Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Neuroinformatik und Kognitive Robotik

Evolutionäre Problemzerlegung mit Hilfe von neuronalen Netzen

Masterarbeit zur Erlangung des akademischen Grades Master of Science

Robert Kaltenhäuser

Betreuer: Dipl.-Inf. Erik Schaffernicht, FG Neuroinformatik und Kognitive Robotik
Dipl.-Inf. Florian Steege, Powitec GmbH
Verantwortlicher Hochschullehrer:
Prof. Dr. H.-M. Groß, FG Neuroinformatik und Kognitive Robotik

Die Masterarbeit wurde am 01.08.2012 bei der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau eingereicht.

Erklärung: „Hiermit versichere ich, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Alle von mir aus anderen Veröffentlichungen übernommenen Passagen sind als solche gekennzeichnet.“

Ilmenau, 01.08.2012 .....

Robert Kaltenhäuser

Inhaltsverzeichnis

1 Einleitung **1**

2 Problemstellung **3**

2.1 Grundbegriffe in einem Regelsystem 3

2.2 Beispielanwendung 4

2.3 Arten der Problemzerlegung 6

2.3.1 Zeitliche Problemdekomposition 8

2.3.2 Nichthierarchisch funktionale Problemdekomposition 11

2.3.3 Hierarchisch funktionale Problemdekomposition 13

2.4 Vorteil einer Problemzerlegung 16

2.5 Zusammenfassung Problemstellung 18

3 Grundlagen **21**

3.1 Evolution 21

3.2 Evolutionäre Algorithmen 24

3.2.1 Selektion 28

3.2.2 Mutation 29

3.2.3 Rekombination 30

3.3 Levenberg-Marquardt Lernverfahren 31

3.4 Mutual Information 36

3.4.1 Abschätzung der Mutual Information aus Daten 37

3.5 Zusammenfassung der Grundlagen 42

4 Methoden zur Problemzerlegung **45**

4.1 Fitnessberechnung 45

4.1.1 Fitnessberechnung über die Zugehörigkeit der Problemgrößen . . 46

4.1.2 Fitnessberechnung über die Abhängigkeiten 48

4.1.3 Fitnessberechnung über die Fehlerrate 50

4.2 Network Pruning 53

4.2.1 Bayes'sche Regularisierung 55

4.2.2 Optimal Brain Surgeon 70

4.3 Verfahren zur Problemzerlegung 73

4.3.1 Problemzerlegung durch Input Pruning 74

4.3.2 Problemzerlegung durch Partitionierung Variante 1 81

4.3.3 Problemzerlegung durch Partitionierung Variante 2 87

4.4 Erweiterungen der Verfahren zur Problemzerlegung 93

4.4.1 Time Delay Estimation 93

4.4.2 Time Delay Estimation durch Mutual Information 95

4.4.3 Berechnung der Anzahl der Teilprobleme 98

4.4.4 Nicht exklusive Problemzerlegung 104

4.5 Zusammenfassung Methoden zur Problemzerlegung 109

5 Experimentelle Untersuchungen **115**

5.1 Verwendete Testdaten 115

5.1.1 Simulierte Testdaten 115

5.1.2 Realweltdaten 126

5.2 Test der Algorithmen 127

5.2.1 Test der Standardverfahren 131

5.2.2 Test der Time Delay Estimation 136

5.2.3 Test der Ermittlung der Anzahl der Teilprobleme 141

5.2.4 Test der nicht exklusiven Problemzerlegung 145

5.3 Vergleich mit anderen Verfahren 145

5.4 Zusammenfassung experimentelle Untersuchungen 148

6 Zusammenfassung und Ausblick **151**

A Totzeiten und Tiefpässe der Testdatensätze **155**

A.1 Testdatensatz P1 156
A.2 Testdatensatz P1g 157
A.3 Testdatensatz P2 158
A.4 Testdatensatz P2g 159
A.5 Testdatensatz P3 160
A.6 Testdatensatz P3g 161
A.7 Testdatensatz P4gS 162
A.8 Testdatensatz P9g 163

B Ergebnisse der Tests der Verfahren **165**

B.1 Test der Standardverfahren 166
B.1.1 Problemzerlegung durch Input Pruning 166
B.1.2 Problemzerlegung durch Partitionierung Variante 1 168
B.1.3 Problemzerlegung durch Partitionierung Variante 2 171
B.2 Test der Time Delay Estimation 174
B.2.1 Problemzerlegung durch Input Pruning 174
B.2.2 Problemzerlegung durch Partitionierung Variante 1 176
B.2.3 Problemzerlegung durch Partitionierung Variante 2 179
B.3 Test der Ermittlung der Anzahl der Teilprobleme 183
B.3.1 Problemzerlegung durch Partitionierung Variante 1 183
B.3.2 Problemzerlegung durch Partitionierung Variante 2 186

C Ergebnisse der Time Delay Estimation **189**

C.1 Testdatensatz P1 190
C.1.1 Totzeiten für Datensatz P1 190
C.1.2 Tiefpässe für Datensatz P1 190
C.2 Testdatensatz P1g 191
C.2.1 Totzeiten für Datensatz P1g 191
C.2.2 Tiefpässe für Datensatz P1g 191

C.3 Testdatensatz P4gS 192
C.3.1 Totzeiten für Datensatz P4gS 192
C.3.2 Tiefpässe für Datensatz P4gS 193

D Liste verwendeter Formelzeichen **195**

Algorithmenverzeichnis **201**

Abbildungsverzeichnis **207**

Literaturverzeichnis **211**

Kapitel 1

Einleitung

In heutzutage immer komplexer werdenden Regelsystemen, welche immer mehr Möglichkeiten zur Beeinflussung der Prozesse besitzen und oftmals auch miteinander verkoppelt sind, ist es für das menschliche Personal, aber auch für eventuell vorhandene technische Lernsysteme immer schwieriger den Überblick zu behalten. Ein denkbares Szenario für diesen Fall wäre zum Beispiel ein Kohlekraftwerk mit seiner hohen Anzahl an Reglern, Messwerten, Kameras und einem Verbrennungsprozess, für welchen bis heute noch kein zufriedenstellendes physikalisches Modell existiert. Aufgrund der hohen Komplexität und des wahrscheinlich daraus resultierenden mangelnden Verständnisses für den Prozess kann eine suboptimale Regelung resultieren, welche viel Potenzial des zu regelnden Systems brachliegen lässt.

