierung

1.5.1 Verbesserungen der Hauptschleife

Innerhalb der Hauptschleife von EANT2 werden neue Neuronale Netzwerke generiert, die Parameteroptimierung angestoßen und schließlich die nächste Generation selektiert.

Da die Selektion und die Mutation im Vergleich zur Parameteroptimierung (weniger als eine Sekunde gegenüber häufig deutlich mehr als eine Stunde) keine Verbesserung der Geschwindigkeit benötigen, bleibt als Ansatz für die Beschleunigung innerhalb der Hauptschleife nur die Auswahl der zu optimierenden Individuen. In Kapitel 4.1 wird untersucht, inwieweit eine Verbesserung durch Weglassen von Mehrfachoptimierungen möglich ist.

Um bessere Auswertungen vornehmen zu können, wurde eine verbesserte Protokollierung der Vorgänge innerhalb von EANT2 eingebaut, die in Kapitel 4.2 vorgestellt wird.

Die Funktionalität in EANT2, Optionen aus einer XML-Datei zu lesen, war nur auf die verwendeten Aktivierungsfunktionen beschränkt. Dies Funktionalität wurde stark erweitert, um das von Bötel und Siebel in [SBS09] **Pruning** ebenso wie das von Jordt programmierte Binäre Genom [SJS09] ein oder aus schalten zu können. Außerdem können nun hier diverse Einstellungen für die Parameteroptimierung vorgenommen werden, ohne das Programm neu compilieren oder den Quellcode ändern zu müssen .

1.5.2 Verbesserungen der Parameteroptimierung

Die Parameteroptimierung ist einer der wichtigsten Punkte innerhalb von EANT2, da die besten Strukturen ohne richtige Wahl der Parameter keine brauchbaren Ergebnisse generieren können. Da die Parameteroptimierung gleichzeitig auch der zeitintensivste Vorgang ist, besteht hier auch das größte Potential für Geschwindigkeitsgewinne.

Eine Möglichkeit ist die Reduktion der Anzahl der Fitnessfunktionsevaluationen mittels Approximation der Fitnessfunktion durch Lokale Meta-Modelle. Dies wird in Kapitel 8 untersucht.

Da laut [HK04] die Standard-Populationsgrößen von CMAES nicht immer die optimalen Ergebnisse erreichen, werden in Kapitel 7 größere Populationen untersucht.

Ob der Austausch von CMAES durch eine andere Optimierungsmethode (hier Partikel-Schwarm-Optimierung) EANT2 verbessern kann wurde in Kapitel 6 untersucht.

1.6 Probleme

Damit EANT2 versuchen kann, ein Problem zu lösen, bedarf es prinzipiell nur einer Funktion, die die Lösungskandidaten mit einem Fitnesswert bewertet, sowie einer Repräsentation der Anfragen und Ausgaben als (Fließkomma-) Zahlen. Im Folgenden werden die Probleme kurz vorgestellt, auf die EANT2 angewandt wurde und die auch in den weiteren Kapiteln als Grundlage zur Bewertung der Verbesserungen dienen.

1.6.1 Visual Servoing

Eines der Probleme, auf das EANT2 angewandt wird, ist das Visual-Servoing-Problem. Hierbei soll ein Roboterarm durch Visuelles Feedback eine vorher bestimmte Teachposition erreichen. Der Roboterarm wird in die Teachposition bewegt (siehe Abbildung 1.10 (b)). Dann wird eine Aufnahme von einem Objekt (Abbildung 1.10 (a)), das mit vier Markierungen versehen ist, gemacht. Nun wird der Roboter in eine andere, ihm unbekannte Position bewegt und ein weiteres Bild gemacht. Mit Hilfe dieses Bildes soll der Roboterregler Steuerbefehle ermitteln, die ihn wieder zurück in seine Ausgangslage bringen sollen. In dieser Arbeit wird ein Roboterarm mit 3 Freiheitsgraden.

Hierfür werden aus dem Bild zehn Bildmerkmale extrahiert, aus denen die gewünschte Position berechnet werden soll. Diese Merkmale sind die Abstände zwischen den jeweils gegenüberliegenden Markierungen, aus denen die Entfernung zum Objekt hergeleitet werden kann. Die anderen acht Merkmale sind die Abweichungen der 2D-Koordinaten der Markierungen der aktuellen Position von der Teachposition.

Das Ergebnis ist die neue Position, die der Roboterarm annehmen soll, um die Teachposition zu erreichen. Aus diesen werden mittels Inverser Kinematik Stellwerte für die Gelenke des Roboters ermittelt. Einen vergleichenden Überblick über verschiedene Modelle und Regler für das Visual-Servoing-Problem gibt Peters in [Pet09].

Traditioneller Regler

Möchte man Positionen im Bereich der Robotik beschreiben, stellt man fest, dass es eine Vielzahl von Koordinatensystemen gibt, die als Grundlage zur Beschreibung dienen. Abbildung 1.11 zeigt einen Überblick über die verschiedenen Koordinatensysteme. Dabei bezeichnet W das unbewegliche Weltkoordinatensystem, während sich die Koordinatensysteme T (Toolskoordinaten) und C (Kamerakoordinaten) mit dem Arm zusammen bewegen.

(a) Objektmarkierungen (b) Aufnahme der Teachposition

Abbildung 1.10: Aufbau Visual-Servoing-Problem. Fotos aus [Pet09]

Die Sensor- und Bildkoordinatensysteme (S bzw. I) sind hierbei nur zweidimensional.

Um zu verdeutlichen, aus welchem Koordinatensystem die Koordinaten stammen, wird ihnen der jeweilige Buchstabe des Koordinatensystems als hochgestellter Index vorangestellt. So bezeichnet z.B. ${}^{S}x$ die X-Koordinate im Sensorkoordinatensystem.

Eine Möglichkeit, das Visual-Servoing-Problem zu lösen, ist der sogenannte Traditionelle Regler, der ausführlich in [Sie99] vorgestellt wird.

Ziel ist es die Stellgröße $u_n \in \mathbb{R}^3$ zu finden, die den Roboterarm in die richtige Pose bringt, so dass der momentane Bildfehler $||\Delta y_n||_\infty$, also die Abweichung zur Teachposition $y_n^\star$, möglichst 0 wird. Dabei bezeichnet $n \in \mathbb{N}_0$ den Momentanen Zeitpunkt.

Dafür benötigt man die Bildjacobimatrix J_n

$$J_n = \begin{pmatrix} -\frac{f}{{}^{C}z} & 0 & \frac{{}^{S}x}{{}^{C}z} \\ 0 & -\frac{f}{{}^{C}z_1} & \frac{{}^{S}y_1}{{}^{C}z_1} \end{pmatrix} \quad , \tag{1.3}$$

die man für kleine $||u||_2$ durch folgendes lineares Modell approximieren kann:

$$y_{n+1} - y_n \approx J_n u \quad = \begin{pmatrix} -\frac{f}{{}^{C}z_1} & 0 & \frac{{}^{S}x_1}{{}^{C}z_1} \\ 0 & -\frac{f}{{}^{C}z_1} & \frac{{}^{S}y_1}{{}^{C}z_1} \\ \vdots & \vdots & \vdots \\ -\frac{f}{{}^{C}z_M} & 0 & \frac{{}^{S}x_M}{{}^{C}z_M} \\ 0 & -\frac{f}{{}^{C}z_M} & \frac{{}^{S}y_M}{{}^{C}z_M} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix},$$

wobei $M = 4$ die Anzahl der Bildmarkierungen ist.

Für diese Matrix bildet man die Pseudoinverse $J_n^+ \in \mathbb{R}^{3\times 8}$

$$J_n^+ = (J_n^T J_n)^{-1} J_n^T,$$

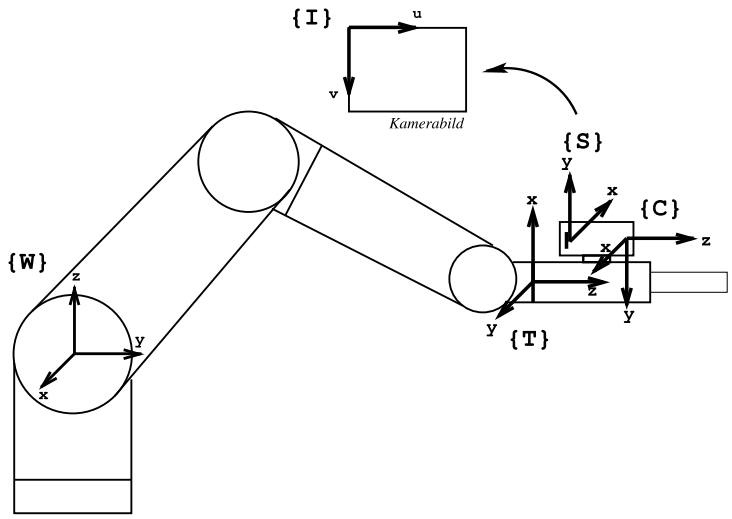


Abbildung 1.11: Verschiedene Koordinatensysteme eines Roboterarms. Aus [Sie99]

mit deren Hilfe man den sogenannten vollen Gauß-Newton-Schritt zur Minimierung des Bildfehlers durchführt:

$$\Delta u_n = J_n^+(-\Delta y_n).$$

In der Regel wird allerdings nicht der volle Schritt ausgeführt, sondern man verwendet einen Dämpfungsfaktor $k \in \mathbb{R}$ mit $0 < k \leq 1$ und erhält dann

$$u_n = k \cdot \Delta u_n = k \cdot J_n^+(-\Delta y_n).$$

Visual-Servoing in EANT2

Es ist natürlich wenig praktikabel die EANT2 Versuche zum Visual-Servoing an einem echten Roboterarm auszuführen. Im Verlauf eines EANT2-Durchlaufs werden tausende von Strukturen generiert, die dann für die Parameteroptimierung nochmal in größeren Populationen optimiert werden, wobei sie z.B. 5000 mal jeweils 1023 verschiedene Posen ansteuern müssen. Deshalb werden die Eingabedaten und Ergebnisse nur berechnet.

Die Berechnung der Fitness wird in [SS07] ausführlich beschrieben und hier kurz zusammengefasst. Dabei werden insgesamt 1023 Startposen ausgewertet, die 29 verschiedene Teachposen als Ziel haben. Als Grundlage zur Berechnung der Fitness dient ein Vektor $y_n \in \mathbb{R}^8$, der die Posiotion der vier Bildmerkmale nach der simulierten Roboterbewegung enthält. Dieser wird

mit den Soll-Bildmerkmalen $y_i^\star \in \mathbb{R}^8$ der Teachpose verglichen. Die Fitnessfunktion des Netzes N lautet somit

$$F(N) := \sqrt{\frac{1}{1023} \sum_{i=1}^{1023} \left(\frac{1}{4} \sum_{j=1}^{4} d_j(y_i)^2 + b(y_i) \right)}.$$

Dabei gibt $b(y_i)$ einen Bestrafungsterm an, der Werte größer Null annimmt, wenn der Roboter das Objekt durch seine Bewegung aus der Sicht verliert oder das Objekt berühren würde. Ansonsten gilt $b(y_i) = 0$. Die Distanz $d_j(y_i)$ berechnet sich durch

$$d_j(y_i) := \|(y_i^\star)_{2j-1,2j} - (y_i)_{2j-1,2j}\|_2 \ ,$$

wobei $(y)_{2j-1,2j}$ aus der $2j-1$-ten und der $2j$-ten Position des Vektors y besteht.

1.6.2 Kantenerkennung

Bei der Kantenerkennung handelt es sich um eine häufig auftretende Form der Bildklassifikation. Praktisch jedes Bildbearbeitungsprogramm bietet die Möglichkeit, Kantenbilder aus den eigenen Bildern zu generieren. In Abbildung 1.13 sieht man ein Beispiel für ein Kantenbild des berühmten Lena-Bildes (Abbildung 1.12). Dieses Kantenbild wurde mittels „The GIMP 2.2“ und dem Sobel-Kantendetektor (siehe 1.6.2) erstellt.

Abbildung 1.12: Lena

Abbildung 1.13: Kantenbild (Sobel)

Mathematisch gesehen handelt es sich bei einer Kante um einen Wendepunkt der Helligkeitsfunktion[11]. Da die Helligkeitsfunktion allerdings nicht als solche vorliegt, sondern nur ihre Ergebnisse an diskreten Punkten (Pixeln), können diese Wendepunkte nur geschätzt werden.

Auch die genaue Position der Kante kann nicht problemlos bestimmt werden, da eine Kante häufig zwischen zwei Pixeln liegt und nicht auf einem. Somit besteht für einen Kantendetektor die Möglichkeit entweder ein oder zwei Pixel als Kante zu markieren, wenn man nicht ein um ein halbes Pixel verschobenes und um ein Pixel verkleinertes Kantenbild erhalten will.

Die Entwicklung eines Kantendetektors durch EANT2 ist (einschließlich Trainingsdaten und Berechnung des Fitnesswerts) in Kapitel 2 beschrieben. Im Folgenden werden zwei der gängigsten „herkömmlichen“ Kantendetektoren vorgestellt.

[11] Also eine Nullstelle der zweiten bzw. ein Extrempunkt der ersten Ableitung der Helligkeitsfunktion

Sobel

Zu den bekanntesten Kantendetektoren gehören der Canny-Algorithmus und der Sobel-Operator, die hier und in Abschnitt 1.6.2 kurz vorgestellt werden.

Die Idee bei Sobel ist, das Bild zeilenweise mit zwei 3x3 Masken abzutasten, die ein Maß angeben sollen, ob ein Wendepunkt in Richtung der Spalten (mittels $\mathbf{H}_c$) oder in Zeilenrichtung ($\mathbf{H}_r$) vorliegt. Die Masken, mit denen die Ableitung approximiert wird lauten:

$$\mathbf{H}_c = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & \mathbf{0} & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}_r = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 0 & \mathbf{0} & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Diese Masken werden für die beiden $\alpha \in \{c, r\}$ mittels

$$g_\alpha(c, r) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} g(c+i, r+j)(\mathbf{H}_\alpha)_{j,i}$$

eingesetzt. Hier ist g ein digitales Bild gegeben als Abbildung $\mathcal{S} \times \mathcal{Z} \xrightarrow{g} \mathcal{I}$, wobei $\mathcal{S} := \{0, 1, \ldots, N_\mathcal{S} - 1\}$ für die Spalten, $\mathcal{Z} := \{0, 1, \ldots, N_\mathcal{Z} - 1\}$ für die Zeilen, $N_\mathcal{S}$ und $N_\mathcal{Z}$ für die Spalten- bzw. Zeilenanzahl und $\mathcal{I} := \{0, 1, \ldots, N_\mathcal{I} - 1\}$ für die $N_\mathcal{I}$ möglichen Grauwerte des Bildes stehen. Weiterhin steht $(\mathbf{H}_\alpha)_{j,i}$ für das Element aus $\mathbf{H}_\alpha$, das in Zeile j in Spalte i steht, wobei $(0, 0)$ die dick gedruckte Zahl angibt.

Die Ergebnisse werden dann mittels

$$\sqrt{\sum_{\alpha \in \{c,r\}} g_\alpha^2(c, r)}$$

oder durch

$$\sum_{\alpha \in \{c,r\}} \left| g_\alpha^2(c, r) \right|$$

zusammengefasst.

Canny

Verfahren, die auf Ableitungen basieren, sind sehr störungsempfindlich gegenüber Rauschen. Eine Möglichkeit, diese Empfindlichkeit zu verringern,

ist, die Abtastmaske zu vergrößern oder durch einen Filter das Rauschen zu glätten. Letzteren Weg geht der Canny-Algorithmus, der einen Gaussfilter verwendet. Dabei wird eine Gaussfunktion mit $\sigma = 1/2\sqrt{2}$ beispielsweise durch folgende Maske approximiert [Hog06]:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & \mathbf{4} & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Nach der Glättung wird ein Ableitungs-Kantendetektor, z. B. Sobel, benutzt.

Um binäre Daten (Kante ja oder nein) zu erhalten, wird ein **Hysterese** genanntes Verfahren verwendet. Hierfür benötigt man zwei Schwellenwerte. Pixel, deren Werte über dem oberen Schwellenwert liegen, werden als Kanten gewertet, Pixel unter der unteren Schwelle werden als „keine Kante“ gewertet. Pixel, deren Werte zwischen den beiden Schwellenwerten liegen, werden als Kante gewertet, wenn es eine Verbindung über Pixel mit Werten über der unteren Schwelle zu einem Pixel mit einem Wert über der oberen Schwelle gibt.

Es gibt also 3 Parameter, die das Ergebnis des Kantenfilters stark beeinflussen. Die Abbildungen 1.14(a)-(c) zeigen den Canny-Algorithmus angewandt auf das Lena-Bild auf Seite 27. Dabei wurde bei Bild (a) $\sigma = 1$ gewählt und auf eine Kantenausdünnung verzichtet. Bei Bild (b) ist ebenfalls $\sigma = 1$, für die untere Schwelle wurde $0,05$ und für die obere Schwelle $0,5$ gewählt. Bei (c) ist $\sigma = 2$, die untere Schwelle liegt bei $0,01$, während die obere Schwelle bei $0,3$ liegt.

(a)

(b)

(c)

Abbildung 1.14: Kantenbilder des Canny-Algorithmus mit verschiedenen Parametern

1.6.3 Helikopterflug

Das Problem, einen Helikopter in einer wechselhaften Umwelt schweben zu lassen, wird im Kapitel 3 näher vorgestellt. Auf dieses Problem wird in anderen Kapiteln selten Bezug genommen.

Kapitel 2

Kantendetektion mittels EANT2

In [Gru07] wurde bereits gezeigt, dass es möglich ist, mit EANT2 einen Kantendetektor von Null auf generieren zu lassen. Um diese Ergebnisse zu verifizieren, werden 5 weitere Durchläufe von EANT2 jeweils mit 3x3- und 5x5-Masken durchgeführt. Da es sich bei der Kantenerkennung um ein binäres Klassifikationsproblem handelt, wird ein Schwellenwert für die Ausgabe der Netzwerke eingeführt, bei dessen Überschreiten von einer Kante ausgegangen wird und bei Nicht-Überschreiten von keiner Kante.

2.1 Versuchsaufbau

Als Grundlage für die Bewertung der Netzwerke wird ein Bilderpaar bestehend aus dem Trainingsbild (Abbildung 2.1)und der „Ground-Truth“ (Abbildung 2.2) benutzt.

Abbildung 2.1: Trainingsbild

Abbildung 2.2: Ground-Truth

Dabei wird eine Maske der Größe 3x3 bzw. 5x5 zeilenweise über das Bild geführt und die Ausgabe des Netzwerks für die 9 bzw. 25 Eingaben berechnet. Diese Ausgabe wird dann mit der Ground-Truth für die Koordinate verglichen, die dem Mittelpunkt der Maske entspricht. Die Fitness eines Netzwerks

N ermittelt sich also aus

$$f(N) = \frac{\sum_{(x,y)} |G(x,y) - |a(x,y)||}{(b - 2 \cdot r)(h - 2 \cdot r)}$$

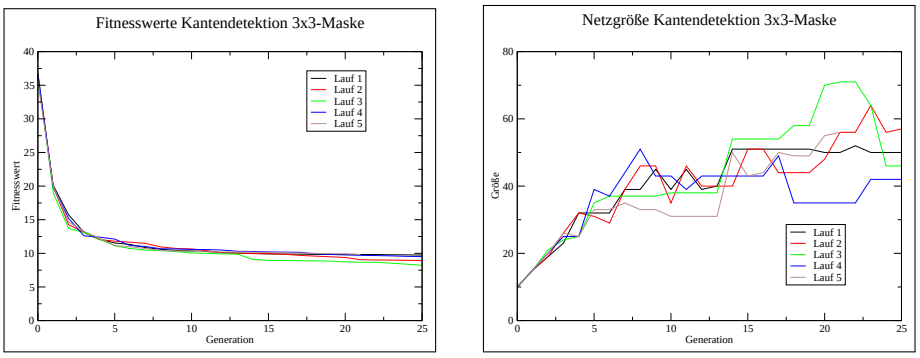
wobei $G(x,y)$ der Grauwert der Ground-Truth an der Stelle (x,y), $a(x,y)$ die Ausgabe des Netzwerks, b und h die Breite bzw. Höhe des Bildes und r die Breite des Rands ist. Der Rands ergibt sich durch die Größe der Maske. Somit ist für 3x3 $b = 1$ und für 5x5 $b = 2$. Das Trainingsbild hat eine Höhe und Breite von 64 Pixeln und 256 Grauwerte. Ein Fitnesswert von 0 wäre somit das optimale und ein Wert von 255 das schlechtest mögliche Ergebnis.

Die Strukturen werden mit 1500 CMAES-Iterationen optimiert.

2.2 Ergebnisse

Abbildung 2.3 zeigt die Fitnesswerte der generierten Netzwerke. Die Entwicklung der Fitnesswerte ist über alle Durchgänge sehr ähnlich und es findet etwa ab Generation 6 nur noch sehr wenig Verbesserung statt.

Die Größe der Netzwerke variiert unter den verschiedenen Läufen allerdings sehr stark. So ist bei Generation 20 die beste Struktur von Lauf 3 rund doppelt so groß, wie die von Lauf 4. Hier ist aber auch zu erkennen, das die Netzgröße auch deutlich kleiner werden kann.

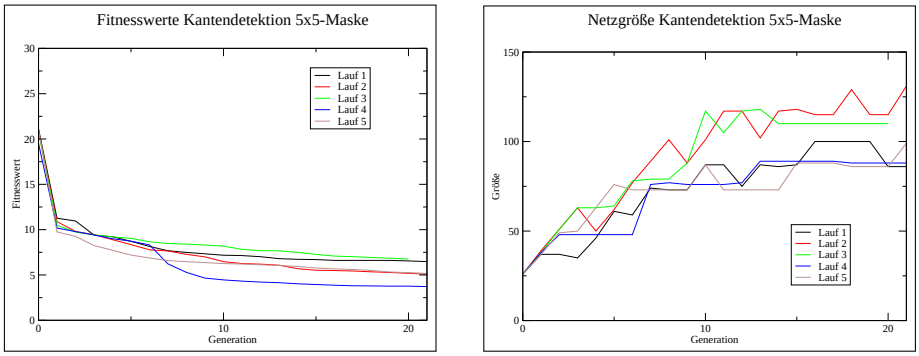


(a) Fitnesswerte (b) Größe

Abbildung 2.3: Fitnesswerte und Netzgröße Kantendetektion mit 3x3-Maske

Bei den Ergebnissen der 5x5-Experimente (Abbildung 2.4) kann man deutlich größere Schwankungen der einzelnen Läufe erkennen. Dies kann an der größeren Netzgröße liegen, da hierdurch einerseits natürlich größere Unterschiede entstehen können, andererseits aber vor allem die Parameteroptimierung durch die größere Dimension erschwert ist.

Die Größe der Netze ist allerdings bei den einzelnen Läufen ähnlicher als bei der 3x3-Maske. Auch kann hier nur selten eine Verkleinerung beobachtet werden. Dies liegt natürlich zum einen an der größeren Mindestgröße von 25 gegenüber 9 und zum anderen an der leicht geringeren Generationsanzahl.



(a) Fitnesswerte (b) Größe

Abbildung 2.4: Fitnesswerte und Netzgröße Kantendetektion mit 5x5-Maske

2.3 Vergleichstest

Da ein guter Fitnesswert auch durch ein „Auswendiglernen“ der Trainingsdaten erreicht werden kann, ist es wichtig, mit von diesen abweichenden Testdaten das Ergebnis zu überprüfen. Ebenso benötigt man für die Einschätzung der Qualität des Ergebnisses auch Vergleichsdaten. Hierfür wird der Sobel-Kantendetektor [SF68] benutzt. Außerdem werden noch die Ergebnisse des Canny-Algorithmus angegeben. Dieser liefert aber keine direkt vergleichbaren Ergebnisse, deshalb kann hier nur optisch verglichen werden.

2.3.1 Versuchsaufbau

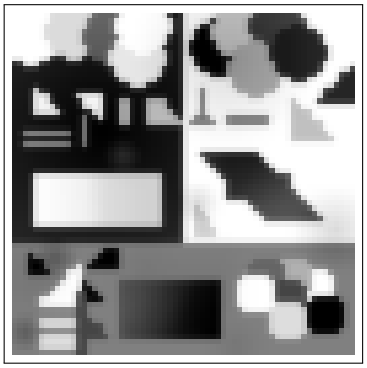
Da es sich bei der Kantendetektion um eine binäre Klassifikation handelt, müssen die Ergebnisse der Netzwerke und von Sobel von kontinuierlichen Werten in klare Ja/Nein-Aussagen umgewandelt werden. Hierfür bietet es

sich an, eine Schwelle einzuführen. Liegt das Ergebnis unter diesem Schwellenwert, so wird keine Kante detektiert, liegt das Ergebnis über dem Schwellenwert, so wird eine Kante detektiert.

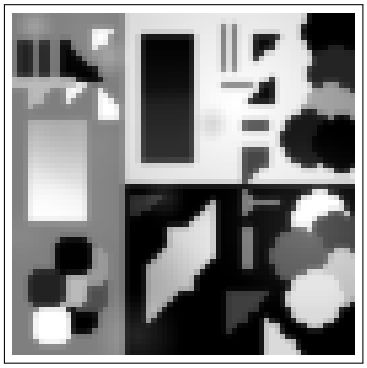
Für die generierten Netze wird dieser Schwellenwert auf 0,5 gesetzt, was bei 256 Grauwerten einem Wert von 128 entspricht. Für den Sobel-Operator wird hingegen ein Schwellenwert trainiert, der die Klassifikationsrate für die Trainingsdaten optimiert. Der ermittelte Schwellenwert für Sobel liegt bei 0,05.

Dasselbe Verfahren wird für Canny benutzt und somit werden für Canny optimale Ergebnisse auf den Trainingsdaten mit $\sigma = 0,1$ erreicht. Für die Schwellenwerte werden 0,061 bzw. 0,08 als optimal ermittelt.

So wird das beste Netzwerk mit 3x3 und 5x5 Maske ebenso wie Sobel und Canny auf 4 Testbilder (Abbildungen 2.5 (a)-(d)) angewandt.