Eine mögliche Herangehensweise an dieses Problem ist die Zerlegung des Systems in mehrere kleinere, möglichst entkoppelte Teilprobleme, welche überschaubarer und damit besser optimierbar sind. Eine manuelle Zerlegung durch einen menschlichen Experten ist jedoch aufgrund der hohen Komplexität schwierig und benötigt detailliertes Wissen über den Prozess, welches nicht immer zur Verfügung steht.

Diese Masterarbeit ist Teil des APD-Projektes des Fachgebietes *Neuroinformatik und kognitive Robotik* an der *TU Ilmenau*. Das Ziel dieses APD-Projektes ist es, mit Hilfe gesammelter Prozessdaten, ein solches komplexes Regelsystem automatisch in kleinere

Teilprobleme zu zerlegen. Zu diesem Zweck werden mehrere Lösungsansätze für diese Aufgabe untersucht und verglichen.

In dieser Masterarbeit soll einer dieser Lösungsansätze untersucht werden, welcher sich der Methoden aus dem Bereich der evolutionären Algorithmen bedient. Mit diesem sollen im Verlauf einer künstlichen Evolution mehrere neuronale Netze entsprechend der Struktur des Gesamtsystems zerlegt werden. Aus der Struktur der resultierenden neuronalen Netze können so Schlüsse auf die Struktur und die Teilprobleme des Regelsystems geschlossen werden. In der vorliegenden Arbeit sollen mehrere Möglichkeiten dies durchzuführen vorgestellt werden. Die Verfahren werden anhand von Simulationsdaten und Realweltdaten aus einem Kohlekraftwerk getestet und sowohl untereinander als auch mit Verfahren aus anderen Lösungsansätzen des APD-Projektes verglichen.

Dazu wird in Kapitel 2 zunächst die Themenstellung spezifiziert und die Arten der Problemzerlegung aufgezeigt, sowie anhand eines Beispiels verdeutlicht. Anschließend werden in Kapitel 3 einige Grundlagen erläutert, welche im Laufe der Arbeit benötigt werden. Die Algorithmen zur Problemzerlegung selbst werden schließlich im Kapitel 4 vorgestellt und im darauf folgenden Kapitel 5 getestet und sowohl untereinander, als auch mit anderen Algorithmen aus dem APD-Projekt verglichen. Schließlich erfolgt im Kapitel 6 eine Zusammenfassung der Ergebnisse der Arbeit.

Kapitel 2

Problemstellung

In diesem Abschnitt werden zunächst einige Grundbegriffe im Zusammenhang mit Regelsystemen erläutert. Anschließend wird ein Beispiel vorgestellt, welches die Problemstellung verdeutlichen und die Idee, welche hinter der Problemzerlegung und dieser Arbeit steckt, illustrieren soll und anhand dessen die Sachverhalte im Laufe des Kapitels und der Arbeit näher erläutert werden. Außerdem soll gezeigt werden, auf welche verschiedenen Arten ein Problem zerlegt werden kann. Eine Übersicht über diese Methoden der Problemdekomposition wird erstellt und weiterhin wird das Thema für diese Arbeit umrissen und eingegrenzt. Gegen Ende werden die Vorteile einer Problemdekomposition vorgestellt und deren Nutzen an einem Beispiel demonstriert.

2.1 Grundbegriffe in einem Regelsystem

Zum weiteren Verständnis der Sachverhalte sollen hier einige im Folgenden verwendete Begriffe im Zusammenhang mit Regelsystemen geklärt werden. So besteht ein Regelsystem aus mehreren sogenannten **Problemgrößen**, welche einer der drei folgenden Kategorien angehören:

1. **Kenngrößen**
   Kenngrößen bezeichnen Problemgrößen, welche durch keine anderen Größen des Systems beeinflusst werden können. Sie sind mehr oder weniger fest vorgegeben

und können somit als reine Eingangsgrößen für das Regelsystem angesehen werden. Allerdings besitzen sie selbst Einfluss auf andere Größen. Mögliche Beispiele sind die Umgebungstemperatur oder der Brennwert der verwendeten Kohle.

2. **Messgrößen**
   Messgrößen können, im Gegensatz zu Kenngrößen, von anderen Prozessgrößen beeinflusst werden. Sie sind, wie der Name schon sagt, messbar und sollen, entweder direkt oder zumindest indirekt, so geregelt werden, dass ein gewünschtes Verhalten des Systems realisiert wird. Beispiele für Messgrößen sind die Kesseltemperatur oder der Kohlenmonoxidausstoß in einem Kraftwerk.

3. **Regelgrößen**
   Regelgrößen sind Größen, durch welche Einfluss auf bestimmte Messgrößen genommen und diese dadurch geregelt werden sollen. Ein dazwischengeschalteter Regler „beobachtet“ die Messgröße. Bei einem Verhalten, welches vom Gewünschten abweicht, nimmt dieser Einfluss auf die Regelgröße, welche wiederum die zu regelnde Messgröße beeinflusst. Eine denkbare Regelgröße ist die Brennstoffzufuhr eines Brenners, durch welche die Temperatur in einem Kessel geregelt werden soll.

2.2 Beispielanwendung

Eine Beispielanwendung, bei welcher eine Problemdekomposition erfolgreich durchgeführt werden könnte, ist in einem Kohlekraftwerk. Die grundlegende Funktionsweise eines solchen Kohlekraftwerks ist vereinfacht in Abbildung 2.1 verdeutlicht. Zuerst wird in einer Kohlemühle Kohle zu Kohlestaub zermahlen, welcher dann, zusammen mit der zugeführten Luft, im Kessel verbrannt wird. Die erhitzten Abgase steigen nach oben, werden dort von Schadstoffen und Restasche gereinigt und entweichen schließlich aus dem Schornstein.

In einem solchen Kohlekraftwerk wäre es möglich, beispielsweise den Kohlenmonoxidgehalt in den Abgasen (CO) und die Menge der Restkohle in der Asche (unburned

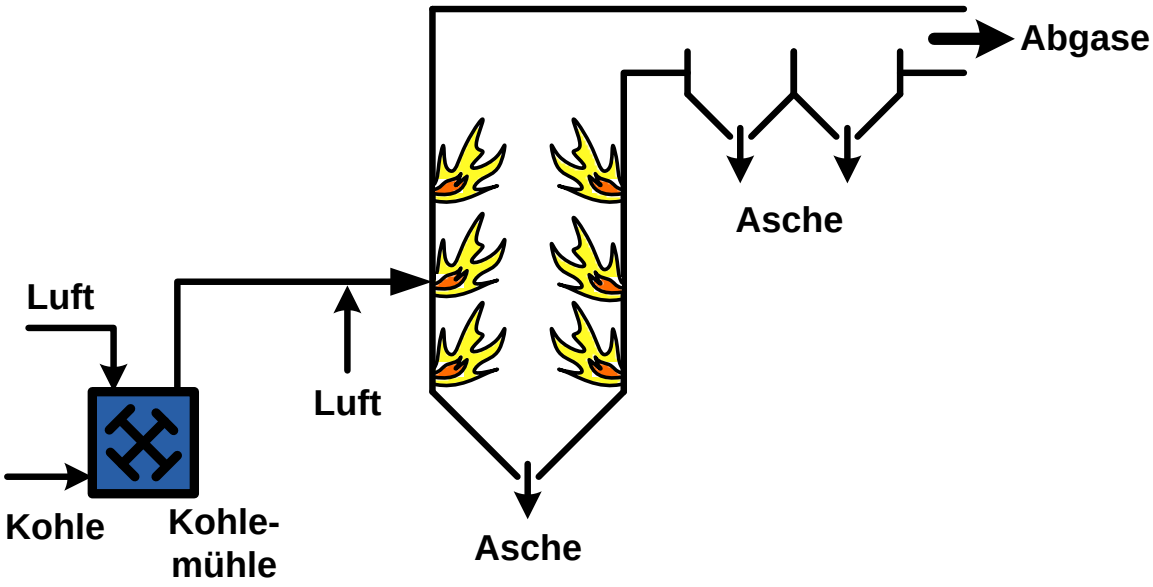

Abbildung 2.1: Darstellung des grundlegenden Ablaufs in einem Kohlekraftwerk.

Carbon in Ash, CIA) zu optimieren. Ersteres entsteht vor allem, wenn beim Verbrennen der Kohle ein Mangel an Sauerstoff herrscht. Die wichtigsten Einflussgrößen sind folglich die Luftzufuhr und die Kohlemenge bzw. deren Verhältnis zueinander. Die Restkohle wiederum entsteht, wenn in der Kohlemühle die Kohle nicht fein genug gemahlen wird. Die zu großen Kohlekörner verbrennen nicht vollständig und verlassen den Kessel unverbrannt. Abbildung 2.2 stellt diese Größen und deren Abhängigkeiten grafisch dar.

Auf die Kohlemenge besitzt keine der übrigen Prozessgrößen einen Einfluss. Auch wird sie nicht zur Regelung anderer Größen verwendet, da sie sonst, abhängig vom Verhalten dieser Größe, verändert werden müsste. Entsprechend den Bezeichnungen aus Kapitel 2.1 ist sie demnach eine Kenngröße (blau). Die Luftzufuhr wiederum wird, abhängig vom Kohlenmonoxidausstoß geregelt. Ist der Anteil von Kohlenmonoxid in den Abgasen beispielsweise zu hoch, wird durch einen Regler die Luftzufuhr erhöht. Sie ist damit eine Regelgröße (grün). Das Gleiche gilt für den Grad der Feinkörnigkeit der Kohlemühle, welcher sich bei Erhöhung des Restkohlegehalts in der Asche

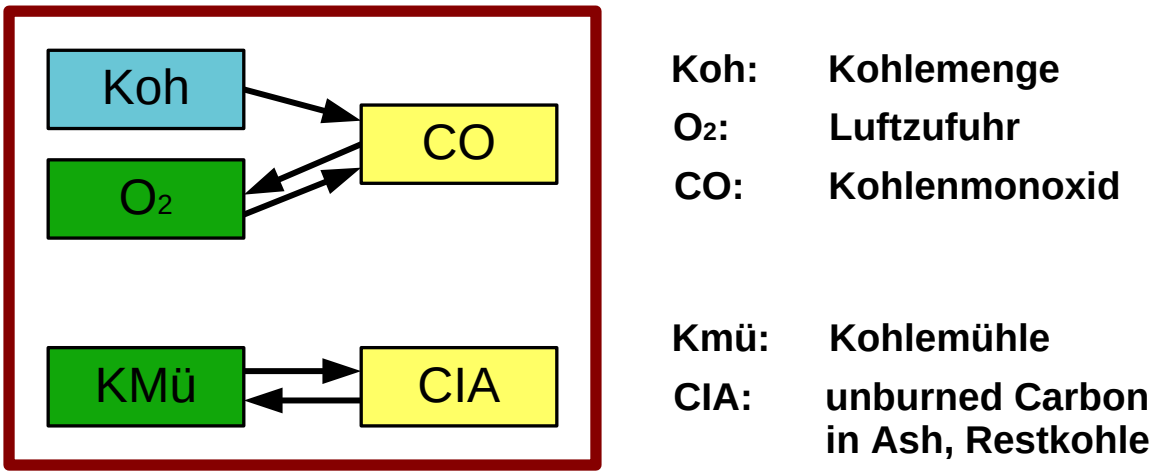

Abbildung 2.2: Darstellung der Abhängigkeiten im Beispielszenario Kohlekraftwerk. Die Messgrößen sind gelb, Kenngrößen blau und Regelgrößen grün gekennzeichnet. Die Pfeile zeigen die Abhängigkeiten.

erhöht. Der Kohlenmonoxidausstoß und die Restkohle in der Asche schließlich sind Messgrößen (gelb), da sie selbst nichts regeln, aber von anderen Größen beeinflusst werden.

2.3 Arten der Problemzerlegung

Das Beispiel aus Kapitel 2.2 lässt sich mit Hilfe gemessener Daten aus dem Kraftwerk und eines neuronalen Netzes simulieren. Dafür nimmt das neuronale Netzwerk die vergangenen Messwerte aller Problemgrößen zu den Zeitpunkten t , $t-1$, $t-2$, ... , $t-x$ als Inputs und versucht daraus alle von diesen Größen abhängigen Mess- und Regelgrößen zum Zeitpunkt $t+1$ vorauszusagen. Dies ist in Abbildung 2.3 bildlich dargestellt.