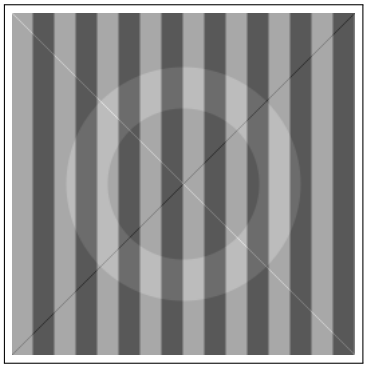


(a) Trainingsbild (vergrößert)



(b) Rotiertes, invertiertes Trainingsbild

(c) Testbild



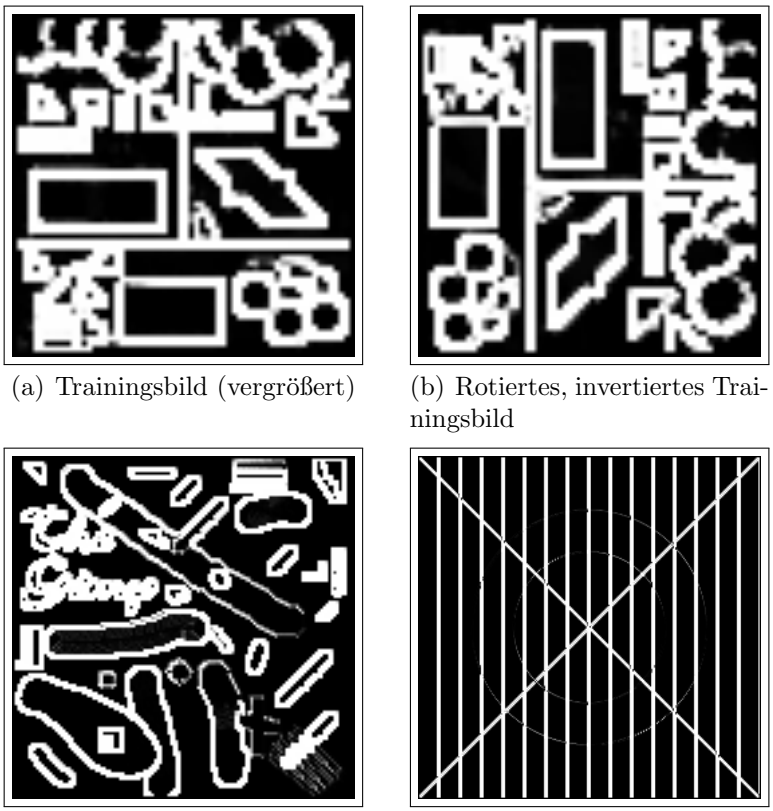
(d) Ledge (hier verkleinert)

Abbildung 2.5: Testbilder Kantendetektion

Bei Abbildung 2.5 (b) handelt es sich um das Trainingsbild, das rotiert und invertiert wurde. Bild (c) enthält mehrere Helligkeitsveränderungen, die

nicht als Kante erkannt werden dürfen, kontrastarme Kanten unten rechts sowie diverse andere Kanten. Bei (d) handelt es sich um das „Ledge“-Bild von Hans du Buf[1].

[1]Zur Verfügung gestellt unter http://w3.ualg.pt/ dubuf/pubdat/ledge/ledge.html



(a) Trainingsbild (vergrößert)

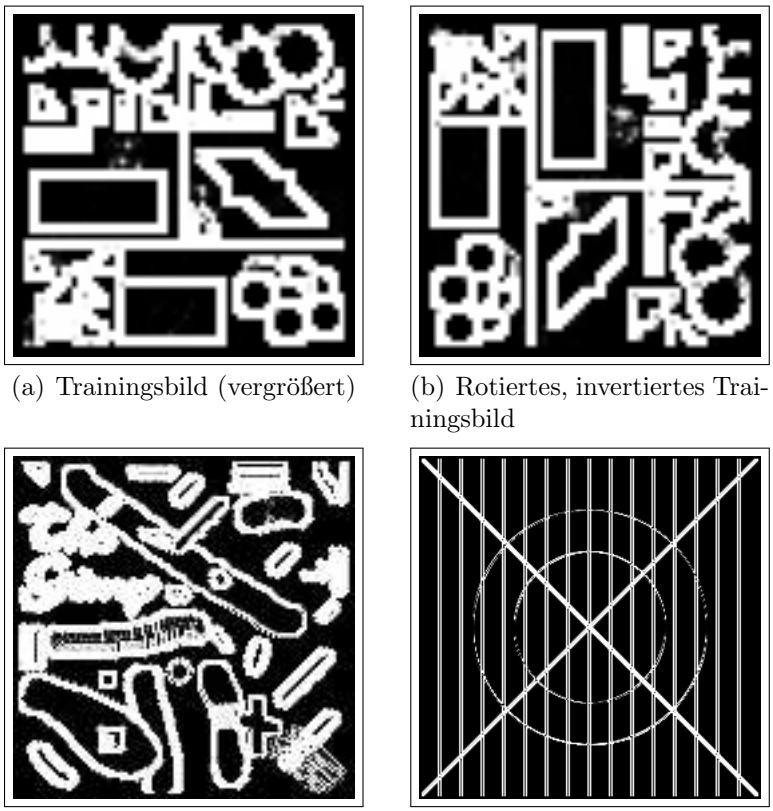
(b) Rotiertes, invertiertes Trainingsbild

(c) Testbild

(d) Ledge (hier verkleinert)

Abbildung 2.6: Ergebnisse EANT2 Kantendetektion 3x3-Maske

Die Ergebnisse der 3x3 Maske sind in Abbildung 2.6 zu sehen. Da die Ergebnisse für das Trainingsbild (a) und dem invertierten und rotiertem Bild (b) sehr ähnlich sind, kann man erkennen, dass das Netz gelernt hat, das Kanten rotations- und invertierungsinvariant sind. Bei dem Testbild ist zu erkennen, dass die Umrisse der Objekte meist gut getroffen sind. Die Übergänge wurden nicht als Kanten erkannt, allerdings wurden die kontrastarmen unteren rechten Kanten auch nicht als solche erkannt. Bei dem Ledge-Bild wurden, bis auf den Kreis, alle Kanten erkannt. Optisch scheint es bei diesem Detektor fast keine falsch positiven Kanten zu geben. Wie man in Tabelle 2.1 sehen kann, hat der 3x3-Detektor bei allen Testbildern eine bessere Erkennungsrate als Sobel.



(a) Trainingsbild (vergrößert)

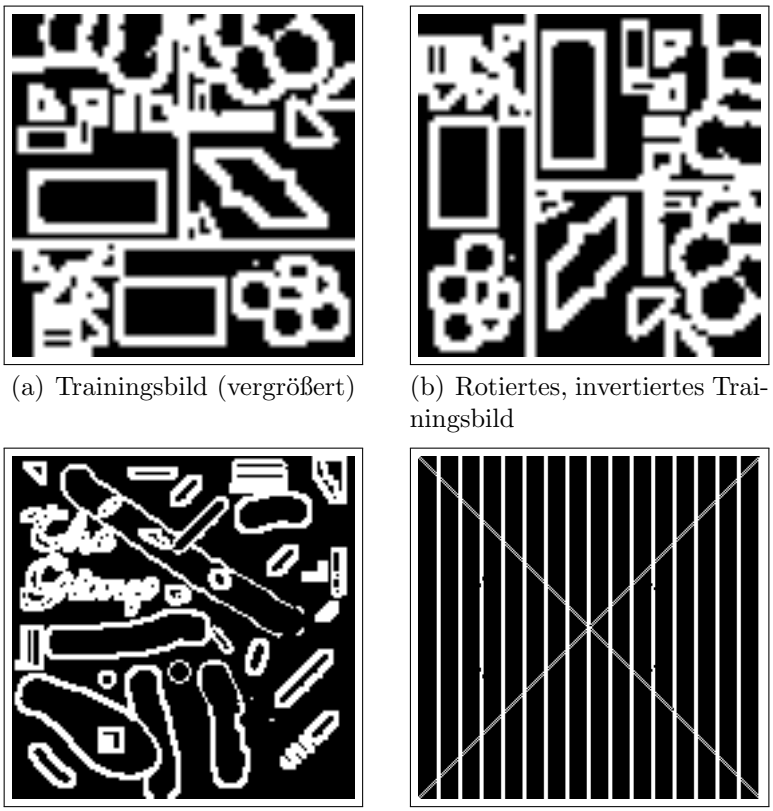
(b) Rotiertes, invertiertes Trainingsbild

(c) Testbild

(d) Ledge (hier verkleinert)

Abbildung 2.7: Ergebnisse EANT2 Kantendetektion 5x5-Maske

Auch der 5x5-Detektor hat die Rotations- und Invertierungsinvarianz erlernt, wie man in Abbildung 2.7 (b) sehen kann. Bei ihm sind allerdings im Testbild (c) deutlich falsch-positive Kanten im Bereich der Helligkeitsübergänge zu erkennen. Dafür hat er die kontrastarmen Kanten erkannt, ebenso wie die Kreise in Ledge. Allerdings sind hier (auf dem Bild leider nicht zu erkennen) die vertikalen Linien sowie die Diagonalen als zu dicke Kanten erkannt.



(a) Trainingsbild (vergrößert)

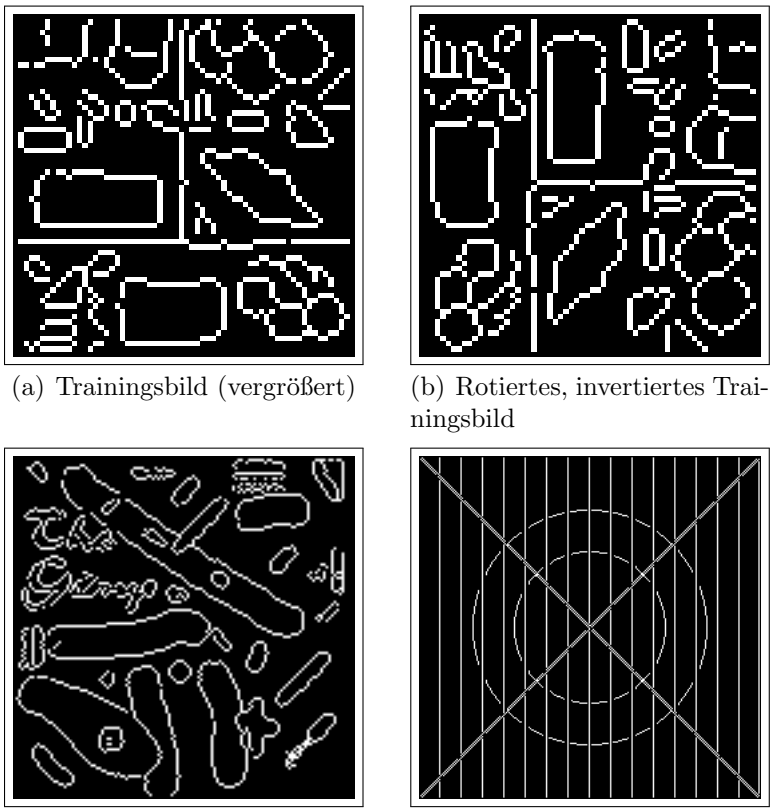
(b) Rotiertes, invertiertes Trainingsbild

(c) Testbild

(d) Ledge (hier verkleinert)

Abbildung 2.8: Ergebnisse Kantendetektion Sobel

Sobel ist durch seine Funktionsweise von Hause aus rotations- und invertierungsinvariant, deshalb sind die Ergebnisse von Abbildung 2.8 (a) und (b) bis auf die Rotation identisch. Bei (c) erkennt er die Übergänge richtig als keine Kante, erkennt aber auch die unteren Kanten dafür nicht als solche. Bei dem Ledge-Bild erkennt auch er den Kreis nicht als Kante.



(a) Trainingsbild (vergrößert)

(b) Rotiertes, invertiertes Trainingsbild

(c) Testbild

(d) Ledge (hier verkleinert)

Abbildung 2.9: Ergebnisse Kantendetektion Canny

Ein Vergleich mit dem Canny-Algorithmus fällt schwer, da dieser durch das „Thinning“ nur Einfachkanten erstellt. Es fällt teilweise schwer, die kleinen Objekte des Trainingsbildes wiederzuerkennen, da teilweise Ecken abgerundet oder runde Verläufe zu spitzen Ecken wurden. Bei dem insgesamt etwas größeren Testbild ist dieser Effekt weniger deutlich. Hier hat Canny die Helligkeitsverläufe richtig als solche erkannt, aber die Kanten unten rechts nicht. Das Ledge-Ergebnis wirkt auf den ersten Blick sehr gut. Allerdings sind die senkrechten Linien an allen Schnittpunkten mit den Diagonalen unterbrochen und sie selbst unterbrechen die Kreise.

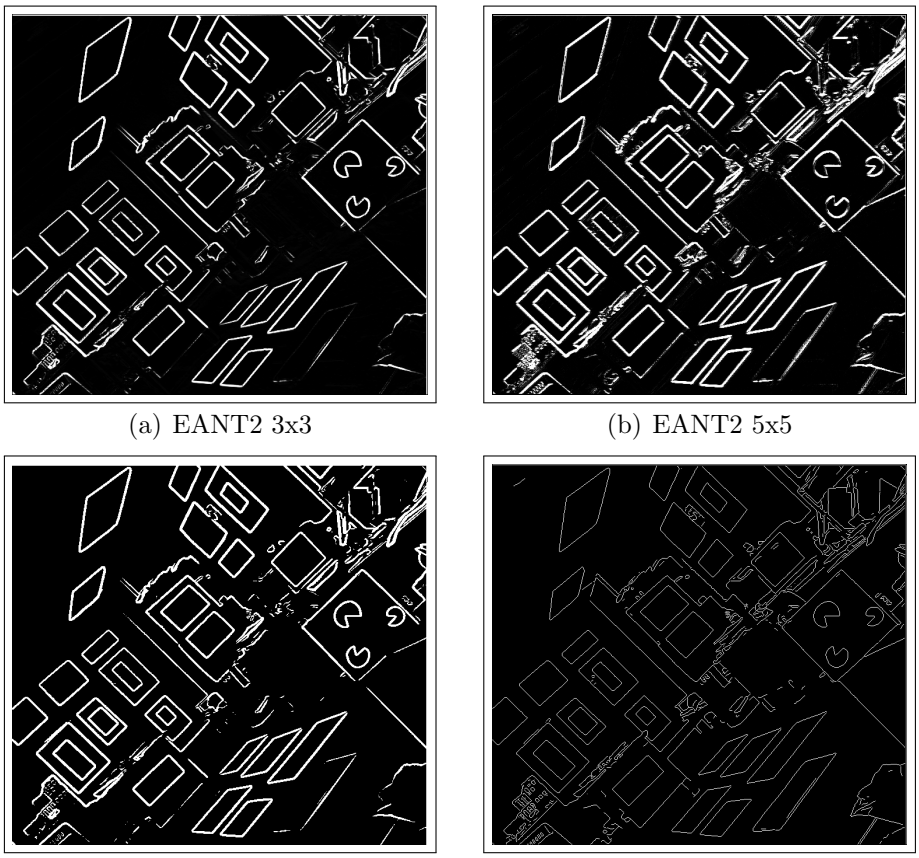
(a) EANT2 3x3 (b) EANT2 5x5

(c) Sobel (d) Canny

Abbildung 2.10: Kantenbilder von Lena der vier Verfahren

In Abbildung 2.10 sieht man die Ergebnisse der Kantendetektoren des Lenabilds. Wie auch schon an den synthetischen Bildern zu erkennen, ist der 5x5-Detektor (b) zu sensitiv und erkennt zu viele Kanten. Der 3x3-Detektor (a) und Sobel (c) erzielen ähnliche Ergebnisse aber der EANT2-Detektor erkennt die Strukturen z.B. des Huts besser. Das Ergebnis des Canny-Algorithmus sieht sehr unruhig aus. Die Augen sind kaum als solche zu erkennen und der Balken links im Hintergrund zeigt keine geraden Linien.

Das Originalbild ist in Abbildung 2.12 (a) zu sehen.



(a) EANT2 3x3 (b) EANT2 5x5

(c) Sobel (d) Canny

Abbildung 2.11: Kantenbilder des Labors der vier Verfahren

Abbildung 2.12 (b) zeigt das Lab-Bild, dessen Kantenbilder in Abbildung 2.11 zu sehen sind. Die Ergebnisse der Kantendetektoren mit Doppelkanten sehen sich sehr ähnlich. Die größten Unterschiede zeigen sich hier bei dem Schreibtisch, der ersten Fläche unten rechts und dem Regal links. Die meisten Kanten detektiert hier der EANT2-5x5-Detektor gefolgt von Sobel und dem 3x3-Detektor.

Das Ergebniss von Canny wirkt hier deutlich ruhiger als bei Lena. Von allen vier Detektoren erkennt er die Fläche unten rechts am besten.

(a) Lena (b) Lab

Abbildung 2.12: Die weit verbreiteten Lena und Lab Bilder

2.4 Fazit

Mit EANT2 konnten Kantendetektoren generiert werden, die Rotations- und Invertierungsinvarianz gelernt haben. Es wurden Netzwerke generiert, die gut von den Trainingsdaten abstrahieren können und auf einer Reihe von Testdaten gute Ergebnisse erzielen. Der 5x5-Detektor zeigt eine deutlich höhere Sensitivität als der 3x3-Detektor, produziert deshalb aber auch mehr falschpositive Ergebnisse. Während der 5x5-Detektor bei dem Trainingsbild und dessen Rotation die besten Ergebnisse von allen Verfahren erzielte, sind seine Ergebnisse für das Ledge- und das Testbild schlechter, als die von Sobel. Der 3x3-Detektor hingegen erzielt auf allen Bildern bessere Ergebnisse als Sobel.

Bild	EANT2 3x3	EANT2 5x5	Sobel
Training	97.72 %	**99,19 %**	96,53 %
Rot., invert.	97,32 %	**98,31 %**	96,53 %
Test	**95,03 %**	91,82 %	93,53 %
Ledge	**94,75 %**	89,52 %	94,22 %

Tabelle 2.1: Richtig klassifizierte Kanten in Prozent

Kapitel 3

Helikopter-Schwebeflug

3.1 Einführung

Die Steuerung eines Helikopters ist eine sehr schwere Aufgabe, die sogar als schwerer angesehen wird als die Steuerung eines Flugzugs [Lei06]. Eines der schwersten Manöver eines Helikopters ist es, an Ort und Stelle zu verharren. Dieses Manöver ist z.B. bei Rettungseinsätzen oder bei Transporten von Außenlasten nötig. Allerdings erfordert es langes Training, damit menschliche Piloten den Schwebeflug beherrschen.

Im Zuge der RL-Competition (Reinforcement Learning-Competition) [WTW09] wurde eine Umgebung zur Verfügung gestellt, in der ein Schwebeflug mittels RL-Glue[1] (einem Framework für Verstärkendes Lernen (siehe folgende Abschnitte)) simuliert und trainiert werden kann.

3.1.1 Verstärkendes Lernen

Eine anschauliche Einführung in Verstärkendes Lernen gibt [HH97]. Im Folgenden wird das Verstärkende Lernen (engl. Reinforcement Learning (RL)) kurz anhand dieses Problems erklärt.

Das RL verfolgt in etwa die Idee des Lernens durch Versuch und Irrtum. Ein Agent versucht sich in einer Umgebung zu behaupten. Dafür muss er, je nach seinem momentanen Zustand, Aktionen durchführen. Für jede Aktion erhält er ein Feedback in Form einer Belohnung. Diese richtet sich nach der Qualität seiner Aktion und ist meist ein negatives Feedback, also eine Bestrafung. Im Fall des Helikopterflugs erhält der Agent also die Eingangsdaten und ermittelt daraus seine Aktion. Als Belohnung erhält er für diesen Schritt beispielsweise die Entfernung zur Sollposition als Strafe. Sollte der Helikopter

[1] http://glue.rl-community.org

durch diese Aktion sogar abstürzen, erhält der Agent noch eine zusätzliche, hohe Strafe (ca. -10^7) und der Durchgang ist beendet. Ansonsten erhält der Agent eine neue Beobachtung, die sich durch seine Aktion aus dem alten Zustand ergibt. Der Agent schreitet also in seiner Umwelt voran. Nach erreichen eines Abbruchkriteriums erhält der Agent teilweise noch eine letzte Belohnung. In diesem Beispiel sind das Erreichen von 6000 Schritten (zu 0,1 Sekunden) oder ein Absturz die Abbruchkriterien. Ein solcher Durchlauf wird auch als Episode bezeichnet.

3.1.2 RL-Glue

RL-Glue ist ein Framework für RL-Aufgaben [TW09]. Es betreibt hierbei eine strenge Trennung zwischen dem Trainer, der die Experimente steuert, der Umgebung, die die Beobachtungen liefert, die Aktionen bewertet und durchführt sowie dem Agenten, der die Lernaufgabe durchführt. Dabei müssen die Komponenten jeweils ein vorgegebenes Interface implementieren, dass die Kommunikation und Steuerung ermöglicht. Das Framework ermöglicht so eine Plattformunabhängigkeit der Komponenten. Es wird somit ermöglicht, auf einfache Art und Weise auf fremde Komponenten zurückzugreifen und sie zu verwenden. Allerdings findet keine direkte Kommunikation oder Steuerung zwischen Agenten und der Umgebung statt, sämtlicher Datenaustausch findet über RL-Glue statt.

3.1.3 Problem

Um das Problem des Schwebeflugs in der simulierten Umgebung der RL-Competition zu lösen, erhält man als Beobachtung 12 Daten:

- Vorwärtsgeschwindigkeit (u_{err})
- Seitliche Geschwindigkeit (v_{err})
- Sinkgeschwindigkeit (d_{err})
- Abstand zur Sollposition (X-Achse) (x_{err})
- Abstand zur Sollposition (Y-Achse) (y_{err})
- Abstand zur Sollposition (Z-Achse) (z_{err})
- Rotation um die X-Achse des Helikopters (p_{err})
- Rotation um die Y-Achse des Helikopters (q_{err})

- Rotation um die Z-Achse des Helikopters (r_{err})
- Quaternionen zur X-Achse (qx_{err})
- Quaternionen zur Y-Achse (qy_{err})
- Quaternionen zur Z-Achse (qz_{err})

Anhand dieser Eingaben müssen die Ausgaben

- Longitudinale, zyklische Blattverstellung (y_1)
- Latitudinale, zyklische Blattverstellung (y_2)
- Kollektiven Blattverstellung Hauptrotor (y_3)
- Kollektiven Blattverstellung Heckrotor (y_4)

ermittelt werden. Diese müssen so gewählt werden, dass der Helikopter seine Anfangsposition hält und vor allem nicht Abstürzt. Erschwert wird die Lösung des Problems durch weiter äußere Einflüsse, wie z.B. den Wind, der nicht direkt beobachtet wird.

3.1.4 Weak-Baseline-Controller

Eine Möglichkeit, den Helikopter im Schwebeflug verharren zu lassen, ist der Weak-Baseline-Controller von Pieter Abbeel, Adam Coates und Andrew Y. Ng der Stanford University. Dieser liegt als Quellcode als Beispiellösung der RL-Competition bei.

Er berechnet die Ausgaben wie folgt:

$$y_1 = -w_y * y_{err} - w_v * v_{err} - w_r * qx_{err} + w_a$$

$$y_2 = -w_x * x_{err} - w_u * u_{err} + w_p * qy_{err} + w_e$$

$$y_3 = -w_q * qz_{err}$$

$$y_4 = w_z * z_{err} + w_w * w_{err} + w_c$$

Dabei sind die Gewichte wie folgt gewählt: $w_a = 0,02$,$w_c = 0,23$, $w_e = 0$, $w_p = 0,7904$, $w_q = 0,1969$, $w_u = 0,0322$, $w_v = 0,0367$, $w_w = 0,1348$ $w_x = 0,0185$, $w_y = 0,0196$, $w_z = 0,0513$

Der Controller schafft es bei 500 Läufen den Helikopter jedes mal über die gesamte Dauer in der Luft zu halten. Dabei erreichte er durchschnittlich eine Fitness von 6670,34.

3.2 Berechnung des Fitnesswerts

Um die Neuronalen Netze mittels EANT2 trainieren zu können, ist es erforderlich, einen Fitnesswert für die Netze anzugeben. Um den Fitnesswert zu ermitteln, wird die durchschnittliche Belohnung gewählt:

$$f = \frac{\sum_{t=0}^{n-1} r_t}{n}$$

Wobei r_t die Belohnung nach Schritt t ist und n die Anzahl der erreichten Schritte. Würde man keinen Durchschnitt bilden, sondern nur die (negativen) Belohnungen aufsummieren, würde ein Netz, das den Helikopter schneller abstürzen lässt als ein anderes, eine bessere Fitness erhalten.

Würde man nur die Anzahl der Schritte betrachten, beispielsweise

$$f = \frac{1}{n} \ ,$$

so würde ein Netzwerk, das den Helikopter steil nach oben fliegen lässt und somit nicht in Gefahr eines Absturzes gerät, das Problem aber auch nicht löst, die gleiche Fitness erhalten, wie ein Netz, welches das Problem wirklich löst.

Ursprünglich war die Simulation des Helikopters zum Betrieb im RL-Glue Framework bestimmt. Hier wird der Aufruf der Agenten von dem Framework gesteuert, so dass es nicht geeignet ist, um mit EANT2 zusammenzuarbeiten, welches selber die Berechnung der Fitness anstoßen können muss. Es war somit nötig, den Simulator, der nur als compilierte Java-Klassen zur Verfügung steht, losgelöst vom Framework zu betreiben.

Die Berechnung der Episoden verläuft hierbei innerhalb einer Black-Box, die, wie sich später herausstellte, sich nicht deterministisch verhält. Um den Simulator zu betreiben, war es nötig, eine Java-Virtual-Machine für jeden Prozess zu erzeugen, in der dieser läuft. Außerdem war es nötig, Daten über die Grenzen verschiedener Programmiersprachen hinaus auszutauschen. Hierbei stellte sich heraus, dass es durchaus üblich ist, innerhalb von Java-Programmen C++-Funktionen aufzurufen, um Geschwindigkeit zu gewinnen. Die andere, hier benötigte, Richtung ist allerdings selten und schlecht dokumentiert.

3.3 Experiment und Ergebnisse

Es wurde ein einzelner Durchlauf von EANT2 durchgeführt mit 15 Strukturen pro Generation, CMAES als Parameteroptimierer und maximal 15000

CMAES-Iterationen. Es wurden innerhalb von 7 Wochen gerade einmal 9 Generationen entwickelt, weshalb auch nur ein Durchlauf durchgeführt wurde. Da sich bei der Auswertung der Ergebnisse herausstellte, dass sich der Simulator nicht deterministisch verhält, wurde noch ein 10. Durchlauf gestartet, mit einer erweiterten Fitnessfunktion, die das Ergebnis von 5 Flugversuchen mittelt:

$$f = \frac{\sum_{i=1}^{5} \sum_{t=0}^{n_i-1} r_t^i}{\sum_{i=1}^{5} n_i},$$

wobei r_t^i die Belohnung nach Schritt t in Durchlauf i ist und n_i die Anzahl der erreichten Schritte in diesem Lauf.

Die Ergebnisse sind in Abbildung 3.1 zu sehen. Verwendet man die entstandenen Netzwerke allerdings in dem Framework und lässt viele Flüge simulieren, stellt man fest, dass ein und dasselbe Netz einmal nach 8 Schritten abstürzt und ein anderes mal die gesamten 6000 Schritte fliegt. Durch diese extremen Unterschiede ist es nicht möglich, zuverlässige Aussagen über die Netze zu treffen. Es kann nicht bewertet werden, ob ein Optimierungsschritt eine Verbesserung oder eine Verschlechterung bewirkt hat. Um hier zuverlässigere Aussagen treffen zu können, müsste man über viele Durchläufe mitteln, was die hohe Laufzeit noch weiter vergrößern würde.

Die Änderung der Fitnessfunktion bewirkt eine Verbesserung der durchschnittlichen Fitnesswerte auf die Hälfte und sorgt für eine geringere Streuung. Die Fitness des besten Individuums konnte sich hierbei jedoch nicht verbessern.

Ein rein zufällig agierender Agent stürzt durchschnittlich nach 3,18 Schritten ab und erreicht eine durchschnittliche Fitness von $3,3 \cdot 10^6$.

3.4 Fazit

Durch die extrem starken Veränderungen innerhalb der Umgebung, die bewirken, dass bei derselben Beobachtung und derselben Aktion unterschiedliche Ergebnisse entstehen (z.B. durch unterschiedliche Windverhältnisse), war es nicht möglich in annehmbarer Zeit Netzwerke zu generieren, die über genügend Robustheit verfügen, diese extrem verrauschten Daten zu verarbeiten. Allerdings war die Fitnessfunktion auch nicht für diese Anforderungen optimiert. Eine Fitnessfunktion, die längere Flüge stärker belohnt und die mehr Flüge durchführt, wäre wahrscheinlich erfolgreicher.

Hinzu kommt, dass der Wind nicht direkt beobachtet werden kann. Somit können Netzwerke, die kein „Gedächtnis“ besitzen, sich nicht auf den Wind einstellen. Hierfür wären Netzwerke mit hohem Grad an Rekurrenz nötig.

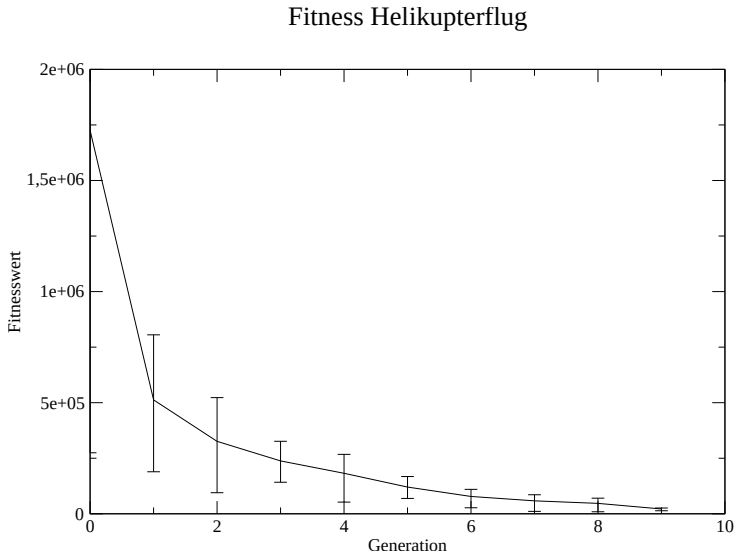


Abbildung 3.1: Fitnesswert der Netzwerke des Helikopterschwebeflugs

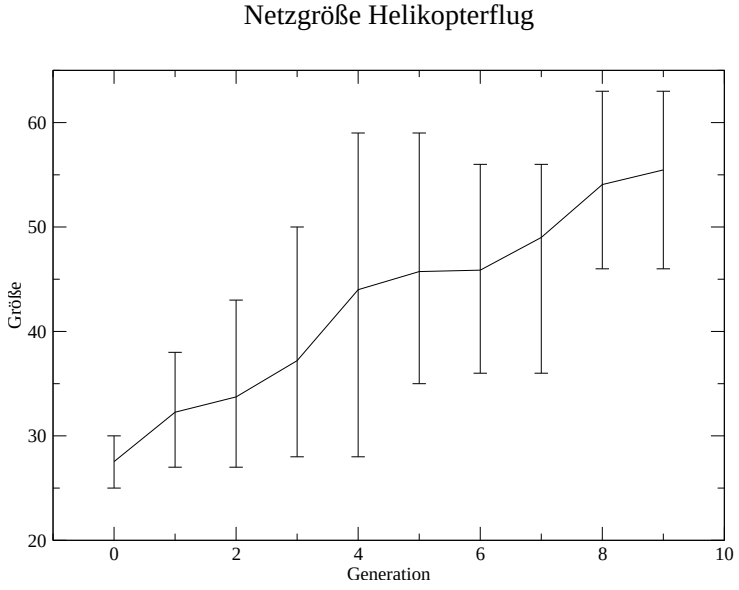


Abbildung 3.2: Größe der Netzwerke des Helikopterschwebeflugs

Kapitel 4

Verbesserungen der Hauptschleife

Die Hauptschleife in EANT2 bezeichnet den gesamten Algorithmus ohne die Parameteroptimierung. Die wichtigsten Punkte an dieser Stelle sind somit die Netzwerke an sich, die Mutation und die Selektion. Die Netzwerke konnten bereits durch die Einführung des Binären Genoms durch Siebel und Jordt [SJS09] stark beschleunigt werden. Außerdem wurden durch Siebel und Reinhold verschiedene Aktivierungsfunktionen sowie RBF-Neuronen eingeführt. Das von Siebel und Bötel eingeführte Pruning [SBS09] mindert durch abtrennen von nicht benötigten Verbindungen ein unnötiges Anwachsen der Netzwerke.

Im Folgenden wird untersucht, ob man durch Weglassen der Reoptimierung der Elterngeneration Geschwindigkeit gewinnen kann und in Abschnitt 4.2 werden die Protokollierungsfunktionen von EANT2 erweitert.

4.1 Reoptimierung

Bei einem Standarddurchlauf von EANT2 werden die besten 30 Netzwerke der Generation i ausgewählt, um als Grundlage für die nächste Strukturgeneration $(i+1)$ zu dienen. Aus diesen 30 Netzen werden mittels Mutation 60 neue Netzwerke generiert. Die Parameter dieser 60 Netzwerke werden dann zusammen mit denen der 30 Eltern-Netzwerken mittels CMAES optimiert (**reoptimiert**), wobei jedes Netzwerk mindestens 2 mal parameteroptimiert wird. Die Eltern-Netzwerke wurden allerdings bereits in der Generation i mit jeweils 2 Durchläufen optimiert. Sollten sie aus der Generation $i-1$ stammen, werden sie also 6 mal optimiert. In Generation 0 werden Netzwerke grundsätzlich 3 mal optimiert, so dass ein Netzwerk, das aus Generation 0

		Verbesserungen (kumuliert)					
	Gesamt	$> 0\%$	$> 1\%$	$> 5\%$	$> 10\%$	$> 50\%$	$> 100\%$
Absolut	41229	15454	2213	349	108	5	2
Relativ (in %)	100	37	5,4	0,85	0,26	0,012	$4,9 * 10^{-3}$

Tabelle 4.1: Verbesserung durch Reoptimierung

stammt und in Generation 2 noch existiert, insgesamt 9 mal optimiert wird.

Da die Auswahl der Netze und deren Mutationen im Vergleich zu der Parameteroptimierung nahezu keine Zeit benötigt, könnte man EANT2 durch Auslassen der Reoptimierung in der darauffolgenden Generation beschleunigen. Damit könnte man die Anzahl der zu optimierenden Strukturen ab der ersten Generation um ein Drittel senken und somit EANT2 um ein Drittel beschleunigen. Dieses Verfahren bietet sich natürlich nur an, wenn man hierbei keine signifikante Verschlechterung der Ergebnisse erhält.

4.1.1 Analyse

Um zu überprüfen, ob eine signifikante Verschlechterung durch Weglassen der Reoptimierung zu erwarten ist, werden 42 ältere EANT2-Durchläufe analysiert. Dabei werden unterschiedlich lange Versuche von 5 bis hin zu 157 Generationen ausgewertet.

Um die Daten zu analysieren, muss die Ausgabedatei des Hauptprogramms betrachtet werden. Hieraus können die Zusammenhänge zwischen den Fitnesswerten vor und nach der Optimierung sowie den Identifikationsnummern aus der Reihenfolge hergeleitet werden.

Insgesamt werden so 41229 Individuen betrachtet, womit sich über 80000 Zweitoptimierungen ergeben, die analysiert werden. Eine Verbesserung wird angenommen, sobald einer der (mindestens) zwei Optimierungsversuchen eine bessere Fitness ergibt als am Ende der vorherigen Strukturgeneration. In Tabelle 4.1 und Abbildung 4.1 sind die Ergebnisse der Analyse zusammengefasst. Hierbei ist zu beachten, dass die Angaben der Verbesserung relativ zu der besseren Fitness angegeben sind. Somit sind Werte von über 100% möglich, da eine Verbesserung von 100% bedeutet, dass sich die Fitness halbiert hat.

Es ergeben sich also in rund einem Drittel der Fälle tatsächlich Verbesserungen durch die zusätzliche Optimierung. Allerdings liegt der größte Anteil an Verbesserungen unterhalb von 1%. Durchschnittlich sinkt die Fitness um

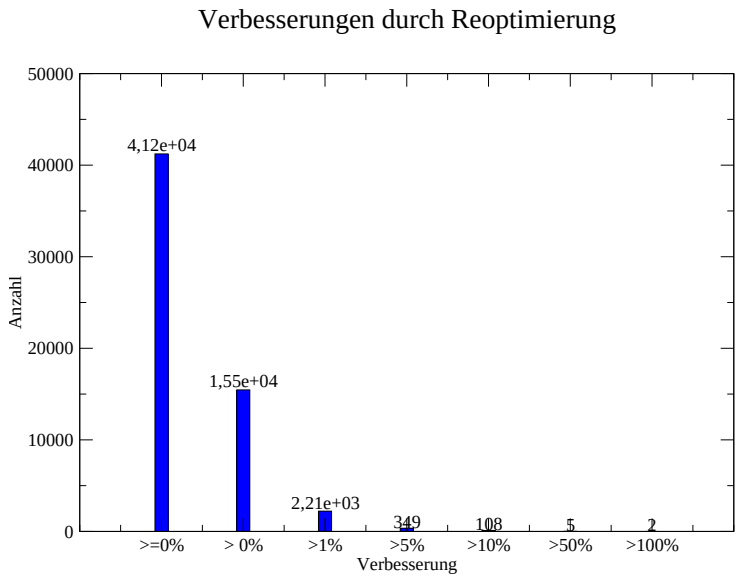


Abbildung 4.1: Verbesserung durch Reoptimierung

$0,087\%$. Betrachtet man weiterhin die Ergebnisse der Netze, die um 50% oder mehr verbessert wurden, so stellt man fest, dass es sich hierbei um vergleichsweise schlechte Netze handelt. Für das Visual-Servoing-Problem mit drei Freiheitsgraden wurden mittels EANT2 Ergebnisse von unter $0,23$ erreicht. Die Netzwerke mit starker Verbesserung erreichten vor der Reoptimierung nur Fitnesswerte von über $0,6$ und durchbrachen nicht die Schwelle von $0,3$, ab der von einem brauchbarem Netzwerk gesprochen werden kann.

Für die besten Netzwerke ist allerdings eine geringe Verbesserung höher zu bewerten, als eine große Verbesserung für die schlechten Netzwerke, sofern diese schlechter bleiben, als die Besseren. Es ist somit Abzuwägen, ob ein Geschwindigkeitsgewinn von über 30% eine eventuelle Verschlechterung der Fitness rechtfertigt. Eine denkbare Möglichkeit wäre auch, die alten Netzwerke nur einmal und nicht zweimal zu optimieren. Ebenso wäre es möglich, nach Beendigung eines EANT2-Durchlaufes nur das beste Netzwerk (mehrmals) zu optimieren.

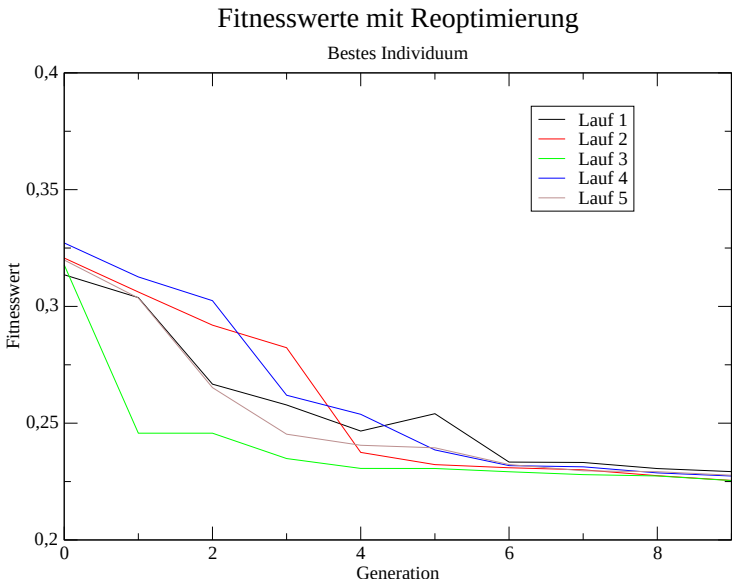


Abbildung 4.2: Referenzdurchlauf mit Reoptimierung - beste Struktur

4.1.2 Versuchsaufbau

Um zu überprüfen, inwieweit das Entfernen der Reoptimierung sich auf die Fitness eines EANT2-Durchlaufs auswirkt, werden 5 Durchläufe mit dem ursprünglichen Ablauf und 5 Durchläufe ohne Reoptimierung gestartet. Als Parameteroptimierung wird CMAES gewählt, welches nach 5000 Iterationen abbricht. In jedem Versuch werden 10 Struktur-Generationen von Null auf erschaffen.

Das zu lösende Problem ist, wie auch bei den meisten analysierten Netzen, das 3DOF-Visual-Servoing-Problem.

4.1.3 Ergebnisse

Die Ergebnisse für die Durchläufe mit Reoptimierung der Elterngeneration sind in Tabelle 4.2 sowie den Abbildungen 4.2 und 4.3 zu sehen. Die erste Abbildung zeigt die jeweiligen Ergebnisse der besten Struktur. In der zweiten Abbildung sind die Durchschnittswerte der einzelnen Läufe abgebildet.

In den ersten fünf Generationen zeigen sich noch recht starke Unterschiede zwischen den einzelnen Läufen. Ab Generation 6 verlaufen alle Durchgänge auf ähnlichem Niveau und es findet nur noch wenig Verbesserung statt.

Im weiteren Verlauf ähneln sich nicht nur die besten Ergebnisse der einzelnen Läufe sondern auch die Ergebnisse innerhalb eines Lauf nehmen fast gleiche Werte an.

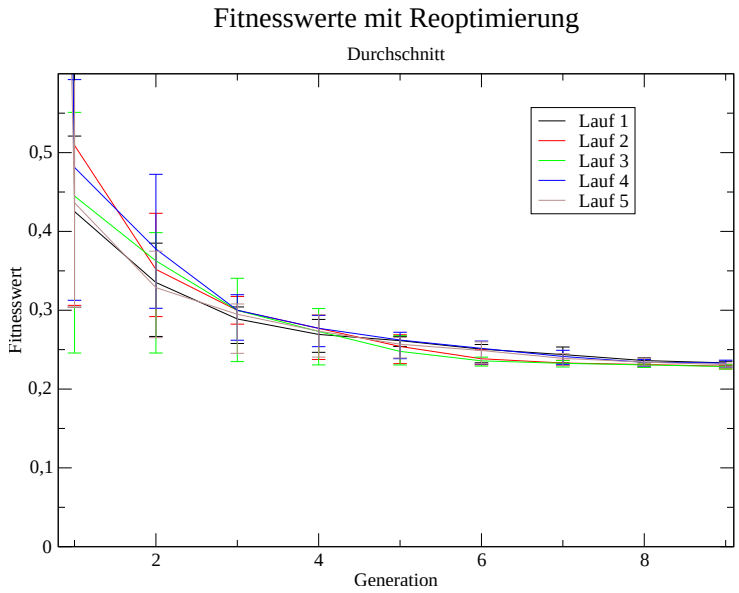


Abbildung 4.3: Referenzdurchlauf mit Reoptimierung - Durchschnittswerte

Generation	Fitnesswert Lauf				
	1	2	3	4	5
0	0.313538795998	0.32070600893	0.317768904133	0.327166652519	0.319845738206
1	0.303769695912	0.30615964178	0.245716830470	0.312614420480	0.303599106253
2	0.266721689009	0.29196937635	0.245700671109	0.302445388995	0.265242311948
3	0.257798646045	0.28232807418	0.234862007109	0.261962903830	0.245274651832
4	0.246621347466	0.23748748938	0.230614908699	0.253786894437	0.240538258404
5	0.254050183109	0.23225886711	0.230598247182	0.238540070997	0.239453858671
6	0.233350888505	0.23090396934	0.229206617304	0.231820814513	0.232280913010
7	0.233168016767	0.23003279093	0.227962437610	0.231317579265	0.229609410955
8	0.230585478737	0.22752707211	0.227393854079	0.228704381329	0.229241433334
9	0.229253713080	0.22547823440	**0.225419895249**	0.227362712314	0.227722343387

Tabelle 4.2: Ergebnisse für das bestes Individuum mit Reopitmierung

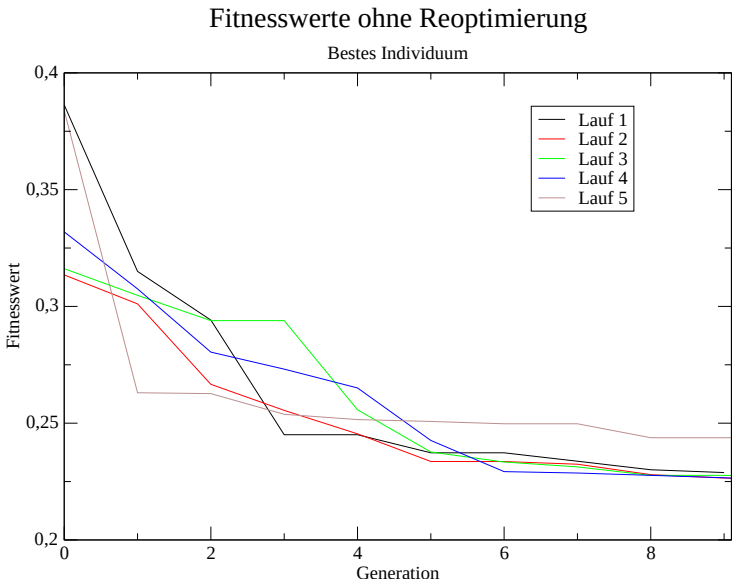


Abbildung 4.4: Durchlauf ohne Reoptimierung - beste Struktur

Bei den Läufen ohne Reoptimierung, deren Ergebnisse in den Abbildungen 4.4 bzw. 4.3 zu sehen sind, ist auch ein Angleichen der Ergebnisse zu beobachten, das allerdings erst später beginnt. Auffällig ist Lauf 5, der mit einer Fitness von $0,244$ deutlich schlechter als die anderen Läufe ist. Da die anderen Läufe aber ein anderes Verhalten zeigen und sich dabei sehr ähneln, handelt es sich hier wahrscheinlich um einen untypischen Ausreißer.

Und auch hier nähern sich die Ergebnisse der Läufe innerhalb einer Generation mit fortschreitender Generation immer weiter einander an.

Wie man in Tabelle 4.3 sehen kann, sind die besten zwei Ergebnisse mit Reoptimierung besser als alle Ergebnisse ohne Reoptimierung. Auf der anderen Seite sind aber auch die zwei besten Netze, die ohne Reoptimierung entstanden sind, besser als drei Netze mit Reoptimierung, somit scheint der Vorteil durch die Reoptimierung nicht immer aufzutreten und nur gering zu sein.

In Abbildung 4.6 ist der Durchschnitt der besten Struktur über alle Läufe dargestellt. Dabei wurde das schlechteste Ergebnis der beiden Versuche nicht berücksichtigt, da es sich bei dem Versuch ohne Reoptimierung um einen Ausreißer zu handeln scheint, der das Ergebnis verfälschen würde. Somit wurde bei beiden Graphen über vier Durchgänge gemittelt.

Man sieht, dass die Läufe mit Reoptimierung konstant besser sind, als ohne. Allerdings wird dieser Abstand zum Ende hin verschwindend gering.

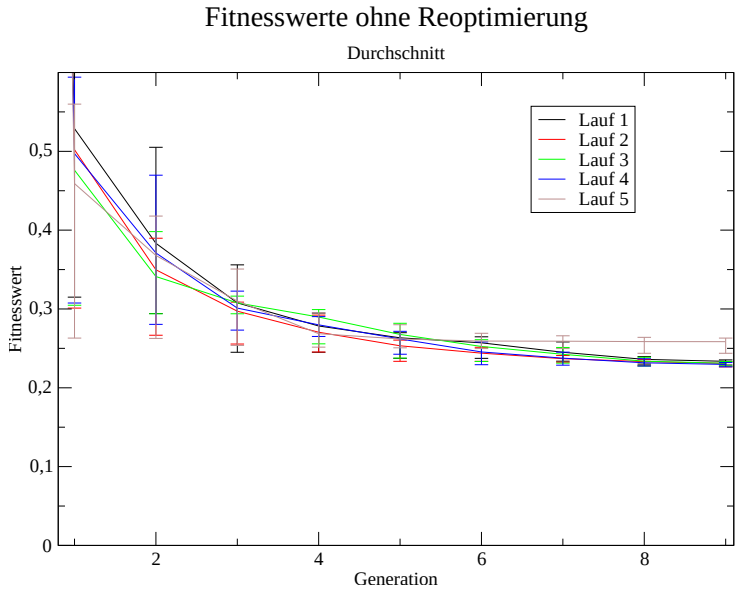


Abbildung 4.5: Durchlauf ohne Reoptimierung - Durchschnittswerte

Ohne Reoptimierung					
	Fitnesswert Lauf				
Generation	1	2	3	4	5
0	0.386292684759	0.313487667093	0.316148452873	0.331922741103	0.383928311511
1	0.315006772641	0.301078756771	0.304738623679	0.307511321673	0.263039127696
2	0.294129503672	0.266640144189	0.293917275926	0.280428901778	0.262665725329
3	0.245053860255	0.255473404942	0.293917275926	0.273149895052	0.253801837286
4	0.245053860255	0.24543762453	0.255750119696	0.265093507788	0.251528668314
5	0.237312320056	0.233603371546	0.23765675589	0.242600396001	0.250735296386
6	0.237312320056	0.233573234618	0.23337398599	0.229256087823	0.24974868118
7	0.23370033364	0.232440307672	0.231273674683	0.228655939074	0.24974868118
8	0.230057627156	0.227984099108	0.227667966677	0.227650216887	0.243779396457
9	0.228862534361	**0.226511854393**	0.227667966677	0.2266400498	0.243779396457
Mit Reoptimierung					
Generation	1	2	3	4	5
9	0.229253713080	0.22547823440	**0.225419895249**	0.227362712314	0.227722343387

Tabelle 4.3: Ergebnisse bestes Individuum ohne Reoptimierung sowie letze Generation mit Reoptimierung

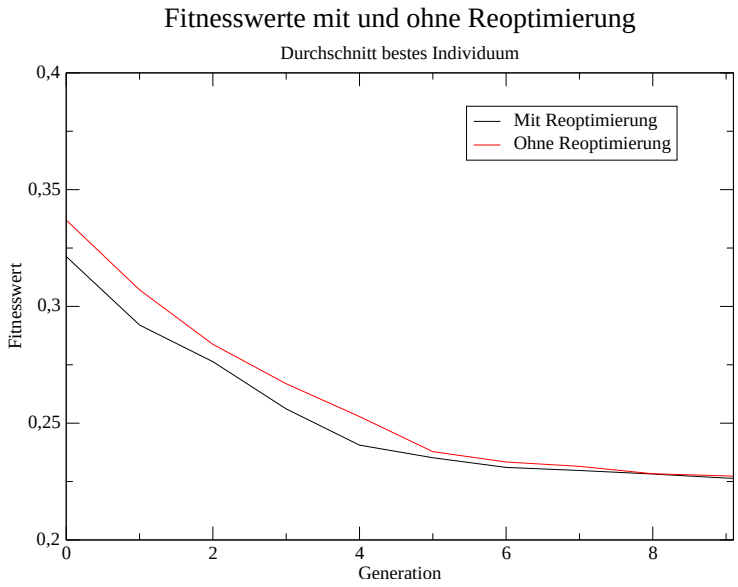


Abbildung 4.6: Durchschnittswert der Besten Ergebnisse mit und ohne Reoptimierung.

4.1.4 Fazit

EANT2 optimiert die Parameter der Eltern in jeder Generation zusammen mit den Kindern ein weiteres mal mit. Entfernt man diese Reoptimierung, so beschleunigt sich die Entwicklung jeder Generation um bis zu ein Drittel. Demgegenüber steht eine Verschlechterung, die bei diesem Test im schlimmsten Fall (bestes Individuum mit Reoptimierung gegenüber schlechtestem ohne) bei rund 8,1% liegt. Durchschnittlich ergibt sich eine Verschlechterung um 1,6%. Das beste Ergebnis ohne Reoptimierung ist sogar nur 0,5% schlechter als sein mehrfach optimiertes Pendant. Da der Geschwindigkeitsgewinn jedoch mit über 30% sehr deutlich ist, könnte man am Ende des Durchlaufs nur das beste Netzwerk mehrmals und evtl. mit höherer Populationszahl (siehe Kapitel 7) optimieren und würde trotzdem Zeit gewinnen.

4.2 Erweiterung der Protokollierung

Um ein Verfahren zu optimieren, ist es sinnvoll genaue Daten über die Schritte des Verfahrens und deren Ergebnisse zu erhalten.

EANT2 gibt in seiner ursprünglichen Form nach jeder Parameteroptimierung und vor der nächsten Hauptiteration, in der neue Strukturen generiert werden, als Ausgabe die Fitnesswerte jedes optimierten Individuums. Als nächstes werden die Netze ausgegeben, die die Grundlage für die nächste Generation bilden, also in der Regel die 30 besten Strukturen (mit Gewichten). Diese Ausgabe enthält neben der Struktur des Netzes, dem Fitnesswert, der Länge und einer eindeutigen Identifikationsnummer (ID) noch die ID des Vorfahrens. Außerdem werden die besten Netze als XML-Datei auf die Festplatte geschrieben. Zusätzlich werden während der Parameteroptimierung weitere Informationen ausgegeben, die allerdings von der verwendeten Optimierungsmethode abhängen. Unter diesen Informationen befindet sich die aktuelle Fitness im Verlauf der Optimierung sowie die Anzahl der Funktionsevaluationen. Die ID des gerade optimierten Netzes lässt sich nicht direkt erkennen.

Durch die gesammelten Daten kann man zum Beispiel den Weg des besten Netzes zurückverfolgen. Eine vorwärts gerichtete Analyse ist jedoch nur begrenzt möglich, da keine Informationen der Netze vorliegen, die nicht zu den Besten einer Generation gehörten. Möchte man erfahren, durch welche Mutation ein Netz aus seinem Vorgänger hervorgegangen ist, ist es nötig, beide Netze zu vergleichen, und manuell aus dem Unterschied die Mutation herzuleiten.

4.2.1 Datenbank

Um zukünftig besser das Verhalten von EANT2 analysieren zu können, wurde dieses um umfassendere Protokollierungsfunktionen erweitert. Die Daten werden in einer separaten Datenstruktur gesammelt und nach jeder Generation als XML-Dokument geschrieben. Diese XML-Datei wird eingelesen, wenn ein unterbrochener Durchgang fortgesetzt wird.

In der Datenbank werden Informationen für alle Netze gespeichert, somit auch für Netze, die in derselben Generation, in der sie entstanden sind, verworfen werden. Die gespeicherten Informationen umfassen die ID des Netzes und seines Vorfahrens, die Fitness bei der Erschaffung sowie die aktuelle Fitness, Einträge über die Lebensdauer, Daten über alle Optimierungsdurchläufe sowie über die Nachfahren.

Die Daten über die Optimierungen enthalten dabei Informationen über die Strukturgeneration, in der sie stattfinden, die Fitnesswerte vor und nach

der Optimierung sowie über die Anzahl der Funktionsevaluationen bei der Optimierung.

Als Information über die Nachfahren wird zu jedem Kind gespeichert, welche ID es erhalten hat, in welcher Generation es erzeugt wurde und durch welche Mutation es entstand.