Mit dieser Abschätzung künftiger Messwerte lassen sich unter anderem Regler für das System entwerfen. Denn mit der Hilfe des neuronalen Netzes ist es ebenfalls möglich, die Reaktionen bei Änderungen der Regelgrößen vorauszusagen. Das einfachste Beispiel hierfür ist eine sogenannte modellprädiktive Regelung. Ausgehend vom aktuellen

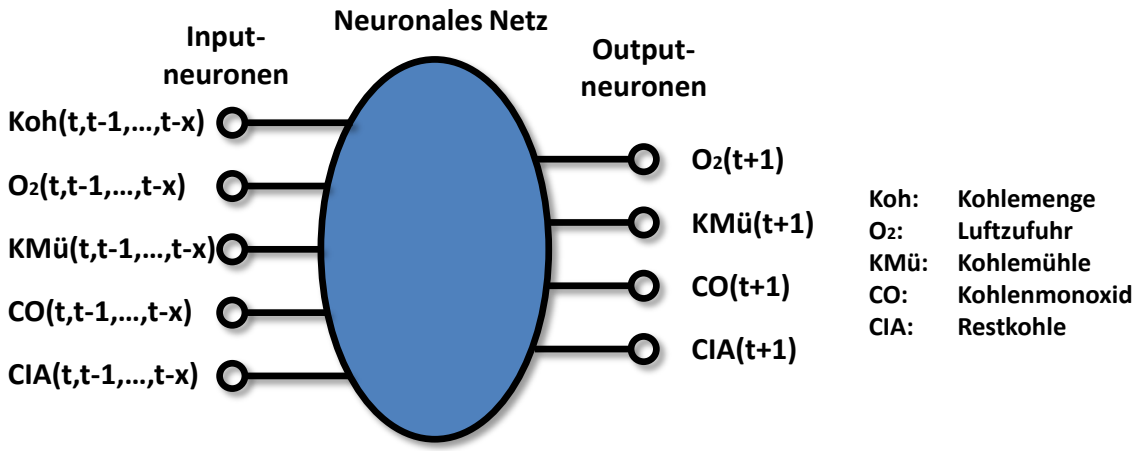

Abbildung 2.3: Schematische Darstellung der Approximation des Beispielregelsystems durch ein neuronales Netzwerk.

Zustand des Systems prüft diese alle aktuell möglichen Aktionen zum Regeln des Systems. Mit Hilfe des neuronalen Netzes kann er die Auswirkungen dieser Aktionen abschätzen und regelt letztendlich mit der Aktion, welche im wünschenswertesten Folgezustand resultiert.

Es ist hierbei offensichtlich, dass je genauer die Güte der Approximation ist, desto besser die daraus abgeleitete Regelung wird. Ein einziges großes monolithisches neuronales Netzwerk ist dafür jedoch oftmals nicht geeignet. Das vorliegende Beispiel ist relativ überschaubar und die Abhängigkeiten sind alle bekannt, in Realweltanwendungen besitzen aber weit mehr Größen Einfluss auf den Prozess, mehrere Größen sollen gleichzeitig optimiert werden und weit mehr Regler sind im Einsatz. Auch die Abhängigkeiten dieser Problemgrößen untereinander sind oftmals unklar. Aufgrund dieser Tatsachen, der hohen Anzahl an möglichen Problemgrößen und der daraus resultierenden Unzahl an Systemzuständen ist ein einzelnes neuronales Netz oftmals überfordert und der Approximationsfehler steigt. In diesem Fall bietet es sich an, das Problem in mehrere kleine, überschaubarere Teilprobleme zu zerlegen. Auf jedes dieser Teilprobleme kann dann jeweils ein einzelnes neuronales Netz trainiert werden, welches auf diese Weise mit einer deutlich verringerten Komplexität konfrontiert ist. Der Approximationsfehler und dadurch die Güte der resultierenden Regler verbessert

sich dadurch merklich.

Es gibt jedoch mehrere mögliche Arten eine Problemdekomposition vorzunehmen: die zeitliche Dekomposition und die funktionale Dekomposition, wobei sich letztere noch einmal in die hierarchisch funktionale und die nichthierarchisch funktionale Dekomposition unterteilt. Abbildung 2.4 fasst diese in einer Übersicht zusammen. Die darin gezeigten Arten der Problemdekomposition sollen im Folgenden ausführlich vorgestellt und, falls möglich, am Beispielproblem verdeutlicht werden.

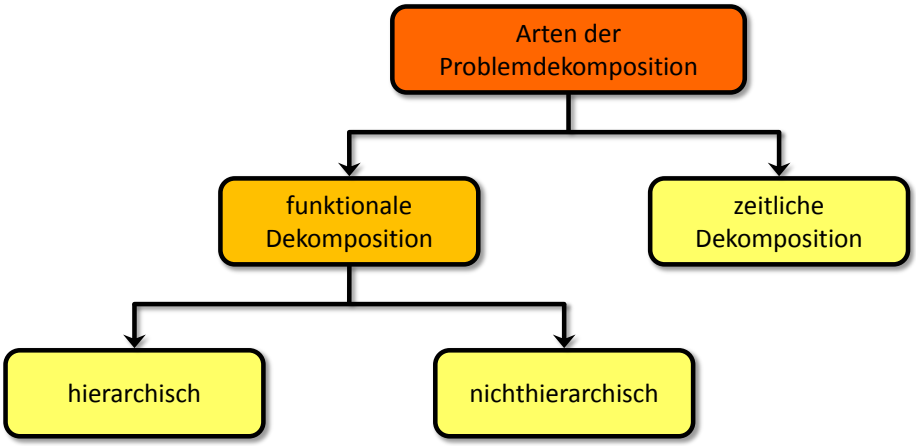

Abbildung 2.4: Übersicht über verschiedene Arten der Problemdekomposition.

2.3.1 Zeitliche Problemdekomposition

Die Grundidee der zeitlichen Problemdekomposition ist es, das Problem zeitlich bzw. nach unterschiedlichen Situationen und Systemzuständen zu zerlegen. So wäre es beispielsweise denkbar, das Beispielproblem in zwei zeitlich getrennte Teilprobleme zu zerlegen, wobei jedes davon für den Betrieb des Kraftwerkes mit einer anderen Kohlesorte steht. Wie in Abbildung 2.5 links gezeigt, würde das System mit all seinen Problemgrößen immer noch als ein einziges Problem angesehen werden, jedoch würde der Inputraum zerlegt werden und die Approximation des Systems, abhängig von der aktuellen Inputsituation, von einem anderen neuronalen Netz übernommen werden. In Abbildung 2.5 rechts wurde dies grafisch dargestellt. Gezeigt wird die Approximation

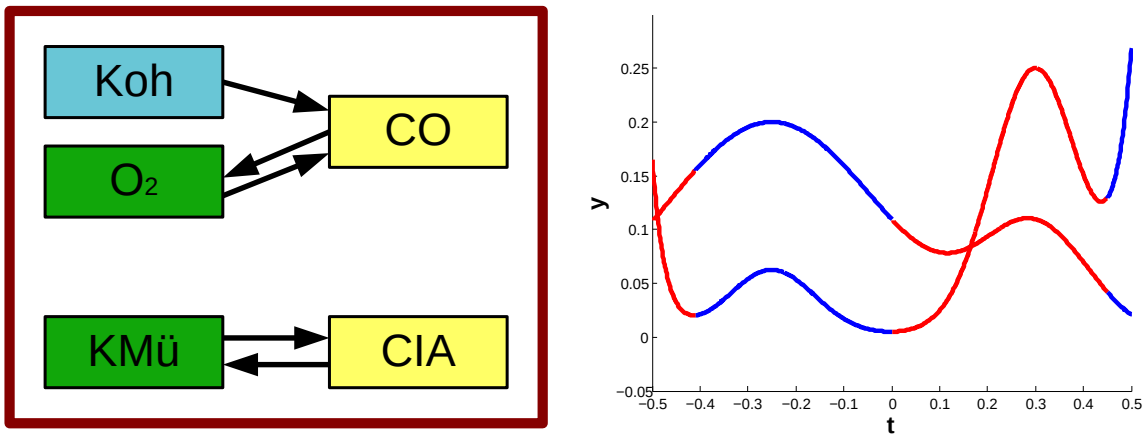

Abbildung 2.5: Darstellung der zeitlichen Problemdekomposition am Beispielproblem. Das linke Bild zeigt die Abhängigkeiten der Problemgrößen und ihre Partitionierung. Sie würden für die zeitliche Dekomposition immer noch als ein einziges Problem angesehen werden. Das Bild rechts zeigt den zeitlichen Verlauf der Approximation von zwei der Problemgrößen. Je nach Inputsituation (z.B. je nach Kohlesorte) würden sie von einem von zwei neuronalen Netzen (blau bzw. rot dargestellt) approximiert werden.

zweier beliebiger Problemgrößen. Je nach Inputsituation, d.h. am Beispiel je nach Kohlesorte, werden diese Größen jedoch von einem von zwei verschiedenen neuronalen Netzen approximiert, welche rot bzw. blau dargestellt sind. Das rote Netz würde das System für Kohlesorte 1 approximieren, das blaue Netz dagegen für Kohlesorte 2.

Die am weitesten verbreitete Methode eine zeitliche Problemdekomposition durchzuführen ist, mit Hilfe der sogenannten Mixture of Experts - Architektur, welche erstmals von [Jacobs et al., 1991] vorgeschlagen und in [Nguyen et al., 2006] auch evolutionär, im Sinne dieser Arbeit, verwendet wurde. Wie bereits beschrieben werden hier mehrere verschiedene neuronale Netze verwendet, welche alle Problemgrößen, allerdings nur zu bestimmten Zeitpunkten bzw. Inputsituationen, approximieren. Diese werden Experten genannt. Der Gesamtoutput ergibt sich jeweils aus den gewichteten Mittelwerten

der Outputs der einzelnen Experten:

$$y_i(x) = \sum_{j=1}^{M} g^j(x) \cdot y_i^j(x) \qquad \text{mit} \qquad \sum_{j=1}^{M} g^j(x) = 1 \tag{2.1}$$

wobei i der Index des Outputs ist und j der Index des Experten. x stellt den Vektor der Inputs dar, g^j ist das Gewicht für Experte j, welches zwischen 0 und 1 liegt, und M gibt die Anzahl der Experten an.

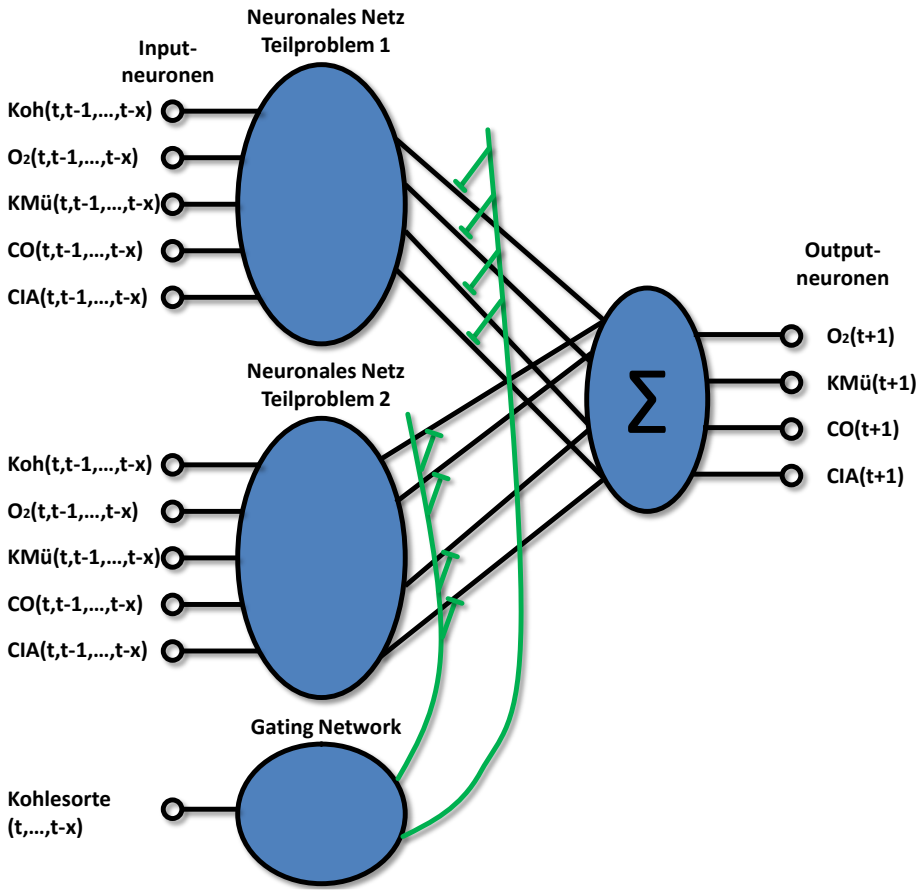

Abbildung 2.6: Mixture of Experts - Architektur für das Beispielproblem.