Der Aufbau des XML-Dokuments entspricht dabei weitgehend dem Aufbau der Datenbank. In der `NeuroEvolutionDatabase` ist eine Liste von `NeuroEvolutionData` gespeichert, die den einzelnen Individuen entsprechen. Hier sind oben genannte Informationen gespeichert, wobei die Optimierungsschritte als Liste von `EvolutionaryOptimisationData` vorliegen und die Nachfahren direkt aus der Liste der `EvolutionaryMutationData` gelesen werden können.

```
<NeuroEvolutionDatabase>
  <NeuroEvolutionData>
    <id>1</id>
    <parent>0</parent>
    <generationOfFirstOccurance>0</generationOfFirstOccurance>
    <generationOfLastOccurance>0</generationOfLastOccurance>
    <currentFitness>-0.562916054865772896854</currentFitness>
    <bestFitness>-0.562916054865772896854</bestFitness>
    <startFitness>-1.556333432905314895223</startFitness>
    <optimisations>
      <EvolutionaryOptimisationData>
        ...
      </EvolutionaryOptimisationData>
    </optimisations>
    <children>
      <EvolutionaryMutationData>
        ...
      </EvolutionaryMutationData>
    </children>
  </NeuroEvolutionData>
</NeuroEvolutionDatabase>
```

`EvolutionaryOptimisationData` wird in der Datenbank als eigener Datentyp repräsentiert, in dem Generation, Fitnesswerte und Evaluationen gespeichert werden.

```
<EvolutionaryOptimisationData>
  <generation>0</generation>
  <oldFitness>-1.556333432905314895223</oldFitness>
  <newFitness>-0.563467413317613297075</newFitness>
  <evaluations>10035</evaluations>
</EvolutionaryOptimisationData>
```

`EvolutionaryMutationData` enthält Angaben über die Generation, die

Art der Mutation sowie eine Referenz auf das neu entstandene Netz.

```
<EvolutionaryMutationData>
  <generation>0</generation>
  <newId>13</newId>
  <mutationAction>AddRandomConnection</mutationAction>
</EvolutionaryMutationData>
```

Die so gewonnenen Daten sind als Datenstruktur einfach maschinell auswertbar und auch die XML-Dokumente können direkt analysiert werden. So ist beispielsweise sofort ersichtlich, ob die mehrfache Parameteroptimierung eines Netzes erfolgreich war und, wenn ja, um wie viel sich das Netz hierdurch verbessert hat.

Kapitel 5

Parameteroptimierung

Die Optimierung der Parameter der von EANT2 generierten Netzwerke ist eine der wichtigsten Komponenten der Methode. Durch den modularen Aufbau von EANT2 ist es möglich, die Parameteroptimierung durch andere Verfahren zu ersetzen oder die Parameter von CMAES zu modifizieren, ohne den prinzipiellen Ablauf von EANT2 zu verändern. Aufgrund dieses modularen Aufbaus ist es auch möglich, die Parameteroptimierung zu testen, ohne für jeden Durchlauf von Null anzufangen und neue Netzwerke zu generieren. Diese Testmethode wird in Abschnitt 5.1 vorgestellt. Ein vollständiger Austausch von CMAES durch eine andere Optimierungsmethode wird in Kapitel 6 untersucht, während in Kapitel 7 versucht wird, CMAES mittels Erhöhung der Populationsgröße zu verbessern. Kapitel 8 beschäftigt sich damit, ob eine Beschleunigung von CMAES durch Reduktion der Funktionsevaluationen mittels lokaler Modelle möglich ist.

5.1 Verkürzter EANT2-Durchlauf

Um aussagekräftige Daten für randomisierte Algorithmen zu sammeln, ist es nötig, Ergebnisse von mehreren Durchläufen zu generieren und zu vergleichen. Aufgrund der Rechenintensivität von vollständigen EANT2-Durchläufen wurden Versuche mit verkürzten Runden benutzt. Bei diesen verkürzten Runden werden bereits in früheren Versuchen generierte Netzwerke aus höheren Generationen geladen und optimiert. Es finden keine Mutationen der Netzwerke statt, sondern eine reine Parameteroptimierung. Da der Vorgang der Strukturentwicklung unabhängig von der Parameteroptimierung ist (mit Ausnahme der Selektion auf Grundlage der Fitnesswerte), ergeben diese Versuche Ergebnisse, die direkt Rückschlüsse auf die Leistung der Optimierer in EANT2 schließen lassen. Außerdem kann so ausgeschlos-

sen werden, dass bei einem Testdurchlauf ein Optimierer besser ist als ein anderer Optimierer, aber aufgrund schlechterer Strukturen ein schlechteres Ergebnis generiert.

Algorithmus 2 Verkürzter EANT2-Durchlauf

```
procedure EANT2
    for all Generationen do
        for (i = 0; i < 5; i++) do
            Lade bestehende Netzwerke
            Parameteroptimierung mittels CMA-ES oder PSO
            Gib optimierte Netzwerke aus
        end for
    end for
end procedure
```

Um möglichst verschiedene Netze als Grundlage des Tests zu benutzen, werden Netzwerke aus drei verschiedenen Generationen aus unterschiedlichen Durchläufen in unterschiedlichen Entwicklungsstadien gewählt. Die Fitnesswerte und Größen der Netzwerke können den Abbildungen 5.1 und 5.2 entnommen werden.

Die Netzwerke aus Generation 1 besitzen hierbei die größte Streuung an Fitnesswerten, die von 0,3038 bis zu 0,5209 reicht. Die Größe reicht von 18 bis 25 Parametern.

In Generation 15 liegen die Fitnesswerte im Bereich von 0,2259 bis zu 0,25 und somit deutlich dichter beieinander, dafür liegen hier die Größen mit 53 bis 96 sehr weit auseinander. Die Fitnesswerte liegen hier schon in einem sehr guten Bereich.

Keinen Unterschied bieten die Netzwerke der Generation 150 bei der Fitness zueinander: Sie alle liegen bei 0,2213. Diesen Wert erreicht das kleinste Netzwerk bei einer Größe von gerade einmal 20 Gewichten, während das Größte mit 32 Parameters kleiner ist, als das Kleinste der Generation 15. Diese Netze stammen aus früheren Versuchen von Siebel und sind ein Beispiel für hoch optimierte Netzwerke, die trotz ihrer geringen Größe ein sehr gutes Ergebnis erzielen. Sie sind außerdem ein gutes Beispiel dafür, das in EANT2 kein „Survival of the Fattest“ gilt.

Diese Netzwerke werden innerhalb eines verkürzten EANT2-Durchlaufs jeweils zweimal optimiert und jeder verkürzte Durchlauf fünfmal durchgeführt, so dass jedes Netzwerk zehnmal optimiert wird. Ein Test besteht also aus 66 verschiedenen Netzwerken, die zehnmal optimiert werden. Es werden somit also 660 Optimierungsversuche pro Testlauf ausgewertet.

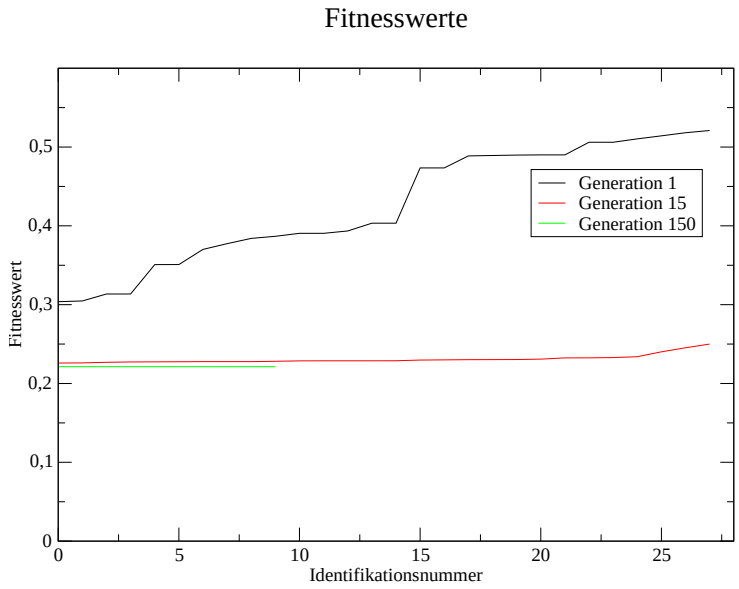


Abbildung 5.1: Ausgangsfitnesswerte der Netzwerke zum Parameteroptimierungstest

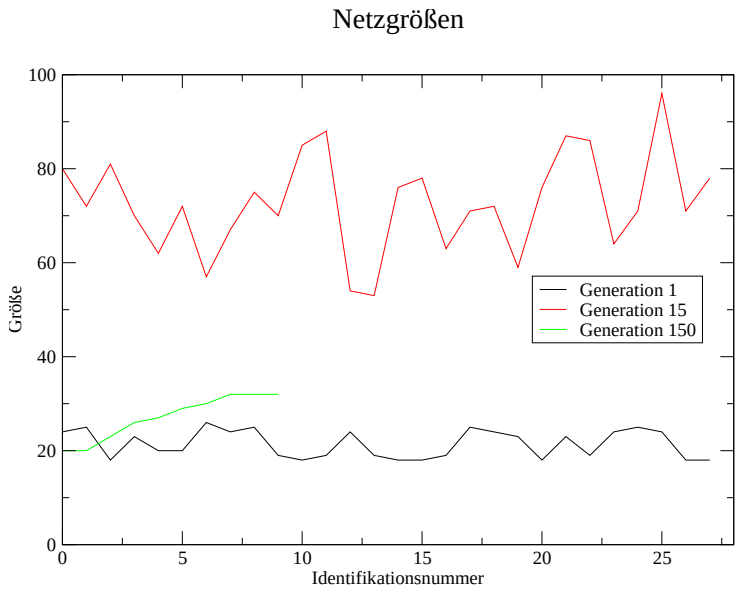


Abbildung 5.2: Größe der Netzwerke zum Parameteroptimierungstest

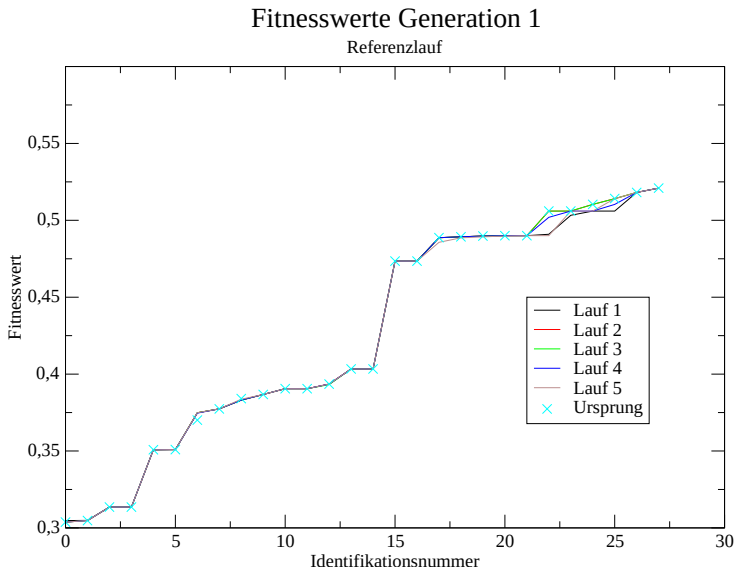


Abbildung 5.3: Fitnesswerte des Referenzdurchlaufs der 1. Generation. X markiert Fitnesswerte, die in den Netzen gespeichert sind.

5.1.1 Referenzdurchlauf

Um Daten als Basis für Vergleiche heranziehen zu können, wurde ein Referenzdurchlauf durchgeführt. Dafür wurden die Netzwerke mittels CMAES über 5000 Iterationen optimiert. Die Abbildungen 5.3 bis 5.5 zeigen die Ergebnisse.

Bei Generation 1 sind nahezu keine Abweichungen von den ursprünglichen Netzen zu beobachten. Auch unterscheiden sich die Ergebnisse der einzelnen Durchläufe nur sehr wenig voneinander.

Die Ergebnisse von Generation 15 zeigen bei den höheren Strukturen deutliche „Ausreißer“, die so stark von den anderen Ergebnissen abweichen, dass hier von einem Fehler ausgegangen werden muss. Deshalb werden diese Werte auch nicht weiter berücksichtigt. Bis auf eine kleine Verschlechterung bei Struktur 2 in einem Lauf sind auch hier wieder die Ergebnisse der besten Netzwerke in jedem Lauf fast identisch.

Für die hoch optimierten Netzwerke der Generation 150, die ihre guten Fitnesswerte aus deutlich längeren Optimierungsphasen haben, konnten die guten Vorgaben nicht erreicht werden und die einzelnen Durchläufe zeigen teilweise deutliche Unterschiede.

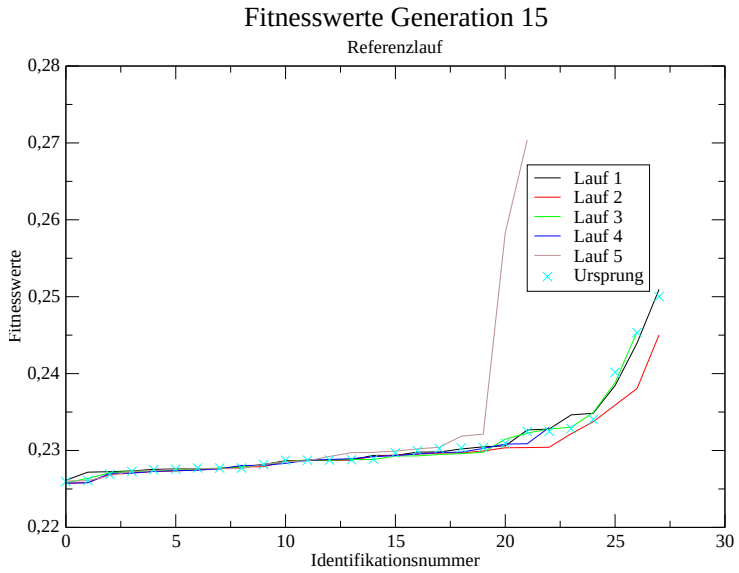


Abbildung 5.4: Fitnesswerte des Referenzdurchlaufs der 15. Generation. X markiert Fitnesswerte, die in den Netzen gespeichert sind.

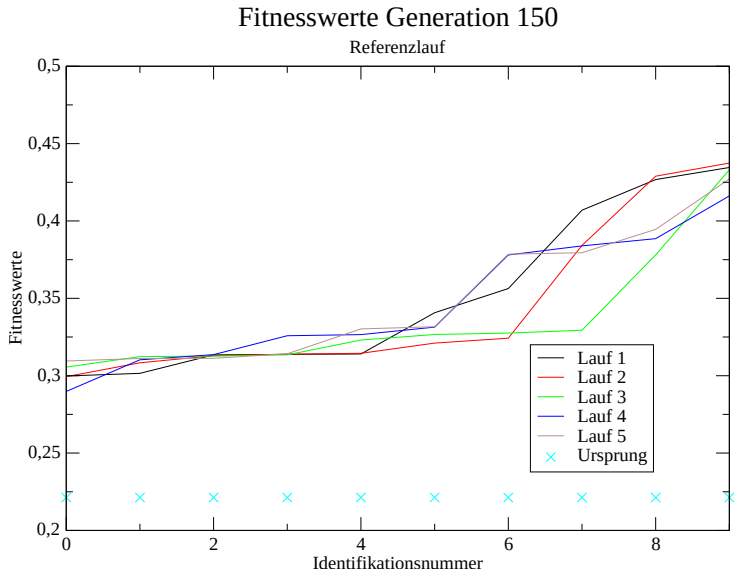


Abbildung 5.5: Fitnesswerte des Referenzdurchlaufs der 150. Generation. X markiert Fitnesswerte, die in den Netzen gespeichert sind.

Generation	Min	Max	∅
1	0.303769695913	0.520949628965	0.424919539821
15	0.225729577421	0.270374979058	0.23030684742
150	0.289847592718	0.437439525049	0.345333504834

Tabelle 5.1: Ergebnisse von jeweils 5 Durchläufen Optimierung durch CMAES bei 5000 Iterationen

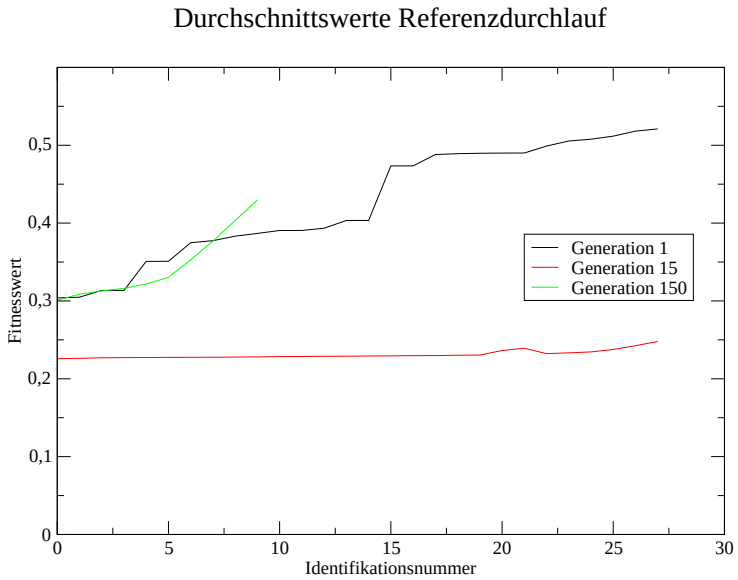


Abbildung 5.6: Gemittelte Fitnesswerte der Referenzdurchläufe

Vor allem in den Generationen 1 und 15 zeigt sich, dass die Ergebnisse des verkürzten Testdurchlaufs sehr gut mit den Ergebnissen des vollständigen Testdurchlaufs übereinstimmen, so dass umgekehrt auch Ergebnisse von diesem Versuchsaufbau auf EANT2 übertragen werden können.

Aus dem Testdurchlauf ergeben sich die in Abbildung 5.6 errechneten Durchschnittswerte, die als Grundlage für die Bewertung anderer Versuche dienen.

Kapitel 6

Partikel-Schwarm-Optimierung

Einer der wichtigsten Teile innerhalb von EANT2 ist die Parameteroptimierung. Nur nachdem die Parameter, also meist die Gewichte der Netzwerke, optimiert wurden, kann die Qualität eines Netzwerkes überhaupt bewertet werden. EANT2 verwendet für die Optimierung der Parameter CMA-ES [HO01, SKS07]. Im Folgendem soll untersucht werden, ob EANT2 durch Benutzung einer anderen Optimierungsstrategie verbessert werden kann. Hierfür wurde die Partikel-Schwarm-Optimierung (PSO) ausgewählt, die sich, ebenso wie Evolutionäre Algorithmen und Neuronale Netze, auch einem Vorbild in der Natur bedient.

6.1 Einführung in die Partikel-Schwarm-Optimierung

PSO wurde bereits erfolgreich benutzt, um Neuronale Netze zu trainieren und sogar zu generieren (z.B. in [ZSL00]) und ist teilweise der „Backpropagation“ überlegen.

Anders als bei Evolutionären Algorithmen, die sich sehr lange Zeiträume und die Entwicklung einer Spezies zum Vorbild genommen haben, werden bei PSO kürzere Zeiträume simuliert. Mit Evolutionären Algorithmen hat PSO gemein, dass es ein populationsbasiertes Verfahren ist, dessen Individuen auch Agenten oder Partikel (daher der Name) genannt werden. Im Gegensatz zu EAs mutieren die Agenten nicht, reproduzieren sich nicht oder sterben aus, sondern es werden die Wanderungen der Individuen im Suchraum betrachtet. Die Idee hinter PSO lehnt sich dabei an Vogelschwärme auf Futtersuche an. Es wird angenommen, dass sich die Vögel im Schwarm etwa auf denjenigen zu bewegen, der die beste Futterstelle (die beste Lösung) gefunden hat. Hierfür findet ein Informationsaustausch statt, so dass jedes

Individuum des Schwarms die beste, bis jetzt gefundene, Lösung kennt. Außerdem merkt sich jeder Partikel i seine bis jetzt gefundene beste Lösung. Aus seiner momentanen Geschwindigkeit v_t^i zum Zeitpunkt t, dem persönlichem besten Ergebnis $pbest_t^i$ und dem global am besten beobachteten Ergebnis $gbest_t$ errechnet sich seine neue Bewegung durch

$$v_{t+1}^i = w * v_t^i + c_1 * \text{rand}() * (pbest_t^i - p_t^i) + c_2 * \text{rand}() * (gbest_t - p_t^i) \quad (6.1)$$

und seine neue Position pos_{t+1} durch

$$p_{t+1}^i = p_t^i + v_{t+1}^i \quad (6.2)$$

Dabei stehen rand() für Zufallszahlen aus $[0, 1]$ und w, c_1 und c_2 sind Lernfaktoren, für die im Original $w = 1$" und $c_1 = c_2 = 2$ gewählt wurden. Im hier verwendeten „standardPSO2007" allerdings werden $c_1 = c_2 = 1,1931$ und $w = 0,7213$ nach [Cle06] gesetzt.

Interpretiert man den Algorithmus in Hinblick auf seinen Ursprung als Simulation von Sozialverhalten, so steht der Term $(pbest_t^i - p_t^i)$ für die eigenen Erfahrungen und das eigene Lernen der jeweiligen Agenten und $(gbest_t - p_t^i)$ steht für das Lernen als Gruppe.

PSO läßt sich mit folgendem Pseudocode beschreiben:

Algorithmus 3 PSO-Pseudocode

```
procedure ParticleSwarmOptimisation
    Initialisiere zufällige Population
    while keinStop do
        Berechne Fitness
        Aktualisiere pbest
        if Stopkriterium erreicht then
            Setze keinStop auf False
        end if
        for all Partikel do
            Berechne Bewegung nach Gleichung (6.1)
            Berechne Position nach Gleichung (6.2)
        end for
    end while
end procedure
```

Eine Variante von PSO sieht vor, nicht den global besten Kandidaten ($gbest$) zu betrachten, sondern das beste Ergebnis der jeweils n nächsten Nachbarn ($nbest_t^i$)

6.2 PSO in EANT2

PSO wird in EANT2 als Alternative zu CMA-ES in der Parameteroptimierung der Neuronalen Netzwerke getestet. Dafür ist es nötig, die, dankenswerterweise von Maurice Clerc[1] zur Verfügung gestellte, Implementierung anzupassen, um in dem objektorientiertem Design von EANT2 arbeiten zu können.

Es wurde die Möglichkeit geschaffen, für EANT2 mittels der NeuroEvolution.xml das Parameteroptimierungsverfahren auszuwählen, so dass mit den selben Binärprogrammen entweder CMA-ES oder PSO benutzt werden kann.

6.2.1 Adaptive Suchraumkorrektur

Da in der Standardimplementierung von PSO die Grenzen des Suchraums vorgegeben werden müssen, diese aber a priori nicht bekannt sind, müsste man diese Grenzen sehr weit fassen, um keine guten Lösungen im Vorfeld auszuschließen. Um dies zu verhindern, wurde eine adaptive Anpassung der Suchraumgrenzen entwickelt. Bei der Initialisierung werden die Grenzen für jeden Parameter auf [-1000,1000] gesetzt. In den folgenden Runden werden die Besten 20% der Population alle 3 Iterationen ausgewertet und wenn sie sich in der Nähe der Grenzen befinden, werden diese vergrößert.

6.2.2 Probedurchlauf

EANT2 mit PSO wurde in normalen Visual-Servoing EANT2-Durchläufen auf Funktionstüchtigkeit getestet, wobei 40 Strukturgenerationen erstellt wurden. Hierbei waren allerdings noch keine PSO-Parameter optimiert und die Anzahl der Funktionsevaluationen wurde auf 10000 begrenzt. Die Schwarmgröße beträgt standardmäßig $10 + 2 \cdot \sqrt{d}$, wobei d die Parameteranzahl des Netzes ist. Das Ergebnis ist in Abbildung 6.1 zu sehen. Die hier erreichte Fitness ist auch nach 40 Generationen schlechter, als die Fitness im Referenzlauf mit CMAES nach einer Generation, was allerdings auf die geringe Anzahl der Funktionsevaluationen zurückzuführen ist.

[1] http://www.particleswarm.info/Programs.html

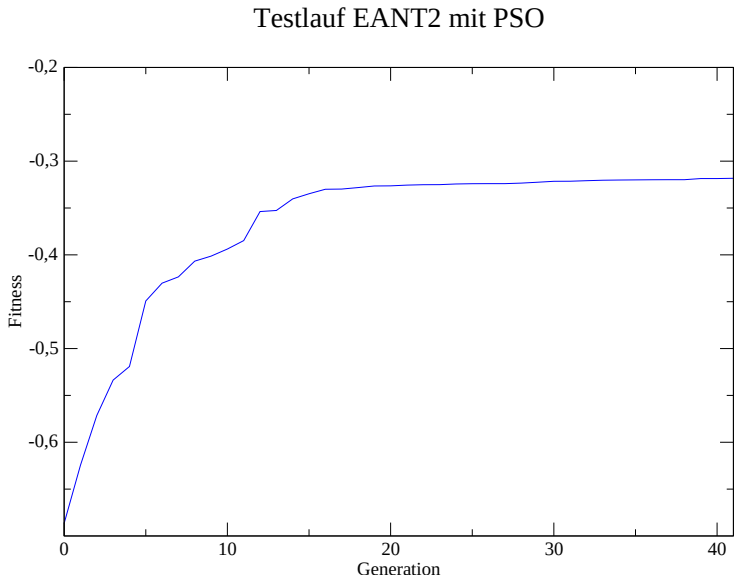


Abbildung 6.1: Ergebnis eines EANT2 Testlaufs mit PSO

Vollständiger EANT2-Durchlauf mit 5000 Iterationen

In einem anderen Probedurchlauf wurden vollständige EANT2-Durchläufe mit PSO bzw. CMAES durchgeführt, in denen die Parameteroptimierer jeweils 5000 Schritte pro Optimierung durchführten, um das VS-Problem zu lösen.

Die Ergebnisse sind in Abbildung 6.2 zu sehen.

In den ersten 7 Generationen ist CMAES PSO deutlich überlegen, ab Generation 8 generieren allerdings beide Durchläufe etwa gleich gute Ergebnisse, wobei PSO eine Fitness von $0,2265$ und CMAES $0,2254$ erreicht.

Bei der Netzgröße verhält es sich andersherum. Hier sind beide Läufe bis etwa Generation 6 gleich groß, ab Generation 7 steigt die Netzgröße des PSO-Durchlaufs allerdings stärker als die von CMAES.