Zusätzlich gibt es ein sogenanntes Gating Network, welches, abhängig von der Inputsituation, die Gewichte g für die Experten bestimmt. Die Gewichte der Experten, welche aktuell nicht zuständig sind, werden dafür auf 0 oder nahe 0 gesetzt. Trainieren kann man dieses Modell mit dem Expectation Maximization (EM) Algorithmus [Azam,

2000].

Abbildung 2.6 zeigt die Mixture of Experts - Architektur für das Beispielproblem. Dargestellt sind die beiden neuronalen Netze für die zwei Teilprobleme und das Gating Network, welches, hier nur abhängig von der Kohlesorte, die Gewichte der beiden Netze für die Mittelwertbildung steuert. Wird aktuell Kohlesorte 1 verwendet, wird das Gewicht von Experte 1 auf 1 gesetzt und das von Experte 2 auf 0. Bei Kohlesorte 2 entsprechend umgekehrt.

2.3.2 Nichthierarchisch funktionale Problemdekomposition

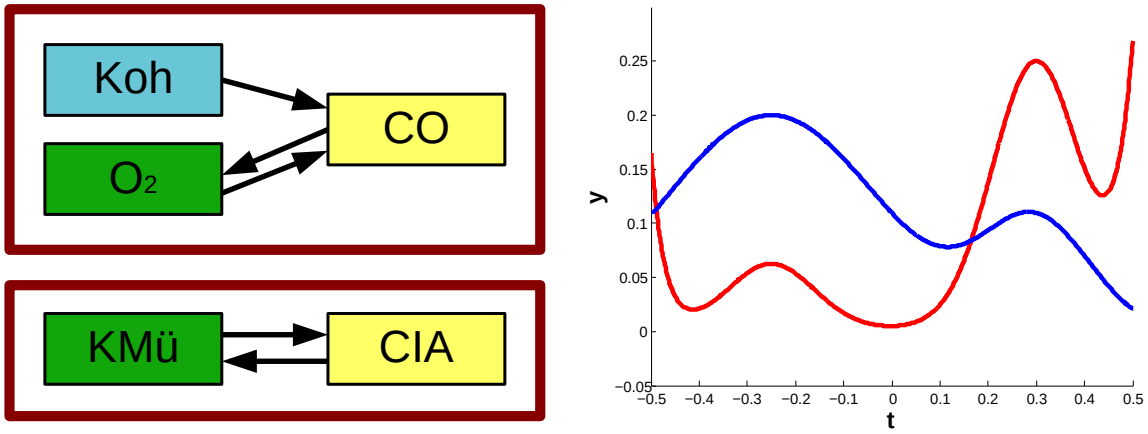

Abbildung 2.7: Darstellung der nichthierarchisch funktionalen Problemdekomposition am Beispielproblem. Das linke Bild zeigt die Abhängigkeiten der Problemgrößen und ihre Partitionierung. So könnte das Problem in zwei voneinander unabhängige Teilprobleme partitioniert werden (rote Umrandung). Das Bild rechts zeigt den zeitlichen Verlauf der Approximation von zwei der Problemgrößen. Sie würden, abhängig von ihrer Partitionszugehörigkeit, jeweils von einem der zwei neuronalen Netzen (blau bzw. rot dargestellt) approximiert werden.

Das Ziel der nichthierarchisch funktionalen Problemdekomposition ist die Partitionierung der Problemgrößen in mehrere, voneinander unabhängige Partitionen. Jede dieser Partitionen kann dann als Teilproblem angesehen werden und wird entsprechend jeweils von einem eigenen neuronalen Netz approximiert. So lässt sich

das Beispielproblem, wie in Abbildung 2.7 links gezeigt, in zwei Teilprobleme zerlegen, bestehend zum Einen aus einer Partition mit dem Kohlenmonoxidwert und den beiden darauf Einfluss nehmenden Größen Luftstrom und Kohlemenge, und zum Anderen in eine zweite Partition mit der Feinkörnigkeit der Kohlemühle und der Restkohle in der Asche. Wie in der Grafik zu sehen, bleiben dabei alle Abhängigkeiten vorhanden, ohne dass eine davon die Partitionsgrenzen übertritt. In Abbildung 2.7 rechts wird die Approximation zweier beliebiger Problemgrößen gezeigt. Anders als bei der zeitlichen Problemdekomposition aus Kapitel 2.3.1 werden sie über den kompletten Zeitraum vom selben neuronalen Netz approximiert, jedoch, wenn man annimmt, dass sie aus verschiedenen Partitionen stammen, jeweils jede Größe von einem Anderen.