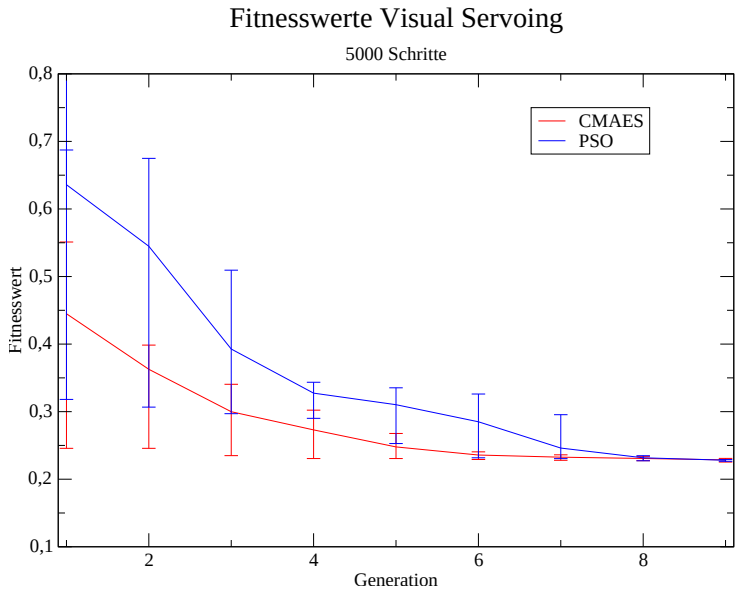


Abbildung 6.2: Ergebnis eines EANT2-Laufs mit PSO bzw. CMAES

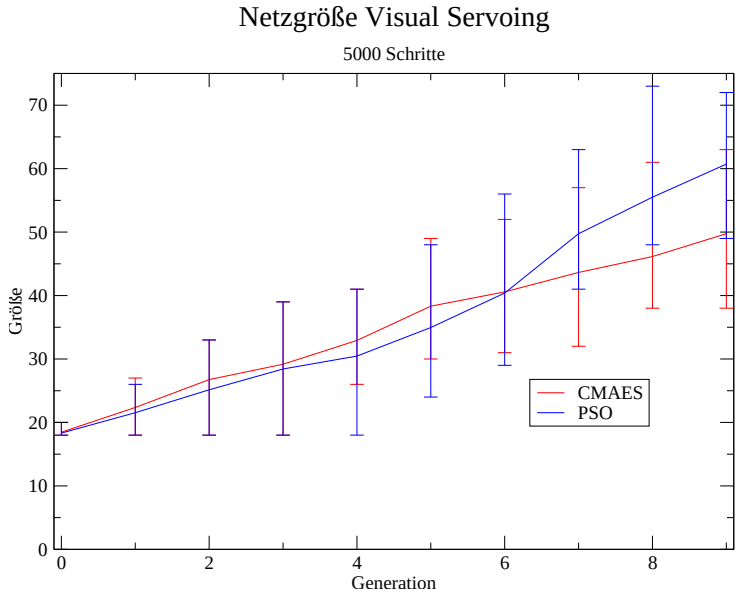


Abbildung 6.3: Netzgröße eines EANT2-Laufs mit PSO bzw. CMAES

PSO			
Generation	Min	Max	∅
1	0.305024661417	0.924402417822	0.462788779025
15	0.243190849969	0.684390523366	0.425651917583
150	0.451716508599	2.37504017034	0.819546917389
CMAES			
Generation	Min	Max	∅
1	0.303769695913	0.520949628965	0.424919539821
15	0.225729577421	0.270374979058	0.23030684742
150	0.289847592718	0.437439525049	0.345333504834

Tabelle 6.1: Ergebnisse von jeweils 5 Optimierungsdurchläufen durch PSO und CMAES bei 5000 Iterationen

6.2.3 Versuch mit 5000 Schritten

Dieser und die folgenden Versuche benutzen den unter 5.1 beschriebenen Versuchsaufbau. In diesem Versuch führt der Schwarm 5000 Bewegungen aus, genauso wie CMAES 5000 Iterationen durchführt. Bei Generation 1 (Abbildung 6.4) kann man erkennen, dass PSO etwa für das beste Individuum ein ähnlich gutes Ergebnis erzielt wie CMAES. Für die anderen Individuen ergeben manche Durchläufe ähnliche Ergebnisse. Im Großen und Ganzen ist der Durchschnittswert von CMAES allerdings besser als PSO.

Bei Generation 15 kann man bei einem Lauf von PSO für spätere Strukturen ebenso Ausreißer beobachten, wie bei CMAES. Auch sonst liegen CMAES und PSO bei den Ergebnissen nahezu gleich auf, wobei hier ein Lauf von PSO besser ist als der CMAES-Durchschnitt.

CMAES erzielt bei Generation 150 allerdings klar bessere Ergebnisse als PSO.

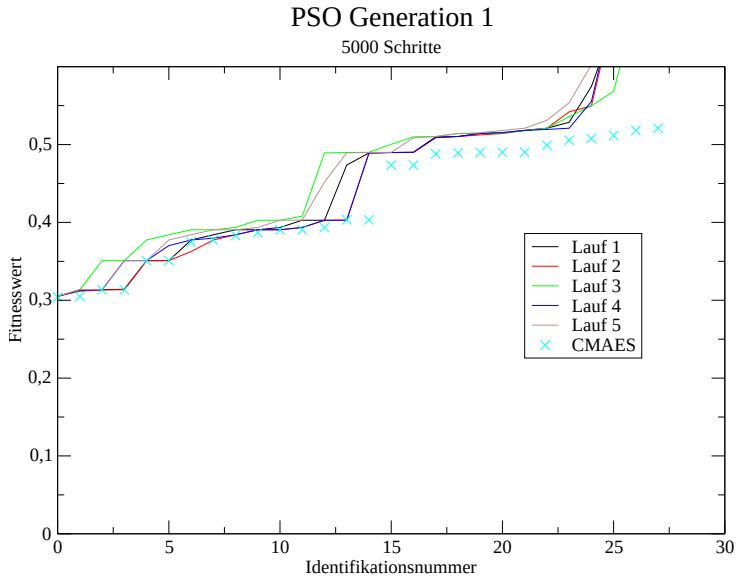


Abbildung 6.4: Fitnesswerte der 1. Generation. Optimiert in 5000 Schritten durch PSO

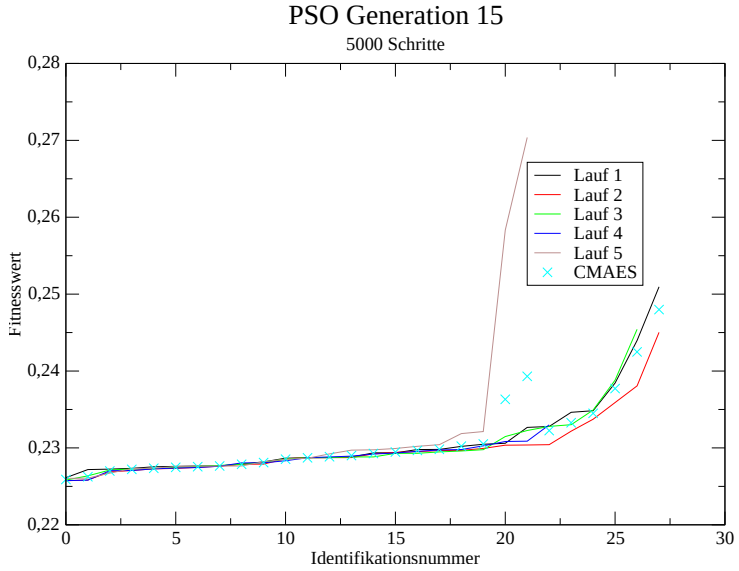


Abbildung 6.5: Fitnesswerte der 15. Generation. Optimiert in 5000 Schritten durch PSO

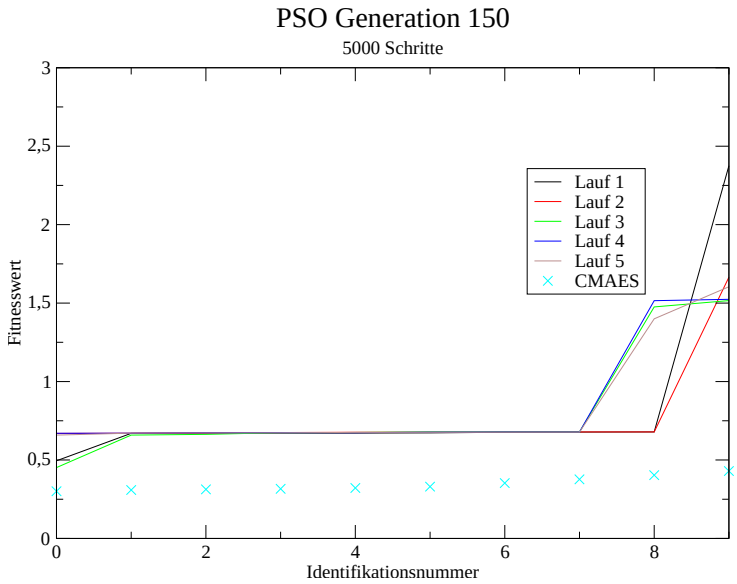
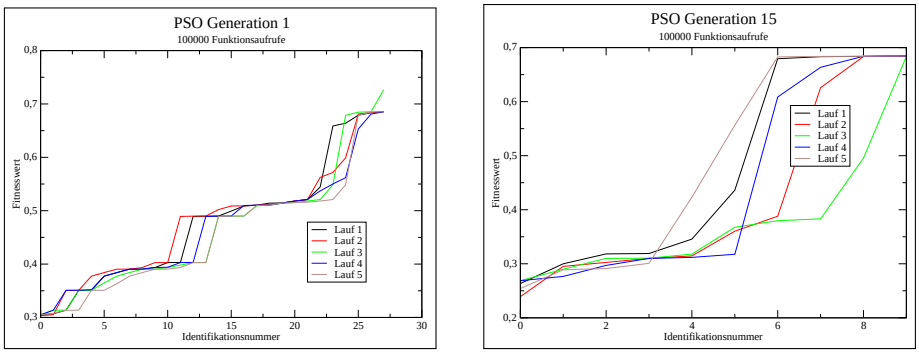


Abbildung 6.6: Fitnesswerte der 150. Generation. Optimiert in 5000 Schritten durch PSO

6.2.4 Versuch mit 100000 Funktionsaufrufen

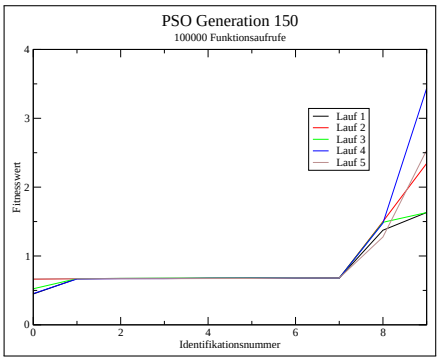
In obigem Versuch wurde die Anzahl der Iterationen der beiden Verfahren auf einen gleichen Wert gesetzt. Da sich allerdings die Populationsgrößen bei den beiden Verfahren unterscheiden, führen sie bei gleicher Iterationsanzahl unterschiedlich viele Fitnessfunktionsaufrufe aus. Die folgenden Ergebnisse wurden deshalb mit auf 100000 Funktionsevaluationen begrenzten Durchläufen optimiert. Dies entspricht, je nach Netz, zwischen 1694 bis 5882 Iterationen für PSO und gerade einmal 221 bis 5001 Iterationen für CMAES.

Die Abbildungen in 6.7 zeigen die Ergebnisse für den Partikelschwarm. Sie entsprechen bei Generation 1 und 150 etwa denen des vorherigen Versuchs. Bei Generation 15 sind die Ergebnisse deutlich gestreuter und schlechter, was auf die größeren Netze zurückzuführen ist, da hier PSO mit 100000 Evaluationen nicht die 5000 Schritte des vorherigen Versuchs erreicht.



(a) Generation 1

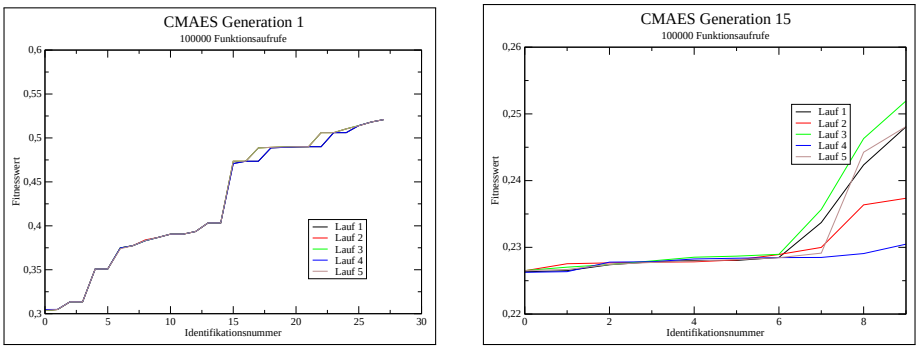
(b) Generation 15



(c) Generation 150

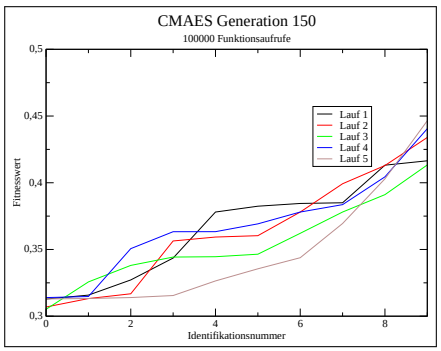
Abbildung 6.7: Ergebnisse PSO bei 100000 Funktionsevaluationen

Bei CMAES (Abbildung 6.8) unterscheiden sich die Ergebnisse der einzelnen Läufe bei Generation 1 fast gar nicht und auch bei Generation 15 nur bei den schlechteren Strukturen. Generation 150 zeigt auch hier wieder deutliche Unterschiede, allerdings liegen die Ergebnisse der besten Netze anders als bei PSO fast auf gleicher Höhe.



(a) Generation 1

(b) Generation 15



(c) Generation 150

Abbildung 6.8: Ergebnisse CMAES bei 100000 Funktionsevaluationen

Betrachtet man die Durchschnittswerte von PSO und CMAES im direkten Vergleich (Abbildungen 6.9-6.11), so sieht man, dass CMAES praktisch die gleichen Ergebnisse erzielt, wie mit 5000 Iterationen und fast immer besser ist als PSO. In Tabelle 6.2 sieht man allerdings, das PSO für Generation 1 tatsächlich ein besseres Ergebnis erreicht hat als CMAES (0, 3034 zu 0, 3038).

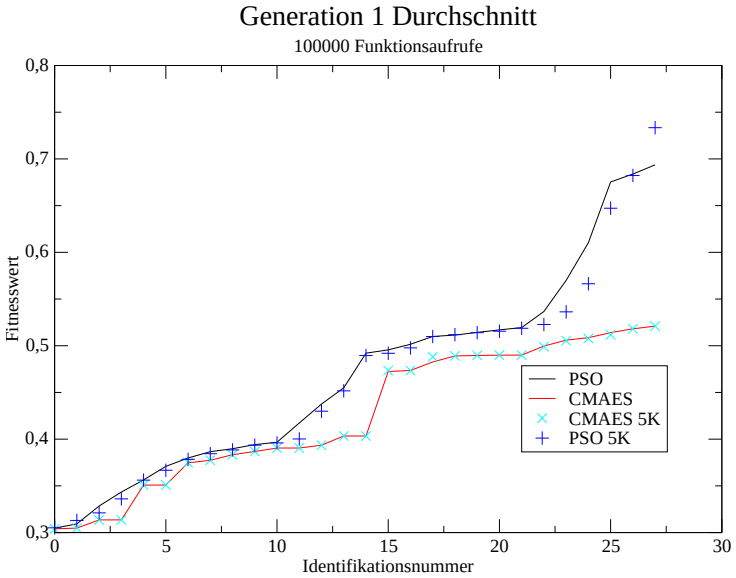


Abbildung 6.9: Durchschnittswerte Generation 1 PSO und CMAES bei 100000 Funktionsaufrufen.

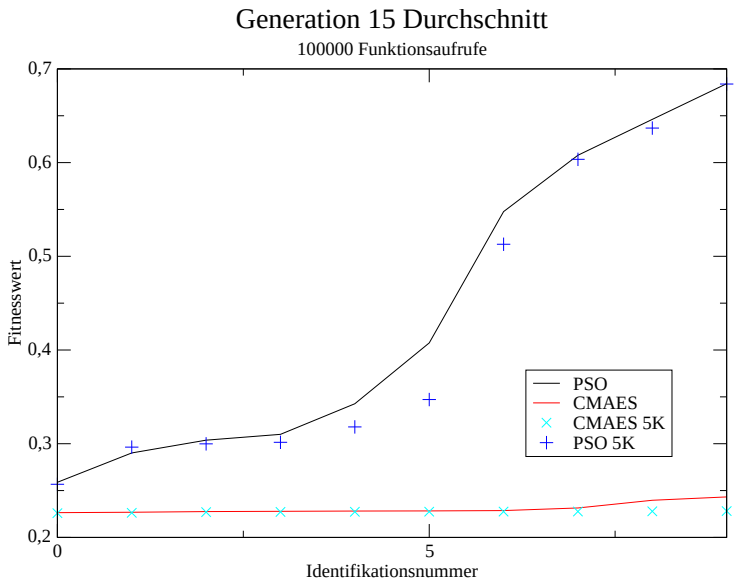


Abbildung 6.10: Durchschnittswerte Generation 15 PSO und CMAES bei 100000 Funktionsaufrufen.

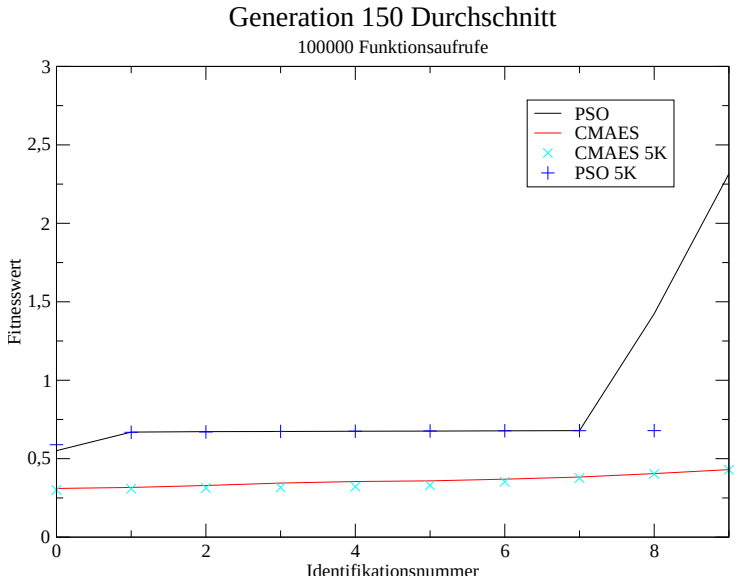


Abbildung 6.11: Durchschnittswerte Generation 150 PSO und CMAES bei 100000 Funktionsaufrufen.

PSO			
Generation	Min	Max	∅
1	0.303397892268	0.726430578161	0.467892296982
15	0.239151512531	0.684510571676	0.439898770306
150	0.447950635714	3.43500466839	0.90129707232
CMAES			
Generation	Min	Max	∅
1	0.303769695917	0.520949628965	0.424831460829
15	0.226254792114	0.251943463977	0.230791078245
150	0.305178458008	0.446488963929	0.360140919484

Tabelle 6.2: Ergebnisse (bestes, schlechtestes und Durchschnitt) von jeweils 5 Optimierungsdurchläufen durch PSO bzw. CMAES bei 100000 Funktionsauswertungen

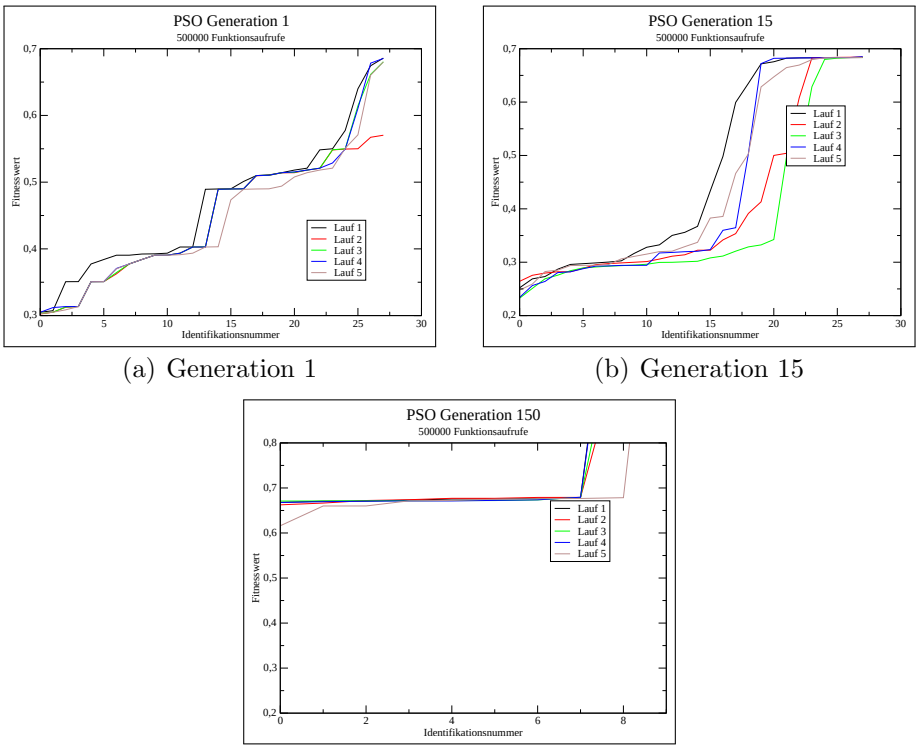
6.2.5 Versuch mit 500000 Funktionsaufrufen

Bei diesem Versuch wird die Begrenzung der maximalen Funktionsevaluationen auf 500000 angehoben. Ansonsten entspricht der Versuch dem Vorherigen. Mit dieser Evaluationsanzahl erreicht PSO 7936 bis 29411 Schritte, während CMAES 2501 bis 25001 Iterationen durchführen kann.

Durch die höhere Anzahl an Schritten, die der Schwarm durchführen kann, gleichen sich die Durchläufe von PSO, die in Abbildung 6.12 dargestellt sind, gegenseitig mehr an, als bei dem Durchlauf mit 10^5 Iterationen. Bei Generation 1 beobachtet man, dass alle Durchläufe fast den gleichen Wert für das beste Individuum erreichen. Bis auf Lauf 1 und später Lauf 5 verhalten sie sich auch über weite Strecken gleich.

Generation 150 weist deutlich weniger Streuung auf, als bei weniger Iterationen. Allerdings gibt es hier deutliche Abweichungen für das Ergebnis des besten Individuums. Nur 2 Läufe erreichen ein Ergebnis von unter $0,25$.

Für Generation 150 verlaufen alle Läufe bis auf einen nahezu identisch. Lauf 5 erreicht für die best Struktur sowie für die schlechtesten Strukturen bessere Ergebnisse, als die anderen Läufe.



(a) Generation 1

(b) Generation 15

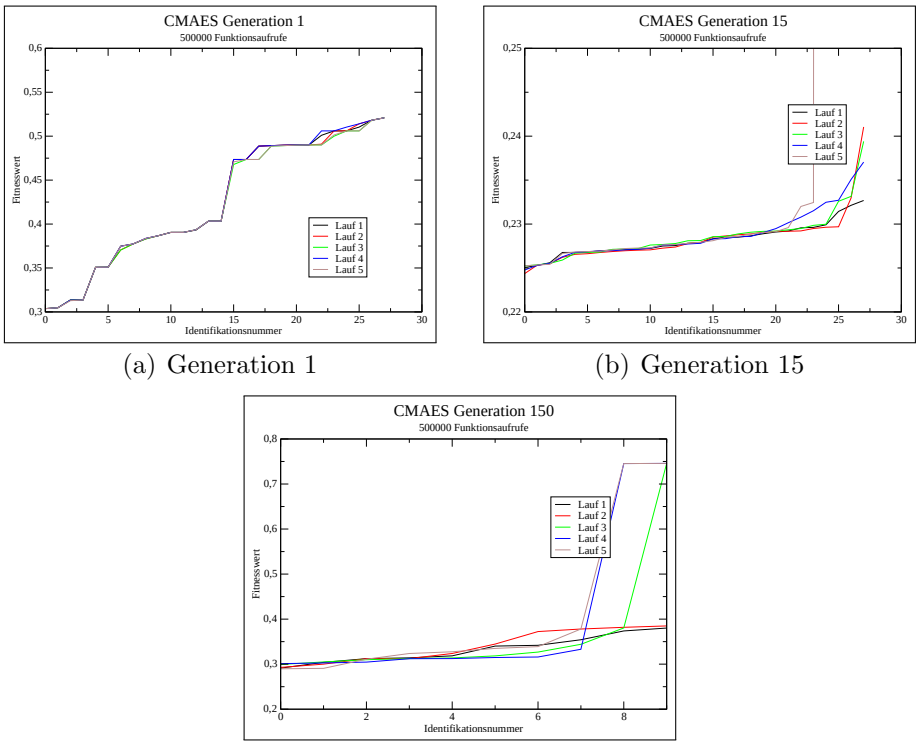
(c) Generation 150

Abbildung 6.12: Ergebnisse PSO bei 500000 Funktionsevaluationen

In Generation 1 erzielen nahezu alle Laufe mit CMAES identische Ergebnisse (siehe Abbildung 6.13).

Auch bei Generation 15 zeigen sich kaum noch Abweichungen bei den besten Ergebnissen. Nur für die schlechteren Ergebnisse weichen die Läufe voneinander ab. Lauf 5 weicht übermäßig stark ab und erreicht Werte über $5,8$.

Selbst bei Generation 150 gibt es kaum Unterschiede zwischen den Läufen und alle erreichen ein ähnliches Ergebnis für die beste Struktur.



(a) Generation 1

(b) Generation 15

(c) Generation 150

Abbildung 6.13: Ergebnisse CMAES bei 500000 Funktionsevaluationen

Betrachtet man die durchschnittlichen Ergebnisse von PSO und CMAES im Vergleich (Abbildung 6.14), so sieht man, dass sich PSO bei Generation 1 leicht Verbessern konnte und der Abstand zu CMAES nur sehr gering ist. CMAES erzielt nahezu die gleichen Ergebnisse, wie in den anderen Versuchen. Auch mit dieser Funktionevaluationsanzahl erzielt PSO das beste Ergebnis mit $0,3024$ für diese Generation, während CMAES dasselbe Ergebnis erzielt, wie bei allen anderen Läufen. Evtl. ist hier ein lokales Minimum, das CMAES oft trifft oder nicht verlassen kann.

Bei Generation 15 (Abbildung 6.15) konnte sich PSO zwar gegenüber dem Durchlauf mit 5000 Schritten stark verbessern, allerdings sind die Ergebnisse von PSO trotzdem noch deutlich schlechter als die von CMAES.

Für CMAES sind keine nennenswerte Verbesserungen gegenüber dem ursprünglichem Lauf zu beobachten. Der sprunghafte Anstieg ab Struktur 23 liegt an den Ausreißern von Lauf 5.

Der Abstand zwischen CMAES und PSO ist bei Generation 150 besonders deutlich. Hier sind beide Optimierer nahezu konstant, doch das Ergebnis von CMAES ist doppelt so gut, wie das von PSO.

Bei diesem Durchlauf hat CMAES im Durchschnitt 3440 Sekunden pro Individuum zur Optimierung gebraucht, PSO nur 2611. Allerdings hat der Referenzdurchlauf von CMAES nur 910 Sekunden benötigt und trotzdem bessere Ergebnisse erzielt, als PSO in 2611. Somit ist PSO zwar schneller, CMAES aber, in diesem Fall, effizienter.

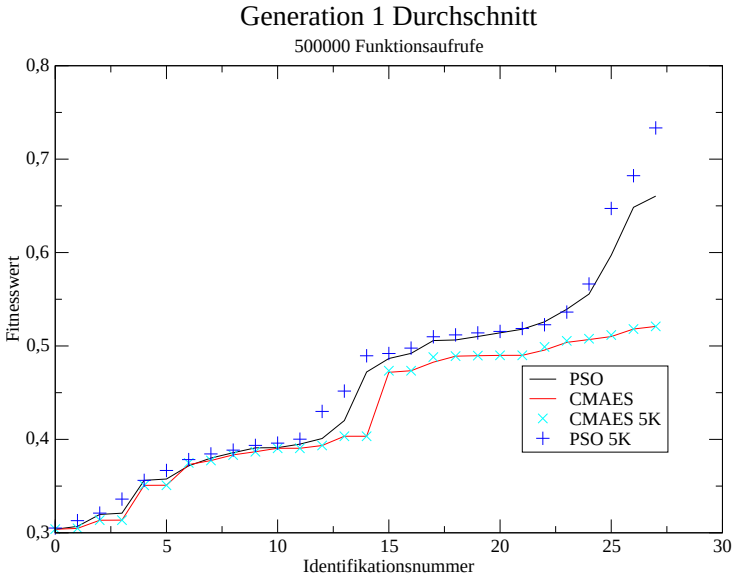


Abbildung 6.14: Durchschnittswerte Generation 1 PSO und CMAES bei 500000 Funktionsaufrufen.