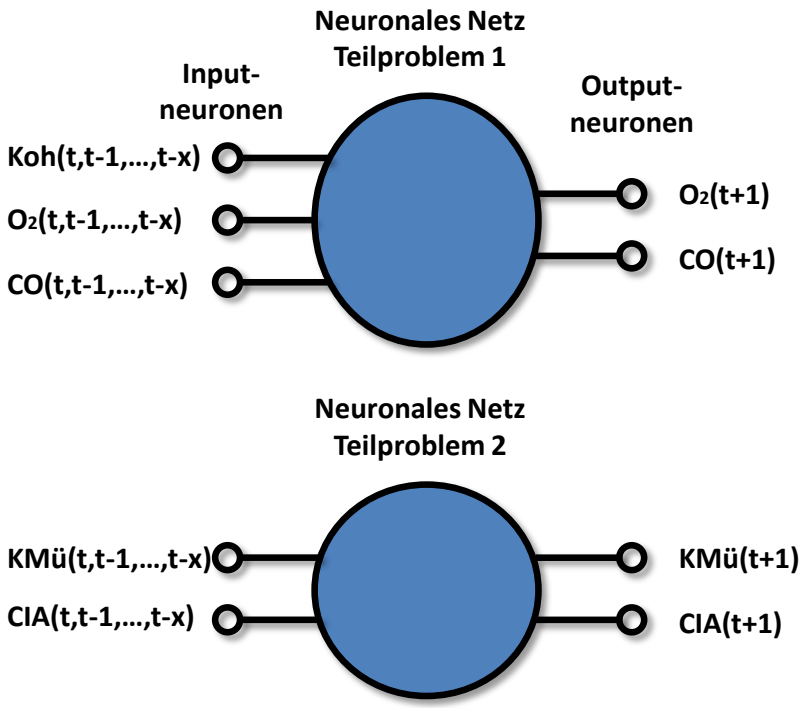

Abbildung 2.8: Architektur der nichthierarchisch funktionalen Problemdekomposition für das Beispielproblem.

Abbildung 2.8 zeigt die Architektur der neuronalen Netze für eine nichthierarchisch funktionale Problemdekomposition für das Beispielproblem. Dabei wird jede Partition,

bzw. jedes Teilproblem, von einem eigenen neuronalen Netz approximiert.

2.3.3 Hierarchisch funktionale Problemdekomposition

Bei der dritten und letzten Methode der Problemdekomposition, welche beispielsweise in [Duro et al., 2010] beschrieben ist, werden weder die Problemgrößen zeitlich zerlegt noch partitioniert. Stattdessen werden einzelne Problemgrößen in ihre funktionalen Bestandteile aufgespalten. Aus diesem Grund ist zur Erläuterung ein weiteres Beispiel vonnöten. Angenommen, eine Größe y setzt sich folgendermaßen zusammen:

$$y = 2\,e^{-x_2} + 2\,e^{-0.1\,((x_1-2)^2+(x_2-3)^2)} \tag{2.2}$$

Dann kann die Funktion in zwei Teilfunktionen y_1 und y_2 zerlegt werden:

$$y_1 = 2\,e^{-x_2} \tag{2.3}$$

$$y_2 = 2\,e^{-0.1\,((x_1-2)^2+(x_2-3)^2)} \tag{2.4}$$

womit sich y zu $y = y_1 + y_2$ ergibt. In Abbildung 2.9 sind y und ihre beiden Teilfunktionen grafisch veranschaulicht.

Außerdem ist es möglich die Hierarchie beliebig weiter fortzusetzen, wie der Name hierarchisch funktionale Dekomposition bereits nahelegt. So wäre es denkbar, y_2 in die Größen:

$$y_{21} = (x_1 - 2)^2 \tag{2.5}$$

$$y_{22} = (x_2 - 3)^2 \tag{2.6}$$

zu zerlegen, und diese durch ein Kombinationsnetzwerk zu:

$$y_2 = 2\,e^{-0.1\,(y_{21}+y_{22})} \tag{2.7}$$

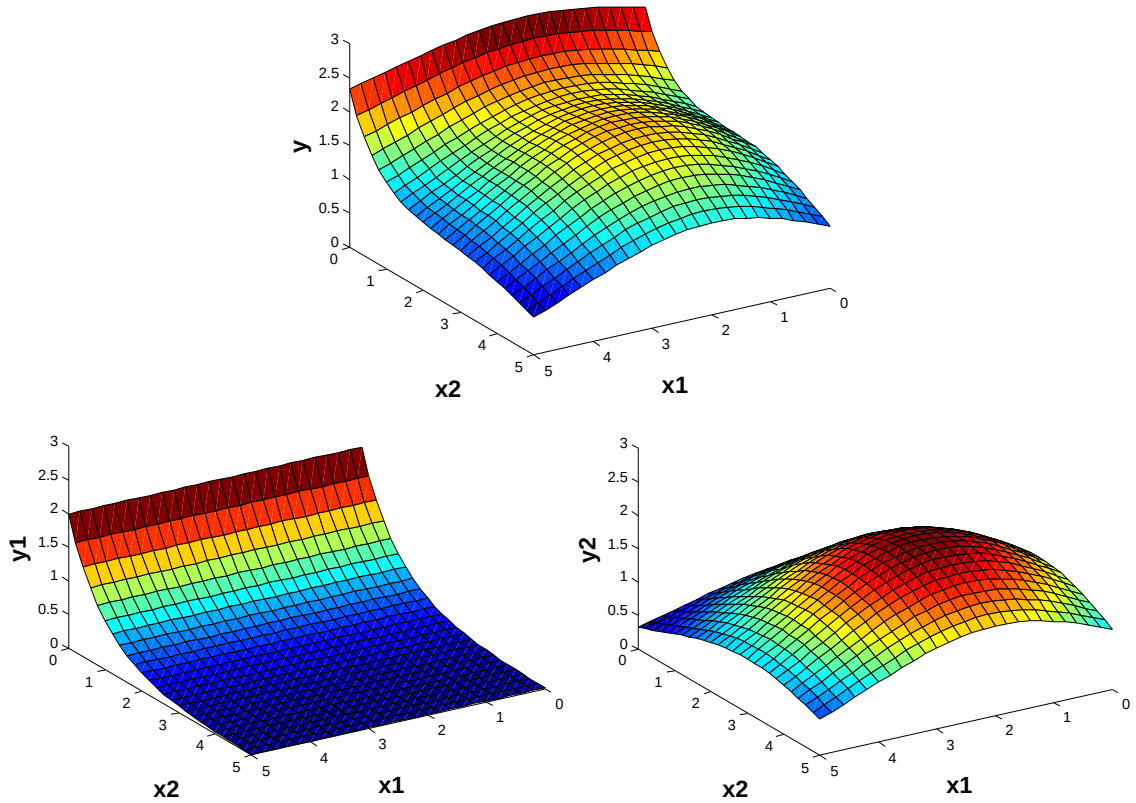

Abbildung 2.9: Veranschaulichung der hierarchisch funktionalen Problemdekomposition anhand des funktionalen Verlaufs der Problemgrößen. Die ursprüngliche Problemgröße (oberes Bild) wird zerlegt in die Summe zweier Teilprobleme (unteres linkes und unteres rechtes Bild).

zusammenzuführen.