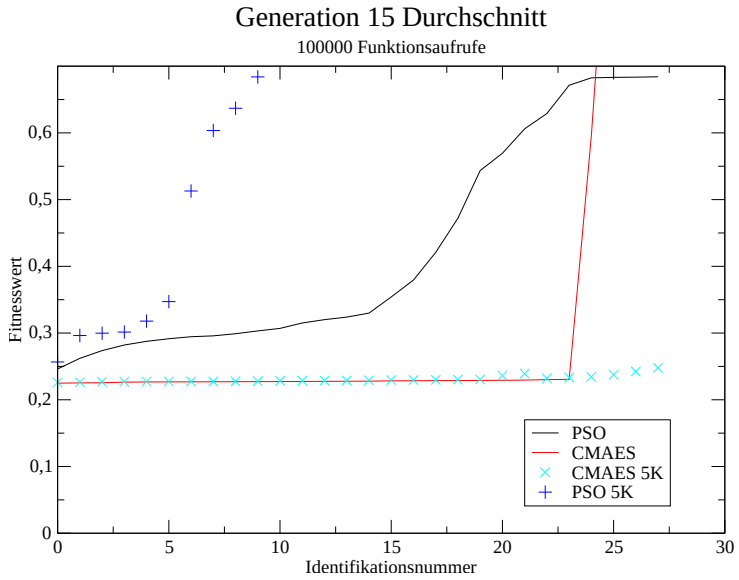


Abbildung 6.15: Durchschnittswerte Generation 15 PSO und CMAES bei 500000 Funktionsaufrufen.

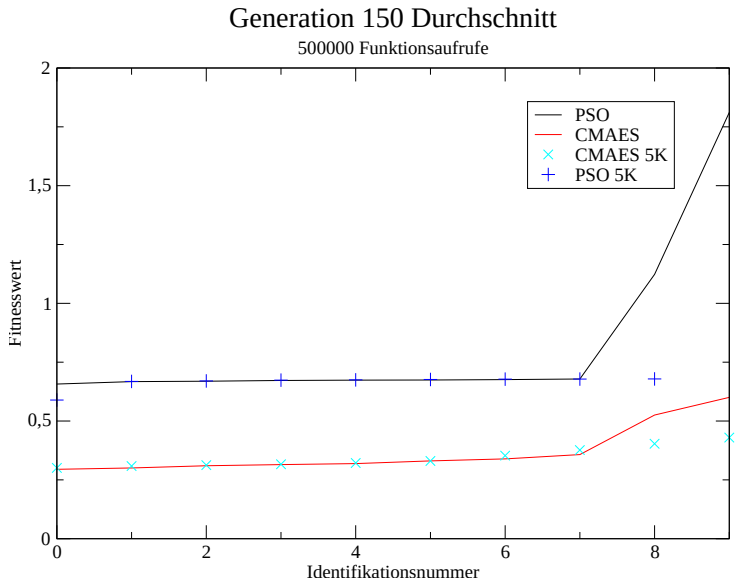


Abbildung 6.16: Durchschnittswerte Generation 150 PSO und CMAES bei 500000 Funktionsaufrufen.

PSO			
Generation	Min	Max	∅
1	0.302387680231	0.685938433957	0.451150789392
15	0.232636295125	0.685041048719	0.421824237152
150	0.616029937794	2.6158383559	0.830239013283
CMAES			
1	0.303769695913	0.520949628965	0.424318725322
15	0.224365711092	5.87944014194	0.347819241703
150	0.289703864694	0.745720387998	0.369241994457
Generation	Min	Max	∅

Tabelle 6.3: Ergebnisse (bestes, schlechtestes und Durchschnitt) von jeweils 5 Durchläufen Optimierung durch PSO bzw. CMAES bei 500000 Funktionsauswertungen

6.3 Kantenerkennung

6.3.1 Versuchsaufbau

Bei diesem Versuch wird diesmal ein kompletter Durchlauf von EANT2, angewandt auf das Kantendetektionsproblem, durchgeführt. Dabei werden jeweils 17 Generationen generiert. Als Parameteroptimierer werden CMAES und PSO gewählt, die nach.jeweils 5000 Schritten die Optimierung abbrechen.

6.3.2 Ergebnis

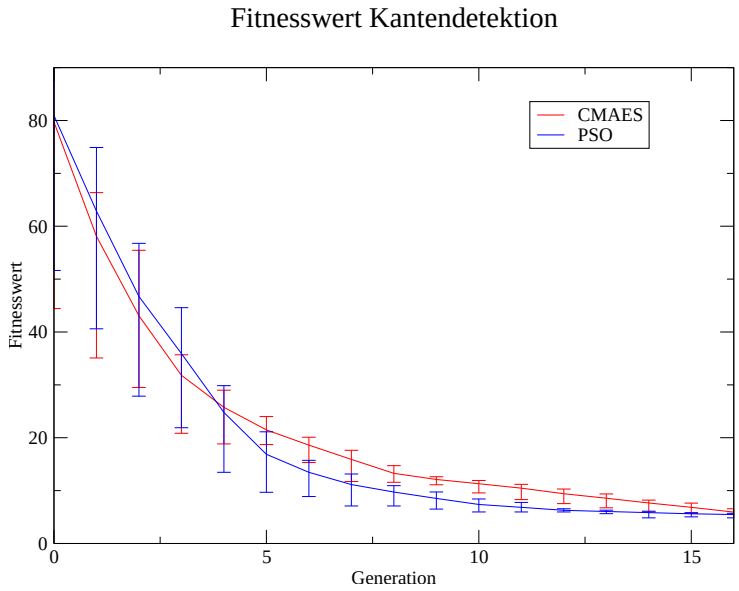


Abbildung 6.17: Fitnesswerte bei CMAES und PSO für die Kantendetektion

Das Ergebnis ist in Abbildung 6.17 zu sehen. Man sieht, dass beide Verfahren gute Ergebnisse erzielen. CMAES ist in den Generationen 0 bis 3 besser, PSO dafür in den Generationen 4 bis 16. Am Ende erreichen sie sehr ähnliche Werte.

Bei der Größe der Netzwerke (Abbildung 6.18) ist zu erkennen, das bei dem CMAES-Durchlauf ab etwa der Hälfte deutlich größere Netzwerke als bei PSO gewählt wurden.

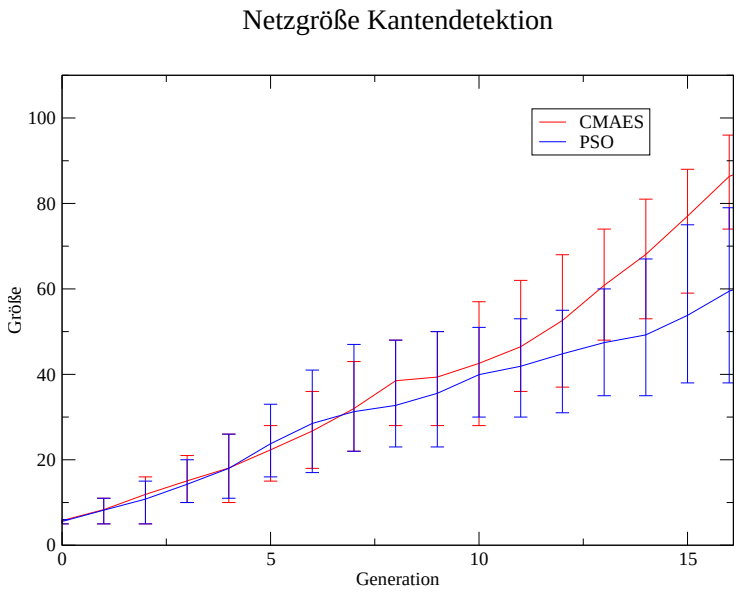


Abbildung 6.18: Netzgröße bei CMAES und PSO für die Kantendetektion

6.3.3 Fazit Kantendetektion mit PSO

In diesem Beispiel ist das Ergebnis von PSO klar besser als das von CMAES. PSO hat kleinere Netze generiert, die zudem bessere Ergebnisse erzielen. Aus dem Ergebnis dieses Versuchs kann allerdings nicht gefolgert werden, dass EANT2 mit PSO bessere Kantendetektoren generieren kann als CMAES. Ein besseres Ergebnis von PSO kann sowohl zufällig bedingt sein, als auch an den generierten Netzen liegen.

6.4 Fazit

Partikel-Schwarm-Optimierung ist ein Optimierungsverfahren, das in EANT2 genutzt werden kann und das erfolgreich Netzwerke optimieren kann. Allerdings erreicht PSO im Durchschnitt nicht die guten Ergebnisse von CMAES bei dem Visual-Servoing-Problem. Hier erreicht CMAES bei fast allen Läufen (zum Teil deutlich) bessere Ergebnisse, obwohl PSO eine Struktur besser optimieren konnte als CMAES.

Bei dem Kantendetektoren-Problem scheint PSO allerdings CMAES leicht überlegen zu sein. Hier wären weitere Versuche notwendig, um endgültig eine Aussage treffen zu können, ob sich dieser Trend weiter fortsetzt.

Somit kann PSO als gute Ergänzung zu CMAES angesehen werden, sollte

in EANT2 aber nicht als Ersatz dienen. Denkbar wäre auch eine Lösung, bei der EANT2 nicht jede Struktur zweimal mit CMAES optimiert, sondern einmal mit CMAES und einmal mit PSO.

Kapitel 7

Vergrößerung der Population der Parameteroptimierung

Ein wichtiger Parameter für Evolutionäre Algorithmen wie EANT2 und CMAES ist die Größe der Population. Kern und Hansen haben in [HK04] gezeigt, dass eine erhöhte Populationsgröße bei multimodalen Problemen die Qualität der Ergebnisse steigern kann. Die bei EANT2 verwendete Populationsgröße λ bei einem Problem mit Dimension d für CMAES liegt bei

$$\lambda = 5 + \lfloor 5 \cdot \log(d) \rfloor$$

7.1 Versuchsaufbau

Auch in diesem Experiment findet der Versuchsaufbau aus Abschnitt 5.1 Anwendung, da es sich um reine Modifikation der Parameteroptimierung handelt.

Als Referenz dienen wieder die CMAES-Durchläufe mit 5000 Iterationen aus Abschnitt 5.1.1.

Um den Einfluss der Populationsgröße zu untersuchen, wurden Versuche mit fünffacher und zehnfacher Populationsgröße durchgeführt. Dafür wurde EANT2 erweitert, um aus der NeuroEvolution.xml einen optionalen Parameter m auszulesen, der einen Parameter-Optimierungs-Populationsgrößen-Multiplikator enthält. Es ergibt sich somit als neue Populationsgröße:

$$\lambda = m \cdot (5 + \lfloor 5 \cdot \log(d) \rfloor)$$

Populationsgrößen-Multiplikator $m = 5$			
Generation	Min	Max	∅
1	0.303209531774	0.70201287717	0.436237726784
15	0.224462872098	2.95929760978	0.281983464977
150	0.299364836802	5.89619731443	0.509333589543
Populationsgrößen-Multiplikator $m = 1$			
Generation	Min	Max	∅
1	0.303769695913	0.520949628965	0.424919539821
15	0.225729577421	0.270374979058	0.23030684742
150	0.289847592718	0.437439525049	0.345333504834

Tabelle 7.1: Ergebnisse von jeweils 5 Durchläufen Optimierung mit einfacher bzw. fünffacher Populationsgröße

7.1.1 Fünffache Populationsgröße

Bei diesen 5 Durchgängen wurde die Populationsgröße gegenüber der Referenz verfünffacht. Es wurden ebenfalls 5000 Iterationen durchgeführt. Für Generation 1 schafft CMAES es, mit vergrößerter Populationsgröße ein geringfügig besseres Ergebniss zu erreichen, als mit Einfacher. Wie man in Tabelle 6.3 sieht, konnte CMAES auch in dem Lauf mit einer Beschränkung auf $5 \cdot 10^5$ Funktionsaufrufen diesen Wert nicht unterbieten, was an internen Abbruchkriterien liegt.

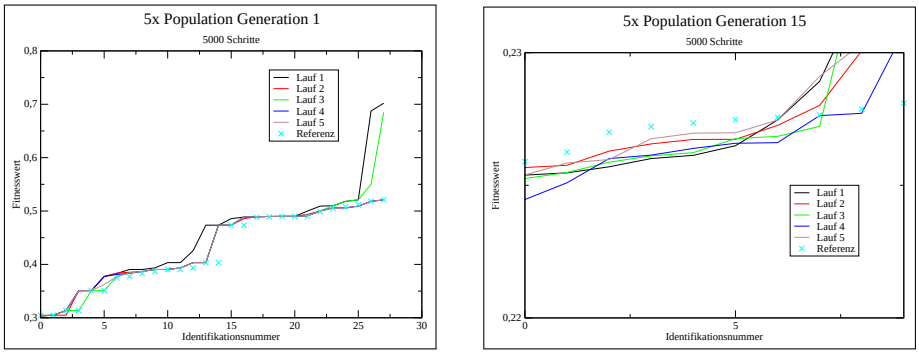
Ebenfalls eine geringe Verbesserung auf $0,22446$ konnte durch die Vergrößerung der Population bei Generation 15 erzielt werden, allerdings konnte hier der CMAES-Durchlauf mit $5 \cdot 10^5$ Funktionsaufrufen ein noch besseres Ergebnis mit $0,22437$ erzielen.

Überraschenderweise hat sich das Ergebnis für Generation 150 durch die Erhöhung der Population von $0,2898$ nach $0,2994$ verschlechtert.

Betrachtet man die einzelnen Läufe in Abbildung 7.1 sieht man für Generation 1 wieder einen fast identischen Verlauf, allerdings mit leichter schlechterer Tendenz als der Referenzlauf.

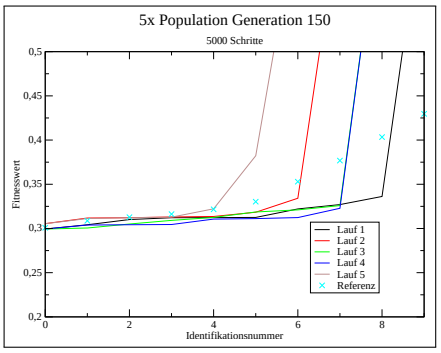
Bei Generation 15 sind die Optimierungen mit höherer Populationsgröße bis auf Ausreißer am Ende besser als mit normaler Größe.

Für Generation 150 zeigt sich ein gemischtes Bild mit Läufen, die deutlich schlechter als der Durchschnitt der Referenz sind und solchen, die besser sind.



(a) Generation 1

(b) Generation 15



(c) Generation 150

Abbildung 7.1: Ergebnisse CMAES mit 5x vergrößerter Population bei 5000 Iterationen

Populationsgrößenmultiplikator $m = 10$			
Generation	Min	Max	∅
1	0.303210827908	5.72464335667	0.489981428067
15	0.216943920321	5.87944014194	0.50286933733
150	0.299364851538	0.679141188189	0.404599195529
Populationsgrößenmultiplikator $m = 1$			
Generation	Min	Max	∅
1	0.303769695913	0.520949628965	0.424919539821
15	0.225729577421	0.270374979058	0.23030684742
150	0.289847592718	0.437439525049	0.345333504834

Tabelle 7.2: Ergebnisse von jeweils 5 Durchläufen Optimierung mit einfacher bzw. zehnfacher Populationsgröße

7.1.2 Zehnfache Populationsgröße

In diesem Versuch wurde die Population nochmal vergrößert, so dass sie nun zehnmal so groß ist, wie bei den Standardeinstellungen.

Wie man an Tabelle 7.2 sehen kann, wurde mit der Vergrößerung bei Generation 1 wieder ein besseres Ergebnis erzielt. Allerdings war das Ergebnis bei $m = 5$ noch etwas besser.

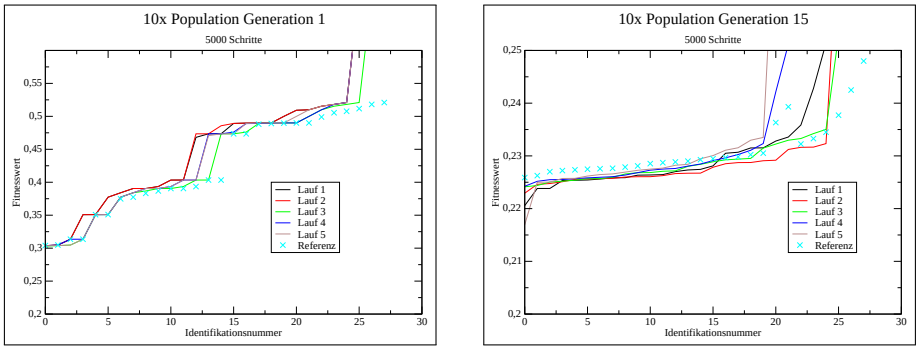
Bei Generation 15 konnte allerdings das Ergebnis deutlich von $0,226$ auf $0,217$ verbessert werden. Die deutliche Verbesserung hier könnte auf die hohe Dimensionalität der Netzwerke dieser Generation zurückzuführen sein.

Und auch bei dieser Populationsgröße kann man wieder eine Verschlechterung bei Generation 150 feststellen.

Bei der Übersicht der einzelnen Strukturen (Abbildung 7.2) kann man in Generation 1 auch wieder bei den meisten Strukturen eher eine geringe Verschlechterung gegenüber der Referenz beobachten.

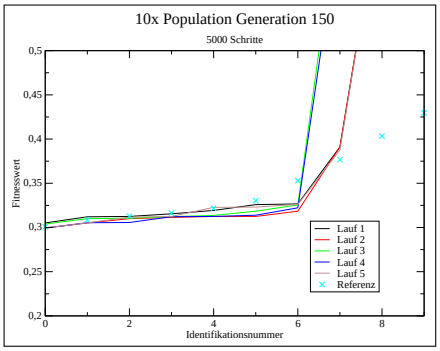
Im Gegensatz dazu sind die Ergebnisse bei Generation 15 für die meisten Strukturen besser als bei der Referenz.

Obwohl Struktur 0 bei Generation 150 schlechter ist, als bei normaler Größe, sind die Ergebnisse der Strukturen 1 bis 6 durchgehend besser als bei der Referenz.



(a) Generation 1

(b) Generation 15



(c) Generation 150

Abbildung 7.2: Ergebnisse CMAES mit 10x vergrößerter Population bei 5000 Iterationen

7.1.3 Zusammenfassung und Fazit

Die Abbildungen 7.3 bis 7.5 zeigen die Durchschnittswerte der 5 Läufe der 3 Versuche.

Bei Generation 1 konnte zwar ein besseres Ergebnis mit beiden vergrößerten Populationen erzielt werden, allerdings wurden die Ergebnisse im Durchschnitt nicht besser. Die Umgekehrte Situation tritt bei Generation 150 auf, bei der die beste Struktur schlechter geworden ist, die meisten anderen aber besser.

Wichtig ist allerdings das Ergebnis für Generation 15. Hier ist bei einer Vergrößerung der Population um den Faktor 10 das gesamte Ergebnis besser geworden. Die hier verwendeten Netze sind mit ihrer Größe auch durchaus typisch für EANT2-Durchläufe.

Für die meisten Ergebnisse ist die Verbesserung sehr gering gegenüber dem Verlust an Geschwindigkeit, da sich eine Erhöhung der Population auch direkt auf die Anzahl der Funktionsaufrufe und somit auf die Laufzeit auswirkt. Also bietet die Vergrößerung der Population des Parameteroptimierers eine Möglichkeit, bessere Ergebnisse zu erzielen, aber es bietet keine Garantie auf Verbesserung.

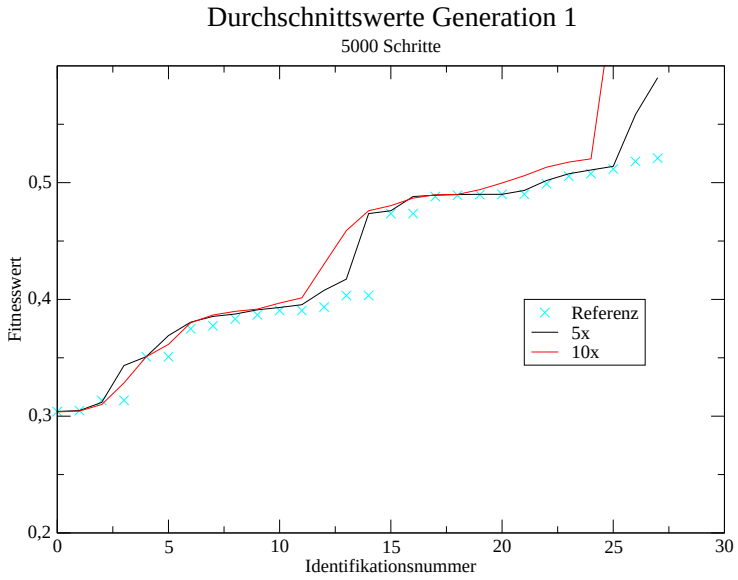


Abbildung 7.3: Durchschnittswerte Generation 1, Populationsgrößen $1 \cdot \lambda$, $5 \cdot \lambda$ und $10 \cdot \lambda$

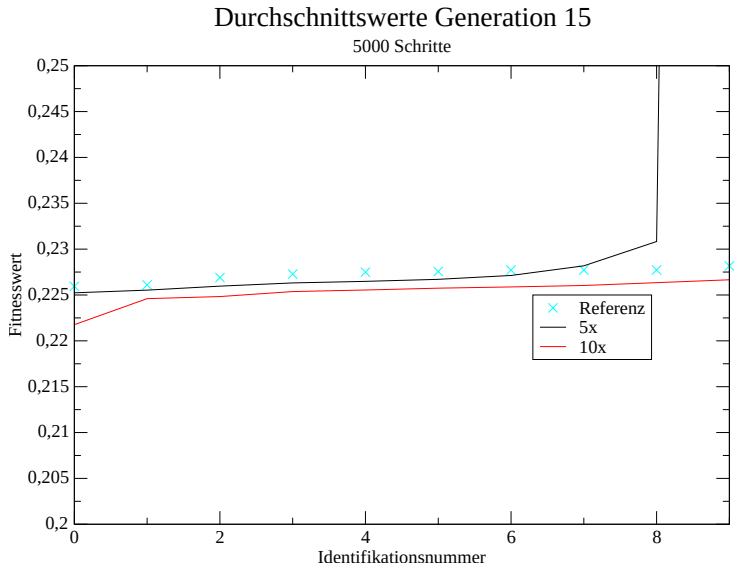


Abbildung 7.4: Durchschnittswerte Generation 15, Populationsgrößen $1 \cdot \lambda$, $5 \cdot \lambda$ und $10 \cdot \lambda$

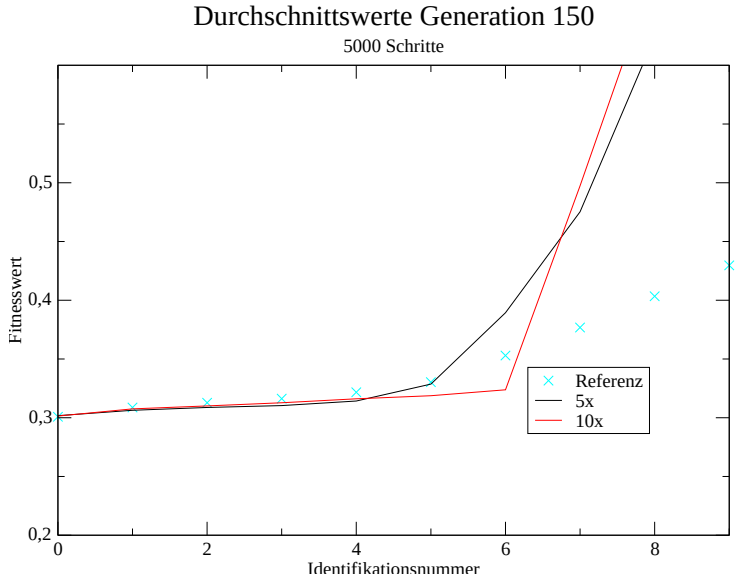


Abbildung 7.5: Durchschnittswerte Generation 150, Populationsgrößen $1 \cdot \lambda$, $5 \cdot \lambda$ und $10 \cdot \lambda$

Kapitel 8

CMAES mit lokalem Metamodell

Bei Evolutionären Methoden ist eine Auswertung einer Fitnessfunktion häufig eine vergleichsweise zeit- und rechenaufwändige Aufgabe, die häufig ausgeführt werden muss. Eine naheliegende Methode um EANT zu beschleunigen ist es also, die Anzahl der Fitnessfunktionsevaluationen zu reduzieren. In [KHK06] wird eine Methode vorgestellt, die genau hierfür geeignet ist. Zur Reduktion wird hier die Fitnessfunktion mittels **lokaler Metamodelle (LMM)** dargestellt und somit nur zum Teil wirklich evaluiert und ansonsten interpoliert bzw. extrapoliert. Dabei hat sich gezeigt, dass mit Hilfe von LMM-CMAES die Anzahl der Fitnessfunktionsaufrufe teilweise drastisch reduziert werden kann (f_{Schwefel} mit $n = 16$ von 5263 auf 626).

8.1 Lokal gewichtete Regression

Eine Einführung in **lokal gewichtete Regression** (engl. locally weighted regression (**LWR**)) geben unter anderem [Cle79] und [AMS97], wobei im Folgenden die Notation aus [AMS97] übernommen wird.

Bei der lokal gewichteten Regression geht es darum, aus m bereits bekannten Paaren von Werten $(\mathbf{x}_i, \mathbf{y}_i)$ mit Hilfe eines Modells für eine Anfrage $\mathbf{q}$ eine möglichst passende Antwort zu generieren. Dabei gilt es, das Model so zu erstellen, dass ein bestimmtes Fehlerkriterium $C(\mathbf{q})$ minimiert wird. Das Modell ist dabei ein Polynom von geringer Ordnung $\hat{f}(\mathbf{x}_i, \boldsymbol{\beta})$, wobei $\boldsymbol{\beta}$ die Parameter des Modells (also die Koeffizienten des Polynoms) sind.

Die Besonderheit von LWR ist, dass nicht ein globales Modell für alle Anfragen benutzt wird, sondern lokale Modelle benutzt werden, bei denen Daten stärker berücksichtigt werden, die näher an der Anfrage liegen. Hierfür wird

eine Kernelfunktion K verwendet, die eine gewichtende und auswählende Funktion gegenüber den bekannten $\mathbf{x}_i$ übernimmt. Demzufolge ist sie in der Regel eine Funktion $K(d_q) : \mathbb{R} \to [0, 1]$, die bei von $\mathbf{q}$ entfernten Daten null wird und bei $\mathbf{q}$ gleich 1 ist. Der Kernel wird allerdings nicht direkt auf die Distanz ($d(\mathbf{q}, \mathbf{x}_i)$) von $\mathbf{q}$ zu den jeweiligen $\mathbf{x}_i$ angewandt, sondern es wird vorher noch eine Bandweite (auch Glättungsparameter) h berücksichtigt, die adaptiv angepasst wird und ein Maß für die Dichte der Datensätze um den Punkt $\mathbf{q}$ ist.

Somit ergibt sich als Ganzes ein zu minimierendes Kriterium

$$C(\mathbf{q}) = \sum_{i=1}^{m} \left[\left(\hat{f}(\mathbf{x}_i, \boldsymbol{\beta}) - \mathbf{y}_i \right)^2 K \left(\frac{d(\mathbf{q}, \mathbf{x}_i)}{h} \right) \right]$$

Unter der Voraussetzung, dass $\hat{f}$ linear zu $\boldsymbol{\beta}$ ist und somit $\hat{f}(\mathbf{x}_i, \boldsymbol{\beta}) = \tilde{\mathbf{x}}^T \boldsymbol{\beta}$ gilt[1], kann dieses Kriterium minimiert werden, indem die folgende Gleichung gelöst wird:

$$\left(\left(\mathbf{W}\tilde{\mathbf{X}} \right)^T \mathbf{W}\tilde{\mathbf{X}} \right) \boldsymbol{\beta} = \left(\mathbf{W}\tilde{\mathbf{X}} \right)^T \mathbf{W}\mathbf{y}$$

wobei $\tilde{\mathbf{X}} = (\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_m)^T$, $\mathbf{y} = (y_1, \ldots, y_m)^T$ und
$\mathbf{W} = \text{diag}\left(\sqrt{K(d((\tilde{\mathbf{x}}_i), \mathbf{q}/h)} \right)$ sind.

[1] Zur Wahl von $\tilde{\mathbf{x}}$ siehe für LMM-CMAES Gleichung 8.1

8.2 Local Meta-Model CMAES

LMM-CMAES unterscheidet sich von CMAES nur an einer Stelle. Wenn bei CMAES die Fitnessfunktion aufgerufen wird, um Individuen zu bewerten, wird stattdessen bei LMM-CMAES eine Methode aufgerufen, in welcher das Modell entwickelt und ausgewertet wird. Hierbei wird - je nach Qualität des Modells - nur teilweise die Fitnessfunktion benutzt, an anderen Stellen wird das Ergebnis mittels der lokalen Modelle ohne Benutzung der Fitnessfunktion erzeugt. Für CMAES findet dieser Vorgang vollständig transparent statt. Hierfür verwendet LMM-CMAES die oben vorgestellten Methoden mit folgenden Ausprägungen:

8.2.1 Modell

Als Modell verwendet LMM-CMAES ein quadratisches Modell mit Kreuztermen, also

$$\hat{f} = \boldsymbol{\beta}^T \left(x_1^2, \ldots, x_n^2, x_1 x_2, \ldots, x_{n-1} x_n, x_1, \ldots, x_n, 1\right)^T \tag{8.1}$$

und somit ergibt sich eine Dimensionalität für das Modell von

$$\frac{n\,(n+3)}{2} + 1, \tag{8.2}$$

wobei n die Dimension des Problems ist.

8.2.2 Kernel

Ein Biquadratischer Kernel bildet die Kernelfunktion:

$$K\,(d) = \begin{cases} \left(1 - d^2\right)^2 & \text{falls } d < 1 \\ 0 & \text{sonst} \end{cases}$$

8.2.3 Bandbreite

Die Bandbreite h wird dynamisch für jeden abgefragten Punkt $\mathbf{q}$ einzeln bestimmt, indem h auf die Distanz des k'ten nächstem Nachbarn von $\mathbf{q}$ gesetzt wird. Daraus ergibt sich, dass die Bandbreite mit größerer Datendichte sinkt und bei geringerer Datendichte steigt. Der optimale Wert für k wurde empirisch ermittelt und als

$$k = n\,(n+3) + 2$$

festgelegt.

8.3 LMM-CMAES für EANT2

In [KHK06] wurde gezeigt, dass LMM-CMAES die Anzahl der Funktionsevaluationen teilweise stark verringern kann. Allerdings wurden für EANT-Verhältnisse nur kleine Dimensionen (bei f_{Schwefel} und $f_{\text{Rosenbrock}}$ jeweils $n = 16$, bei $f_{\text{Rastrigin}}$ sogar nur $n = 10$) betrachtet. Da wie in (8.2) gesehen, allein die Dimension des Modells quadratisch zur Dimension des Problems ist, ist es besonders wichtig, das Verhalten von LMM-CMAES bei höheren Dimensionen zu untersuchen. Man beachte, dass in diesem Anwendungsfall die Dimension des von LMM-CMAES zu optimierenden Problems nicht die Dimension des von EANT zu lösenden Problems besitzt, sondern dass LMM die Anzahl der Parameter der generierten Netzwerke als Dimensionalität besitzt. Diese ist schon in der nullten Generation mindestens so groß, wie die des ursprünglichen Problems. Betrachtet man das Visual-Servoing-Problem, so führt allein ein vollständiges Netz ohne versteckte Neuronen bereits zu einer Dimension von 30 nur durch die Gewichte der Verbindungen der Eingänge zu den Ausgängen. Hinzu kommen noch eventuelle Gewichte der Ausgangsneuronen, versteckte Schichten, sowie Parameter der Neuronen selbst (z.B. σ der Aktivierungsfunktionen), sodass auch Netze mit über 100 Parametern in späteren Generationen keine Seltenheit sind. Da das Aufstellen und Lösen des Modells selber auch rechenintensiv ist, wurde im Folgenden LMM-CMAES vor allem auch auf die Gesamtdauer der Optimierung von Problemen untersucht.

8.3.1 Versuchsaufbau

Untersucht wird die Geschwindigkeit von LMM-CMAES im Vergleich zu CMAES bei den Problemen

$$f_{\text{Schwefel}}(x) = \sum_{1}^{n} \left(\sum_{i=1}^{n} x_i \right)^2,$$

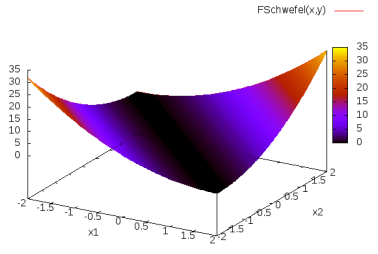
die auch als Rotierte Hyper-Ellipsoid-Funktion bekannt ist,

$$f_{\text{Rosenbrock}}(x) = 100 \cdot \sum_{i=1}^{n-1} \left(\left(x_i^2 - x_{i+1}^2 \right)^2 + (x_i - 1)^2 \right),$$

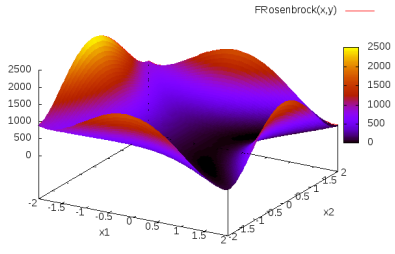
wegen ihres bananenförmigen Tals auch „Bananen-Funktion“genannt und

$$f_{\text{Rastrigin}}(x) = 10 \cdot \left(n - \sum_{i=1}^{n} \cos(2\pi x_i) \right) + \sum_{i=1}^{n} x_i^2$$

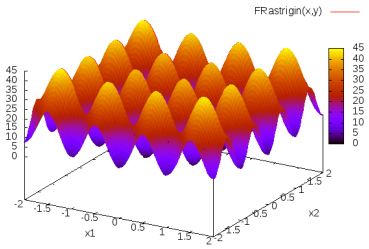
die durch ihre vielen lokalen Minima eine schwer zu optimierende, multimodale Funktion ist.



(a)



(b)



(c)

Abbildung 8.1: Plot der zu optimierenden Funktionen: f_{Schwefel} (a) ,$f_{\text{Rosenbrock}}$ (b) und $f_{\text{Rastrigin}}$ (c)

Diese drei Funktionen werden von beiden Optimierern bei aufsteigenden Dimensionen jeweils 5 mal optimiert bis sie konvergieren. Als Konvergenzkriterium wird jeweils eine Veränderung von weniger als 10^{-6} angenommen. Benutzt wird der original Code von LMM-CMAES. Als CMAES-Implementierung wird der in LMM enthaltene Code benutzt, bei dem der Aufruf von LMM durch die original Fitnessfunktion ersetzt wird. Beide Optimierer liegen als MATLAB-Programme vor, wobei ein Teil des Codes von LMM-CMAES in Fortran geschrieben und kompiliert ist. Dies ist zwar potentiell ein Vorteil für LMM-CMAES, der sich im Folgenden aber als unerheblich herausstellt. Aufgelistet finden sich in den folgenden Tabellen jeweils die Dimension n, die resultierenden Fitnesswerte, die Anzahl der Fitnessfunktionsaufrufe (Feval) und bei LMM-CMAES zusätzlich noch die Anzahl der virtuellen Fitnessfunktionsaufrufe (virt. Feval).

	LMM-CMAES				CMAES		
n	virt. Feval	Feval	Fitness	Laufzeit	Feval	Fitness	Laufzeit
2	389	80	9.08902e-10	1.505	513	1.29806e-16	1.623
5	1117	174	6.10398e-10	2.819	1346	9.89092e-16	2.585
10	2592	342	2.97969e-09	15.471	3066	1.54028e-15	3.664
15	4520	538	6.83864e-09	123.334	5526	2.33629e-15	5.737
20	6787	826	1.14394e-08	812.699	8027	3.62874e-15	8.518
25	9889	1138	3.841e-08	3787.277	11535	4.35024e-15	12.096
30	13010	1466	3.93902e-08	12326.788	15245	3.73597e-15	17.340

Tabelle 8.1: Durchschnittliches Ergebnisse von jeweils 5 Durchläufen Optimierung von f_{Schwefel} (Laufzeit in Sekunden)

8.3.2 Ergebnis f_{Schwefel}

In [KHK06] ist der Vorteil von LMM-CMAES gegenüber CMAES bei der Optimierung von f_{Schwefel} besonders deutlich. Bei einer Dimension von 2 benötigt LMM-CMAES nur 81 anstatt 391 Funktionsevaluationen. Bei $n = 16$ sinkt die Anzahl sogar von 5263 auf 626.

Dieses Verhalten kann in diesem Versuch, dessen Ergebnis in Tabelle 8.1 eingesehen werden kann, auch bestätigt werden. Hier setzt sich der Trend weiter fort, dass bei LMM-CMAES die Anzahl der Funktionsevaluationen bei steigender Dimension weniger stark ansteigt als bei CMAES. Bei dieser Funktion konvergiert LMM-CMAES nicht nur bei den tatsächlichen Fitnessauswertungen schneller, sondern auch wenn man die virtuellen Aufrufe von LMM mit den tatsächlichen von CMAES vergleicht (siehe Tabelle 8.1 bzw. 8.4).

Betrachtet man allerdings die Fitnesswerte (siehe Abbildung 8.3), so ist zu erkennen, dass LMM-CMAES durchgängig schlechter ist als CMAES.

Obwohl LMM-CMAES deutlich weniger tatsächliche Funktionsevaluationen benötigt als CMAES, ist die Laufzeit von LMM-CMAES bei steigender Dimension deutlich größer als die von CMAES, wie in Abbildung 8.4 zu sehen ist. Bei den Dimensionen 2 und 5 sind beide Verfahren noch etwa gleich schnell, bei $n > 5$ wird LMM-CMAES deutlich langsamer als CMAES. Dieser Abstand vergrößert sich mit jedem Dimensionsschritt sehr stark. Bei $n = 30$ benötigt CMAES 18 Sekunden um zu konvergieren. LMM-CMAES benötigt hierfür über 3 Stunden.

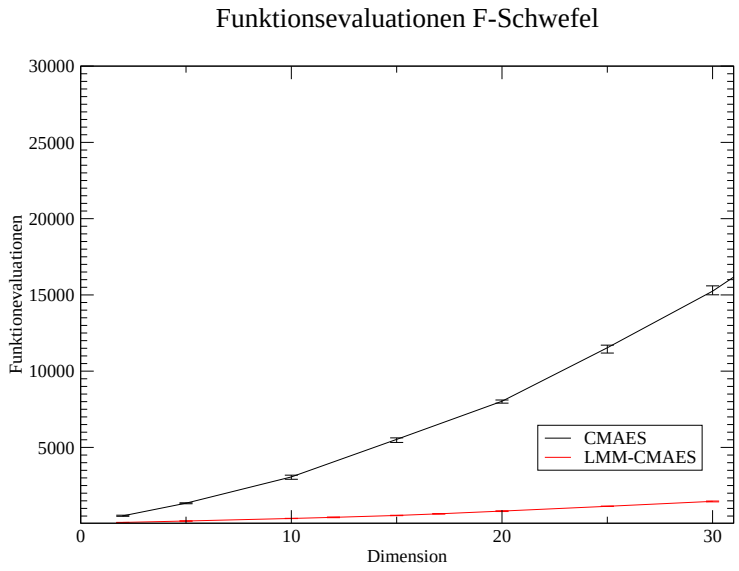


Abbildung 8.2: Funktionsevaluationen von CMAES und LMM-CMAES bei f_{Schwefel}

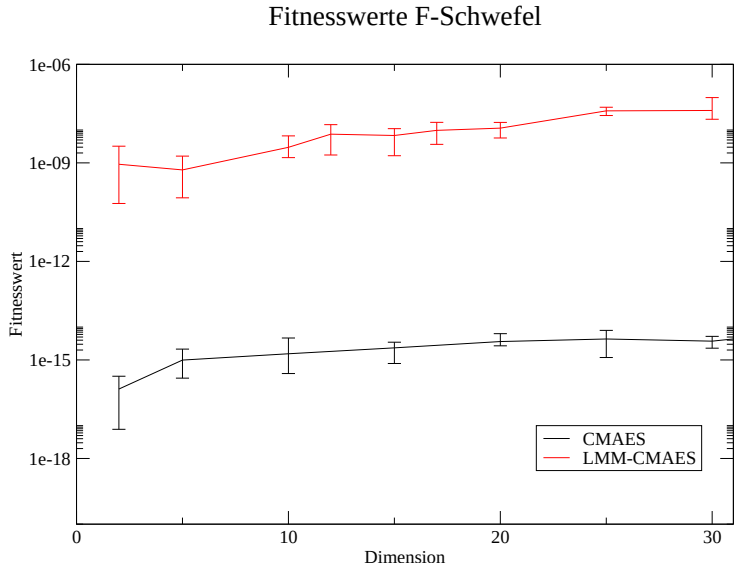


Abbildung 8.3: Fitnesswert von CMAES und LMM-CMAES bei f_{Schwefel}

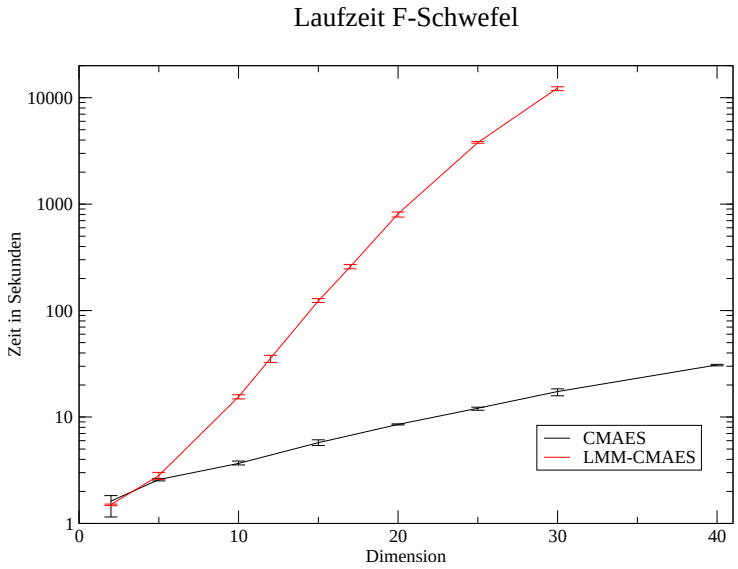


Abbildung 8.4: Laufzeit in logarithmischer Skala von CMAES und LMM-CMAES bei f_{Schwefel}

8.3.3 Ergebnis $f_{\text{Rosenbrock}}$

Auch bei der etwas komplexeren $f_{\text{Rosenbrock}}$ zeigt sich wieder, dass LMM-CMAES weniger Auswertungen der Fitnessfunktion benötigt als CMAES (Tabelle 8.2 und Abbildung 8.5).

Betrachtet man die Fitnesswerte in Abbildung 8.6, so stellt man fest, dass beide Verfahren starke Schwankungen aufweisen. Dies ist auf einzelne Durchläufe zurückzuführen, bei denen die Optimierer mit einer Fitness von rund $3,9$ konvergieren. Dieses Verhalten ist bei beiden Verfahren etwa gleich häufig zu beobachten. Ansonsten kann bei den Fitnesswerten keines der beiden Verfahren als besser erkannt werden.

Bei der Laufzeit verstärkt sich das Ergebnis von f_{Schwefel} noch deutlich und CMAES ist viel schneller als LMM-CMAES. Anders als bei f_{Schwefel} ist hier LMM-CMAES schon bei $n = 2$ langsamer als CMAES. Eine Optimierung bei $n = 25$ dauert mit LMM-CMAES rund 18 Stunden, mit CMAES nur 22 Sekunden. Aufgrund der langen Laufzeit von LMM-CMAES und dem extremen Anstieg der Laufzeit mit Ansteigen der Dimension, wurde hierbei verzichtet, LMM-CMAES mit $n = 30$ auszuführen.

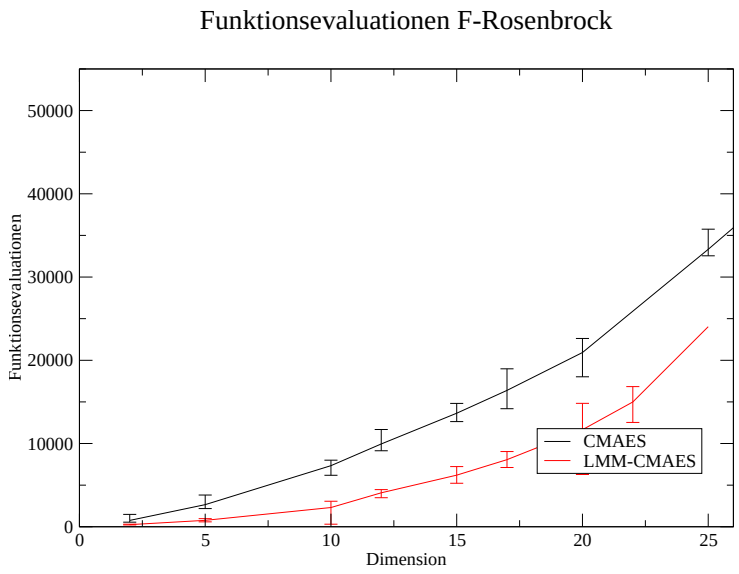


Abbildung 8.5: Funktionsevaluationen von CMAES und LMM-CMAES bei $f_{\text{Rosenbrock}}$

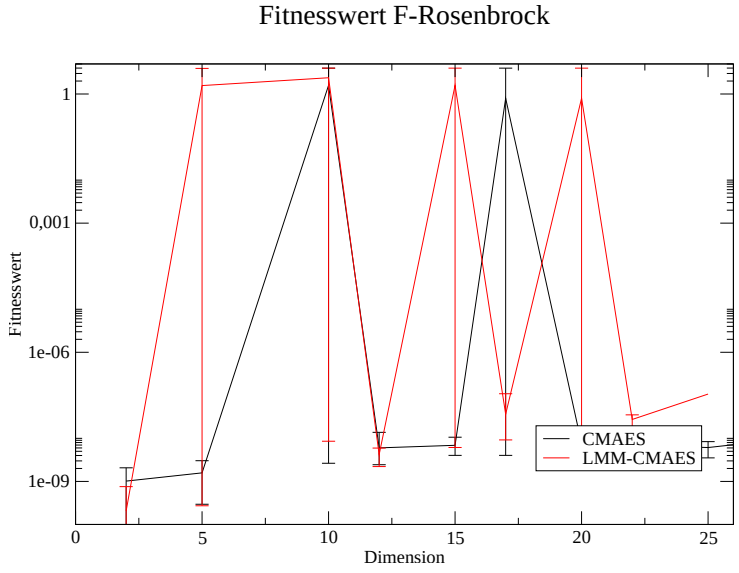


Abbildung 8.6: Fitnesswert von CMAES und LMM-CMAES bei $f_{\text{Rosenbrock}}$

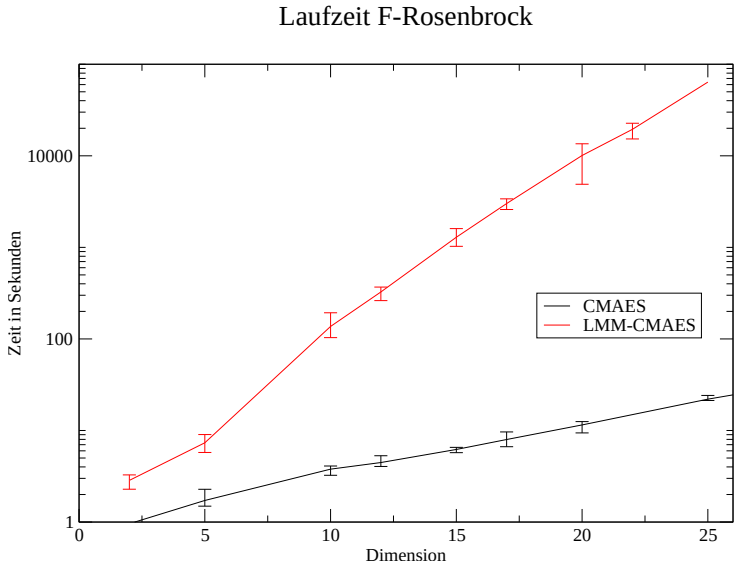


Abbildung 8.7: Laufzeit in logarithmischer Skala von CMAES und LMM-CMAES bei $f_{\text{Rosenbrock}}$

	LMM-CMAES				CMAES		
n	virt. Feval	Feval	Fitness	Laufzeit	Feval	Fitness	Laufzeit
2	998	253	2.06051e-10	2.849	760	1.01779e-09	0.959
5	3139	771	1.57234	7.339	2653	1.57672e-09	1.721
10	8780	2313	2.39195	136.859	7337	1.59463	3.786
12	14443	4076	4.20586e-09	325.753	9949	6.02164e-09	4.465
15	22494	6197	1.59465	1290.499	13642	6.89638e-09	6.205
17	29160	8063	3.64668e-08	3008.373	16383	0.797325	7.968
20	41751	11658	0.797325	10088.182	20932	7.54821e-09	11.493
25	79277	24036	1.07025e-07	63827.514	33333	6.12533e-09	21.981

Tabelle 8.2: Durchschnittliches Ergebnisse von jeweils 5 Durchläufen Optimierung von $f_{\text{Rosenbrock}}$ (Laufzeit in Sekunden)

	LMM-CMAES				CMAES		
n	virt. Feval	Feval	Fitness	Laufzeit	Feval	Fitness	Laufzeit
2	684	172	2.58689	2.070	724	2.18891	2.212
5	4619	1612	9.15361	15.015	1742	8.35764	3.133
10	13494	6201	16.5163	270.313	3592	17.9092	4.400
15	33744	19691	39.7982	3996.487	5610	22.0881	5.538
20	15258	6024	35.4868	2945.690	6098	30.4457	6.191
25	20262	8562	61.4883	11916.86	7315	50.7428	7.023

Tabelle 8.3: Durchschnittliches Ergebnisse von jeweils 5 Durchläufen Optimierung von $f_{\text{Rastrigin}}$ (Laufzeit in Sekunden)

8.3.4 Ergebnis $f_{\text{Rastrigin}}$

Bei der Optimierung von $f_{\text{Rastrigin}}$ tritt erstmals der Fall auf, das LMM-CMAES nicht weniger Funktionsevaluationen benötigt, als CMAES. Dies deckt sich nicht mit den Beobachtungen von Kern, der allerdings nicht die Standardpopulationsgröße verwendet, sondern eigene, größere Werte, die aus [HK04] stammen. Auffällig ist hierbei eine sehr breite Streuung besonders bei $n = 15$. Hier werden mehr Evaluationen benötigt, als bei einer Dimension von 25. Bei höheren Dimensionen gleicht sich LMM wieder CMAES an.

Betrachtet man die Fitnesswerte, die beide Funktionen erreichen, stellt man bei beiden Funktionen starke Schwankungen fest. Bei $n \leq 10$ ist LMM-CMAES geringfügig besser als CMAES, bei höheren Dimension dreht sich die Situation um. Allerdings ist der Unterschied zwischen den beiden Durchschnittswerten im Vergleich zu den Schwankungen relativ gering, so dass hier auch nur schwer eine Bewertung vorzunehmen ist.

Klar fällt allerdings auch hier die Bewertung der Laufzeit zugunsten von CMAES aus. Allerdings ist hier erstmals ein Einbruch der Laufzeit - und somit eine Verbesserung - zu beobachten. Bei Dimension 15 benötigt LMM-CMAES teilweise länger als bei Dimension 20. Dies ist allerdings auf die extrem hohe Anzahl an Optimierungsschritten zurückzuführen, die bei $n = 15$ benötigt wurden. So wurde LMM-CMAES in einem Durchlauf bei $n = 15$ 73153 mal aufgerufen, der größte Wert bei Dimension 20 lag bei 16899.

Man kann beobachten, dass LMM-CMAES bei $n = 15$ durchschnittlich 33744 mal und bei $n = 25$ nur 20262 aufgerufen wurde. Trotzdem benötigte LMM für $n = 25$ rund dreimal so lange. Daran kann man erkennen, dass die Laufzeit von LMM-CMAES stärker von der Dimension des Problems abhängt, als von der Anzahl der Aufrufe.

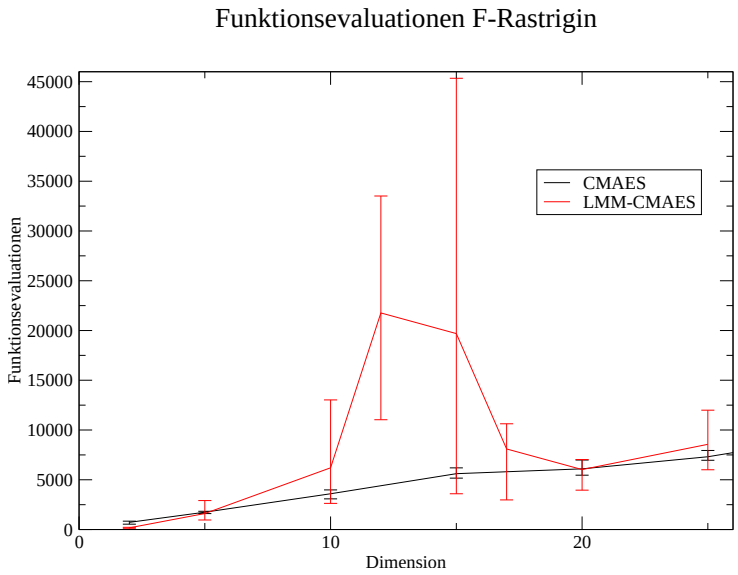


Abbildung 8.8: Funktionsevaluationen von CMAES und LMM-CMAES bei $f_{\text{Rastrigin}}$

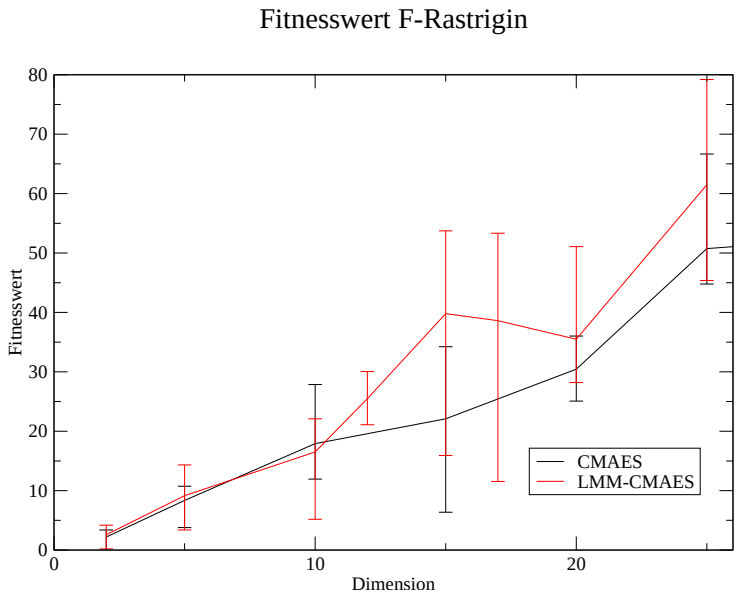


Abbildung 8.9: Fitnesswert von CMAES und LMM-CMAES bei $f_{\text{Rastrigin}}$

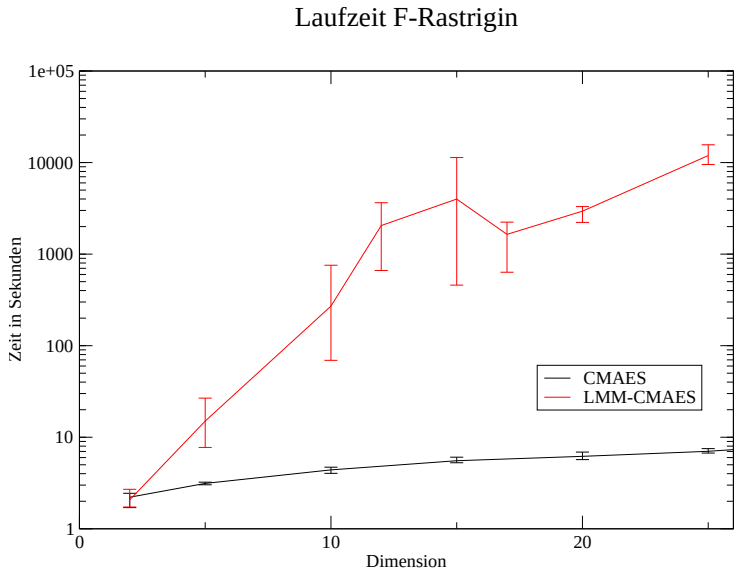


Abbildung 8.10: Laufzeit in logarithmischer Skala von CMAES und LMM-CMAES bei $f_{\text{Rastrigin}}$

8.4 Fazit

Auch wenn es mit LMM-CMAES möglich ist, die Anzahl der Fitnessfunktionsevaluationen teilweise deutlich zu reduzieren, scheint es keine geeignete Möglichkeit zu sein, EANT zu beschleunigen. Hiergegen sprechen vor allem die extrem langen Laufzeiten schon bei (für EANT) kleinen Dimensionen. Bei f_{Schwefel}, bei der LMM-CMAES das beste Optimierungspotential gezeigt hat, erreichte LMM-CMAES nicht die Fitness von CMAES. Bei $f_{\text{Rastrigin}}$ gab es nicht einmal eine Reduktion der Funktionsevaluationen unter Verwendung von CMAES-Normalparametern. Bei dieser Funktion kann LMM-CMAES also nicht schneller sein, als CMAES. Geht man pessimistisch davon aus, dass die von EANT zu optimierenden Funktionen eher wie $f_{\text{Rastrigin}}$ kompliziert und multimodal sind, so ist CMAES deutlich im Vorteil. Auch bei optimistischer Schätzung auf Grundlage von f_{Schwefel} steht der Verlust an Zeit durch das Aufstellen und Lösen des Modells in keinem Verhältnis zu den gewonnen Evaluationen.

8.4.1 Tabellarische Ergebnisse

Die Folgenden Tabellen enthalten die Ergebnisse der einzelnen Läufe der drei Funktionen.

Ergebnisse f_{Schwefel} LMM-CMAES				
n	virt. Feval	Feval	Fitness	Laufzeit
2	385	78	2.6252685e-10	1.498
2	401	83	9.767059e-11	1.542
2	385	78	5.8070111e-11	1.491
2	387	81	3.2173419e-09	1.490
5	1085	170	1.6092687e-09	2.773
5	1094	171	1.302639e-10	2.791
5	1157	179	8.6359818e-11	2.879
5	1013	161	4.7411124e-10	2.619
5	1238	189	7.5198655e-10	3.034
10	2573	340	1.6338784e-09	15.331
10	2496	333	1.4473257e-09	14.764
10	2562	339	6.631465e-09	15.255
10	2694	352	2.4191161e-09	16.156
10	2639	347	2.7666544e-09	15.851
12	3224	407	6.3487876e-09	34.251
12	3512	433	4.6082451e-09	37.908
12	3572	438	1.458445e-08	38.509
12	3200	405	1.7387356e-09	34.032
12	3128	399	1.0069181e-08	33.119
15	4567	542	9.8235719e-09	124.427
15	4528	539	3.5557026e-09	125.006
15	4333	522	1.6523266e-09	117.156
15	4658	550	8.1595657e-09	127.348
15	4515	538	1.1002054e-08	122.732
17	5328	643	8.5328593e-09	257.027
17	5198	632	1.7113676e-08	249.955
17	5549	661	1.571976e-08	268.528
17	5120	626	3.926036e-09	244.374
17	5523	658	3.6719631e-09	268.967
20	6761	824	5.7367444e-09	808.449
20	6605	811	1.703318e-08	785.491
20	7151	856	7.5590204e-09	869.278
20	6813	828	1.4787714e-08	817.514
20	6605	811	1.2080377e-08	782.764

n	virt. Feval	Feval	Fitness	Laufzeit
25	9904	1140	3.588387e-08	3792.992
25	10000	1145	3.8506773e-08	3834.524
25	10000	1145	4.057797e-08	3840.015
25	9680	1123	4.9438246e-08	3687.054
25	9862	1137	2.7643272e-08	3781.799
30	12890	1457	9.7175193e-08	12184.398
30	12920	1459	2.6307844e-08	12235.680
30	13535	1502	2.9501105e-08	12996.834
30	12740	1447	3.7628231e-08	11988.607
30	12980	1464	2.4477062e-08	12258.295

Tabelle 8.4: Ergebnisse aller Durchläufen der Optimierung von f_{Schwefel} durch LMM-CMAES (Laufzeit in Sekunden)

Ergebnisse f_{Schwefel} CMAES			
n	Feval	Fitness	Laufzeit
2	530	1.4786088e-16	2.095
2	470	7.7207043e-18	1.419
2	512	5.1380103e-17	1.502
2	512	3.1845805e-16	1.510
2	542	1.2360801e-16	1.588
5	1330	2.7923962e-16	2.658
5	1330	2.1395961e-15	2.561
5	1386	7.8558294e-16	2.623
5	1322	9.0454295e-16	2.529
5	1362	8.364959e-16	2.554
10	3052	3.8506856e-16	3.782
10	2952	1.163315e-15	3.619
10	3222	4.6595563e-15	3.759
10	3132	9.4576057e-16	3.687
10	2972	5.4767772e-16	3.473
15	5426	7.801836e-16	5.368
15	5510	2.9885728e-15	5.668
15	5726	3.4743604e-15	6.071
15	5474	2.4434661e-15	5.791
15	5498	1.9948597e-15	5.788
20	7994	2.6927029e-15	8.555
20	8066	2.9369412e-15	8.592
20	8150	3.3238582e-15	8.611
20	7982	6.3156021e-15	8.425
20	7946	2.8746118e-15	8.408
25	11442	3.1395109e-15	12.036
25	11494	3.8052259e-15	11.824
25	11364	5.6924249e-15	11.812
25	11884	1.1872273e-15	12.617
25	11494	7.9267887e-15	12.193
30	15472	2.8255637e-15	17.275
30	15094	5.2258725e-15	16.725
30	14898	5.1123851e-15	16.292
30	15486	3.2308187e-15	17.555
30	15276	2.2852154e-15	18.854
40	24272	2.2134197e-14	30.754
40	24587	1.4628621e-14	30.795
40	24302	2.1487737e-14	30.842

n	Feval	Fitness	Laufzeit
40	24887	9.4473327e-15	31.468
40	24392	1.9929512e-14	30.457

Tabelle 8.5: Ergebnisse aller Durchläufen der Optimierung von f_{Schwefel} durch CMAES (Laufzeit in Sekunden)

Ergebnisse $f_{\text{Rosenbrock}}$ LMM-CMAES				
n	virt. Feval	Feval	Fitness	Laufzeit
2	1196	309	4.3203441e-11	3.276
2	936	219	2.0456687e-11	2.629
2	910	227	7.5614291e-10	3.250
2	1134	282	2.9252425e-11	2.809
2	816	227	1.8120138e-10	2.282
5	3069	762	1.3919894e-08	7.236
5	2911	712	3.9308394	6.674
5	3767	971	2.7126968e-10	9.036
5	2594	588	3.9308394	5.759
5	3357	819	3.1846133e-10	7.988
10	10945	2979	3.9865791	130.107
10	10788	2708	8.5704879e-09	120.620
10	9468	2511	3.9865791	103.285
10	11574	3061	3.9865791	137.169
10	1126	305	1.3244469e-08	193.115
12	15081	4076	4.0392085e-09	342.550
12	15536	4460	3.3335736e-09	368.683
12	12672	3488	5.4826429e-09	262.332
12	14318	4146	5.9586832e-09	327.828
12	14612	4209	2.2151787e-09	327.371
15	22863	6269	2.4500949e-08	1319.431
15	21991	6027	3.9866238	1227.306
15	19021	5228	3.9866238	1026.509
15	26187	7227	6.1379791e-09	1602.214
15	22412	6232	1.3184854e-08	1277.033
17	28657	8012	9.1928755e-09	2956.996
17	30394	8285	1.088022e-07	3164.697
17	31780	9031	1.3793369e-08	3386.945
17	26343	7117	2.2278083e-08	2591.271
17	28628	7869	2.8267596e-08	2941.954
20	23950	6279	3.9866239	4889.184
20	52695	14823	6.9356586e-08	13536.557
20	41843	11791	7.7610859e-08	10015.420
20	43449	12191	1.790871e-08	10494.076
20	46820	13207	4.0175515e-08	11505.670
22	51875	15546	3.5182897e-08	20649.555
22	55333	16837	3.0589509e-08	22675.209
22	48041	14974	1.7854771e-08	18811.758

n	virt. Feval	Feval	Fitness	Laufzeit
22	40955	12532	2.7020809e-08	15291.350
22	49987	15071	2.6805404e-08	19858.989
25	79277	24036	1.0702497e-07	63827.514

Tabelle 8.6: Ergebnisse aller Durchläufen der Optimierung von $f_{\text{Rosenbrock}}$ durch LMM-CMAES (Laufzeit in Sekunden)

Ergebnisse $f_{\text{Rosenbrock}}$ CMAES			
n	Feval	Fitness	Laufzeit
2	553	2.0675835e-09	0.893
2	643	4.6316649e-11	0.896
2	559	1.2357765e-09	0.829
2	565	1.1314652e-10	0.836
2	1483	1.626146e-09	1.341
5	2185	1.7716859e-09	1.491
5	2793	1.4647318e-09	1.783
5	2225	3.031385e-09	1.515
5	3809	2.9496259e-10	2.279
5	2257	1.3208283e-09	1.535
10	8001	3.9865791	4.096
10	6181	3.9865791	3.240
10	7051	9.727708e-09	3.634
10	7681	2.6352441e-09	3.943
10	7771	4.3289825e-09	4.019
12	9186	2.4386997e-09	4.119
12	11683	4.0362936e-09	5.297
12	9131	4.2734848e-09	4.049
12	9659	5.5578781e-09	4.349
12	10088	1.3801858e-08	4.510
15	12637	5.3796522e-09	5.729
15	14821	4.0318139e-09	6.546
15	13153	1.064562e-08	6.019
15	13477	5.3630073e-09	6.197
15	14125	9.061809e-09	6.533
17	16609	1.0034297e-08	8.089
17	18973	1.7492444e-08	9.638
17	15877	1.040864e-08	7.607
17	16273	4.028092e-09	7.854
17	14185	3.9866239	6.655
20	21517	7.5696399e-09	12.010
20	21577	9.6363799e-09	12.036
20	18013	5.3486096e-09	9.416
20	22621	7.6382213e-09	12.510
25	32709	5.7632027e-09	21.481
25	32553	4.6325814e-09	21.302
25	35751	8.3932458e-09	24.142
25	33047	3.5227385e-09	21.689

n	Feval	Fitness	Laufzeit
25	32605	8.3148641e-09	21.290
30	42379	2.0193432e-08	33.115
30	46103	1.0349327e-08	37.415
30	46397	6.242397e-09	38.016
30	49659	1.5616446e-08	41.804
30	50597	1.0849096e-08	43.103

Tabelle 8.7: Ergebnisse aller Durchläufen der Optimierung von $f_{\text{Rosenbrock}}$ durch CMAES (Laufzeit in Sekunden)

Ergebnisse $f_{\text{Rastrigin}}$ LMM-CMAES				
n	virt. Feval	Feval	Fitness	Laufzeit
2	945	223	1.9899181	2.701
2	566	152	0.99495906	1.809
2	705	173	4.9747902	2.109
2	677	187	3.9798312	2.000
2	531	127	0.99495906	1.730
5	4523	1460	5.9697493	19.151
5	7544	2911	9.9495805	26.734
5	4263	1497	10.94454	11.807
5	3063	962	3.9798362	7.731
5	3702	1230	14.924351	9.654
10	27436	13023	14.924371	756.477
10	9934	4257	10.94454	141.359
10	6441	2629	12.934458	69.221
10	14343	6812	27.858788	260.173
10	9318	4282	15.91933	124.334
12	50554	31358	27.858823	3590.100
12	19376	11030	20.89412	663.394
12	51074	33514	29.848721	3643.050
12	32665	20588	26.863849	1588.513
12	21328	12336	21.889079	775.311
15	9810	3593	63.677127	458.588
15	73153	45340	31.838619	11349.867
15	48715	30437	39.798282	5631.248
15	27104	15075	25.868829	2080.154
15	9941	4009	37.808384	462.579
17	21183	10468	65.66704	2166.632
17	21585	10626	23.879002	2233.960
17	8742	2979	38.803348	633.572
17	15163	7288	30.843665	1320.815
17	19022	9122	33.828552	1863.481
20	16899	7033	37.808389	3304.875
20	15032	5906	42.783113	2927.643
20	12017	3958	36.81343	2220.193
20	14447	5982	35.818471	2730.812
20	16798	6969	39.798307	3278.087
20	16359	6296	19.899171	3212.532
25	17751	6576	43.778118	10325.143
25	17520	7092	64.672152	9731.935

n	virt. Feval	Feval	Fitness	Laufzeit
25	25489	11992	65.667111	15645.104
25	16594	6003	55.717596	9525.996
25	23958	11145	77.606467	14356.125

Tabelle 8.8: Ergebnisse aller Durchläufen der Optimierung von $f_{\text{Rastrigin}}$ durch LMM-CMAES (Laufzeit in Sekunden)

Ergebnisse $f_{\text{Rastrigin}}$ CMAES			
n	Feval	Fitness	Laufzeit
2	800	0.99495906	2.421
2	776	4.9747902	2.376
2	536	0.99495906	1.703
2	668	1.9899181	2.115
2	842	1.9899181	2.442
5	1834	5.9697493	3.229
5	1834	6.9647084	3.206
5	1602	9.9495805	3.023
5	1618	5.9697493	3.035
5	1826	12.934437	3.173
10	3982	21.888998	4.706
10	3082	7.9596674	4.034
10	3482	12.934458	4.317
10	3962	22.883952	4.660
10	3452	23.878952	4.284
15	6194	9.9495906	6.059
15	5162	37.808363	5.314
15	5690	29.848721	5.589
15	5606	12.934463	5.462
15	5402	19.899166	5.268
20	5774	32.833543	5.694
20	6986	28.853787	6.881
20	6158	24.873951	6.323
20	6110	35.81841	6.270
20	5462	29.848726	5.786
25	7126	55.717571	6.915
25	7945	34.823517	7.514
25	6957	56.712565	6.743
25	7594	52.732719	7.229
25	6957	53.727719	6.717
30	9704	47.757959	8.961
30	8598	41.788225	7.949
30	9886	58.702463	8.885
30	9746	55.717616	8.991
30	8850	57.707509	8.443
40	12452	99.495663	11.850
40	12497	90.541118	11.806
40	14087	48.752959	13.053

n	Feval	Fitness	Laufzeit
40	14702	54.722683	13.492
40	11657	54.722708	10.940

Tabelle 8.9: Ergebnisse aller Durchläufen der Optimierung von $f_{\text{Rastrigin}}$ durch CMAES (Laufzeit in Sekunden)

Kapitel 9

Zusammenfassung

9.1 Ergebnisse

Es ist gelungen mittels EANT2 einen Kantendetektor zu generieren, der mit kleinen, synthetischen Daten trainiert wurde und auf einer Reihe von Bildern gute Ergebnisse erzielt. Dabei liefert er auch bei natürlichen Bildern ansprechende Resultate und erzielt bessere Ergebnisse als der Sobel-Kantendetektor.

Es wurde beobachtet, dass man durch Auslassen der Reoptimierung der Elterngeneration einen deutlichen Geschwindigkeitsgewinn erreichen kann, ohne viel an der Qualität der Ergebnisse zu verlieren. Es stehen hier durchschnittlich 1,6% schlechtere Ergebnisse rund 33% Beschleunigung gegenüber.

Um zukünftig die Analyse der EANT2-Durchläufe zu vereinfachen, wurde ein Modul in EANT2 integriert, welches die Wege der einzelnen Individuen (Mutationen und Optimierungen) aufzeichnet und leicht auslesbar ausgibt.

Für eine der wichtigsten Komponenten des Algorithmus, die Parameteroptimierung, wurden 3 verschiedene Maßnahmen zur Effizienzsteigerung untersucht.

Mit PSO wurde eine Alternative zu CMAES betrachtet. Dabei konnte PSO ähnlich gute Ergebnisse erzielen wie CMAES. PSO war allerdings häufig geringfügig schlechter als CMAES, weshalb PSO nicht als Ersatz hierfür zu sehen ist. Da PSO aber durch seine relative Einfachheit ein schneller Optimierungsalgorithmus ist, der in einem Fall sogar bessere Ergebnisse erzielen konnte als CMAES, kann er als gute Ergänzung angesehen werden.

Die Erhöhung der CMAES-Population erbrachte keine eindeutigen Ergebnisse. Je nach getesteten Netzwerken konnte entweder eine leichte Verschlechterung (Generation 1) oder eine (deutlichere) Verbesserung (Generation 15) beobachtet werden. Eine Erhöhung der Population auf den fünffachen Wert

erbrachte dabei nur eine geringe Verbesserung, so dass (wenn überhaupt) die Population mindestens um den Faktor zehn erhöht werden sollte. Dabei konnte besonders bei den größeren Netzwerken eine Verbesserung beobachtet werden.

Die Verwendung lokaler Metamodelle kann zwar die Anzahl der Funktionsevaluationen reduzieren, bewirkt allerdings keine Beschleunigung, da das Aufstellen der Modelle, bei den hier vorherrschenden Dimensionen, an sich schon zu Zeitaufwendig ist. Außerdem sind die erreichten Fitnesswerte durch Modellbildung teilweise drastisch schlechter als bei dem originalen CMAES.

Durch die wechselhaften Umweltbedingungen war es bis jetzt noch nicht möglich, Netzwerke zu generieren, die einen simulierten Helikopter an Ort und Stelle schweben lassen.

9.2 Ausblick

Vor allem bei der Verwendung von PSO in EANT2 bestehen noch viele Möglichkeiten weiter zu forschen. So könnte man noch untersuchen, inwieweit sich eine Erhöhung der Schwarmgröße auswirkt. Außerdem könnte man eine Version von EANT2 entwerfen, bei der nicht jede Struktur mehrmals mit CMAES sondern einmal mit CMAES und einmal mit PSO optimiert wird.

Auch müsste überprüft werden, ob der Vorsprung, der durch PSO optimierten Netze für die Kantendetektion nur ein einmaliges Ergebnis war oder die Regel für dieses Problem.

Es könnte auch noch untersucht werden, ob der Verlust an Fitness, der durch das Weglassen der Reoptimierung erzeugt wird, durch zusätzliche Optimierungsläufe ausgeglichen und trotzdem einen Geschwindigkeitsgewinn erzielt werden kann. Denkbar wären eine Optimierung der besten Struktur ganz am Ende des gesamten Durchlaufs oder evtl. eine zusätzliche Optimierung aller Strukturen beispielsweise jede fünfte Generationen.

Da sich die Erhöhung der CMAES-Population besonders bei großen Netzwerken positiv ausgewirkt hat, könnte untersucht werden, ob eine Wachstumsrate der Population sinnvoll wäre, die stärker wächst, als die Momentane.

Anhang A

Abkürzungsverzeichnis

Abkürzung	Bedeutung
3DOF	3 Degree Of Freedom
CMAES	Covariance Matrix Adaptation Evolution Strategy
EA	Evolutionärer Algorithmus
EANT (2)	Evolutionary Acquisition of Neural Topologies (2)
LMM-CMAES	Local Meta Model Covariance Matrix Adaptation Evolution Strategy
PSO	Particle Swarm optimisation
RL	Reinforcement Learning
VS	Visual-Servoing

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[AMS97] C.G. Atkeson, A.W. Moore, and S. Schaal. Locally weighted learning. *Artificial intelligence review*, 11(1):11–73, 1997.

[Bis95] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Great Clarendon Street, Oxford OX2 6DP, 1995.

[Cle79] W.S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, pages 829–836, 1979.

[Cle06] M. Clerc. Stagnation analysis in particle swarm optimisation or what happens when nothing happens. *alea*, 5(5.0):0, 2006.

[Dar59] C. Darwin. *The Origin of Species*. John Murray, London, UK, 1859.

[DM06] Dake and Mysid. A simplified view of an artifical neural network. `http://de.wikipedia.org/w/index.php?title=Datei:Neural_network.svg`, 2006.

[ES03] Ágoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, Berlin, Germany, 2003.

[FOW65] L.J. Fogel, A.J. Owens, and M.J. Walsh. Artificial intelligence through a simulation of evolution. In A. Callahan, M. Maxfield, and L.J.Fogel, editors, *Biophysics and Cybernetic Systens*, pages 131–156, 1965.

[FS04] A. Faller and M. Schünke. Der Körper des Menschen. 2004.

[Gru07] Sven Gruenewald. Entwurf eines Kantendetektors mit EANT2. Studienarbeit, Christian-Albrechts-Universität zu Kiel, 2007.

[HH97] M.E. Harmon and S.S. Harmon. Reinforcement learning: a tutorial, 1997.

[HK04] N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. 2004.

[HO01] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[Hog06] S.G. Hoggar. *Mathematics of Digital Images*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 2006.

[Hol73] J.H. Holland. Genetic algorithms and the optimalallocation of trials. *SIAM J. of Computing*, pages 88–105, 1973.

[IN06] Interiot and Noddy93. Typische Struktur eines Neurons mit in der Peripherie befindlichem Axon. `http://commons.wikimedia.org/wiki/File:Neuron_%28deutsch%29-1.svg`, 2006.

[Kas06] Yohannes Kassahun. *Towards a Unified Approach to Learning and Adaptation*. PhD thesis, Cognitive Systems Group, Institute of Computer Science, Christian-Albrechts-University of Kiel, Germany, February 2006.

[KHK06] Stefan Kern, Nikolaus Hansen, and Petros Koumoutsakos. Local meta-models for optimization using evolution strategies. In *Parallel Problem Solving from Nature - PPSN IX*. 2006.

[KS05] Yohannes Kassahun and Gerald Sommer. Efficient reinforcement learning through evolutionary acquisition of neural topologies. In *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN 2005)*, pages 259–266, Bruges, Belgium, April 2005.

[Lei06] J.G. Leishman. *Principles of helicopter aerodynamics*. Cambridge Univ Pr, 2006.

[LR08] R. Lüllmann-Rauch. *Histologie*. De Boeck, 2008.

[Pet09] Dennis Peters. Zusammenspiel von Modellen und Reglern bei Roboterregelungen. Diplomarbeit, Christian-Albrechts-Universität zu Kiel, 2009.

[Rec73] I. Rechenberg. *Evolutionsstrategie: Optimierung TechnischerSysteme nach Prinzipien der Biologischen Evolution.* Fromman-Holzboog Verlag, Stuttgart, Germany, 1973.

[SBS09] Nils T Siebel, Jonas Bötel, and Gerald Sommer. Efficient neural network pruning during neuro-evolution. In *Proceedings of 2009 International Joint Conference on Neural Networks (IJCNN 2009), Atlanta, USA*, pages 2920–2927, June 2009.

[SF68] I. Sobel and G. Feldman. A 3x3 isotropic gradient operator for image processing. *Presentation for Stanford Artificial Project*, 1968.

[SGS08] N.T. Siebel, S. Grünewald, and G. Sommer. Creating edge detectors by evolutionary reinforcement learning. In *IEEE Congress on Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence)*, pages 3553–3560, 2008.

[Sie99] Nils T Siebel. Bildbasierte Roboterregelung in sechs Freiheitsgraden unter Verwendung einer Trust-Region-Methode. Diplomarbeit, Zentrum für Technomathematik und Institut für Automatisierungstechnik, Universität Bremen, Bremen, August 1999.

[SJS09] Nils T. Siebel, Andreas Jordt, and Gerald Sommer. Compiling neural networks for fast neuro-evolution. In *Proceedings of the 2nd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems*, pages 23–29, October 2009.

[SK06] Nils T Siebel and Yohannes Kassahun. Learning neural networks for visual servoing using evolutionary methods. In *Proceedings of the 6th International Conference on Hybrid Intelligent Systems (HIS'06), Auckland, New Zealand*, page 6 (4 pages), December 2006.

[SKS07] Nils T Siebel, Jochen Krause, and Gerald Sommer. Efficient learning of neural networks with evolutionary algorithms. In *Proceedings of the 29th Annual Symposium of the German Association for Pattern Recognition (DAGM2007), Heidelberg, Germany*, pages 466–475, Berlin, September 2007. Springer Verlag.

[Som06] Gerald Sommer. Neuroinformatik. Vorlesungsskript, 2006.

[SS07] N.T. Siebel and G. Sommer. Evolutionary reinforcement learning of artificial neural networks. *International Journal of Hybrid Intelligent Systems*, 4(3):171–183, 2007.

[SS08] N.T. Siebel and G. Sommer. Learning defect classifiers for visual inspection images by neuro-evolution using weakly labelled training data,". In *IEEE Congress on Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence)*, pages 3925–3931, 2008.

[TW09] Brian Tanner and Adam White. RL-Glue : Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10:2133–2136, September 2009.

[WTW09] Shimon Whiteson, Brian Tanner, and Adam White. 2009 reinforcement learning competition. `http://rl-competition.org`, 2009.

[ZSL00] C. Zhang, H. Shao, and Y. Li. Particle swarm optimisation for evolving artificial neural network. In *2000 IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, 2000.