Die Hoffnung hinter dieser Art der Problemzerlegung ist, dass die Teilfunktionen für das neuronale Netz leichter zu handhaben sind und damit bessere Lösungen gefunden werden, z.B. durch einen günstigeren Verlauf der Teilfunktionen oder eine Abhängigkeit von weniger Problemgrößen als das Gesamtproblem. So hängt beispielsweise y_1 nur von x_2, nicht aber von x_1 ab. Auch können auf diese Weise eventuell vorhandene versteckte Problemgrößen gefunden werden.
Die Architektur für eine solche Problemzerlegung wird in Abbildung 2.10 gezeigt. Mehrere neuronale Netze, die sogenannten Modulnetzwerke, approximieren die Teil-

funktionen y_1 und y_2. Diese wiederum werden durch ein Kombinationsnetzwerk zur endgültigen Lösung der Problemgröße y kombiniert.

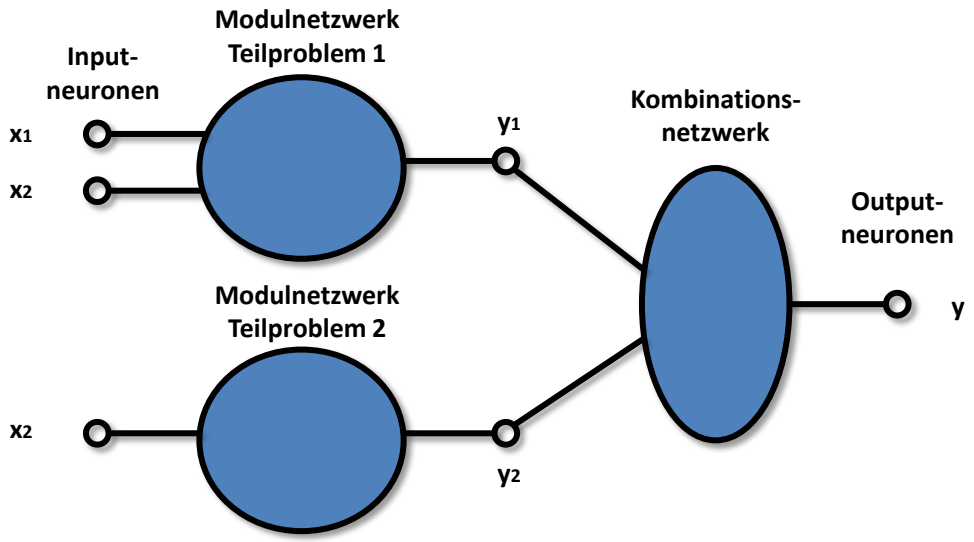

Abbildung 2.10: Architektur der hierarchisch funktionalen Problemdekomposition für das Beispielproblem. Die Ausgaben der beiden Modulnetzwerke, welche die Teilprobleme approximieren, werden durch das Kombinationsnetzwerk zusammengefasst.

Eine hierarchisch funktionale Problemdekomposition kann beispielsweise mit Hilfe von koevolutionären Verfahren durchgeführt werden. Verschiedene Möglichkeiten dafür werden in [Khare et al., 2005], [Nguyen et al., 2006] und [Potter und De Jong, 2000] vorgestellt.

Die Methode der Problemzerlegung, welche im APD-Projekt angestrebt wird, ist die nichthierarchisch funktionale Problemdekomposition, also die Partitionierung der Problemgrößen in voneinander unabhängige Partitionen, weshalb sich auch im weiteren Verlauf dieser Arbeit ausschließlich darauf beschränkt wird. Weiterhin ist der Einfachheit halber, wenn im Folgenden von einer Problemzerlegung oder Problemdekomposition die Rede ist, die nichthierarchisch funktionale Problemdekomposition gemeint.

2.4 Vorteil einer Problemzerlegung

Um den Grund für die Problemzerlegung und die Motivation hinter dieser Arbeit zu verdeutlichen, sollen hier die Vorteile einer Problemzerlegung aufgezählt und anhand von Beispielen veranschaulicht werden.

Der Vorteil einer Problemzerlegung im Allgemeinen und der nichthierarchisch funktionalen Problemdekomposition im Speziellen ist vor allem die Verringerung der Komplexität. Durch die Partitionierung der Problemgrößen erreicht man, dass Problemgrößen aus einer Partition keinen Einfluss auf Problemgrößen aus anderen Partitionen besitzen. Zu sehen ist dies in Abbildung 2.12. So hat zum Beispiel im zerlegten Problem rechts die Kenngröße 1 (K1) keinen Einfluss mehr auf Messgröße 2 (M2) und Regelgröße 2 (PID2) und wird entsprechend auch für deren Training nicht mehr verwendet. Dadurch verringert sich für die Teilprobleme die Dimensionalität des Inputraums. Dies wiederum führt zum Einen zu einem niedrigdimensionaleren und damit einfacheren Fehlergebirge und zum Anderen zu einer höheren Punktdichte der Trainingsdaten, was die Suche nach dem Optimum für den Trainingsalgorithmus wesentlich erleichtern kann. Abbildung 2.11 illustriert die Verringerung der Punktdichte mit zunehmender Dimensionalität. Dafür wurden in einem ein-, zwei- und dreidimensionalen Raum jeweils zehn zufällige Datenpunkte im Bereich $[0, 1]$ generiert. Die mittleren Abstände der Punkte zum nächsten Nachbarn betragen dabei 0.0513 im eindimensionalen, 0.2145 im zweidimensionalen und 0.3560 im dreidimensionalen Raum. Durch diese deutlich niedrigeren Punktdichten in höherdimensionalen Räumen kann die Funktion feinmaschiger und damit genauer abgeschätzt werden.

Ein weiterer Vorteil ist, dass auch kleinere neuronale Netze möglich sind. Ein neuronales Netz, welches nur einen Teil des Gesamtproblems löst und auch nur einen Teil der Inputgrößen bekommt, benötigt natürlich weitaus weniger Neuronen und Gewichte als eines, welches das komplette Problem löst, was wiederum eine geringere Anzahl an zu optimierenden Parametern für den Trainingsalgorithmus bedeutet und außerdem zu einer geringeren Wahrscheinlichkeit für Overfitting und einer höheren

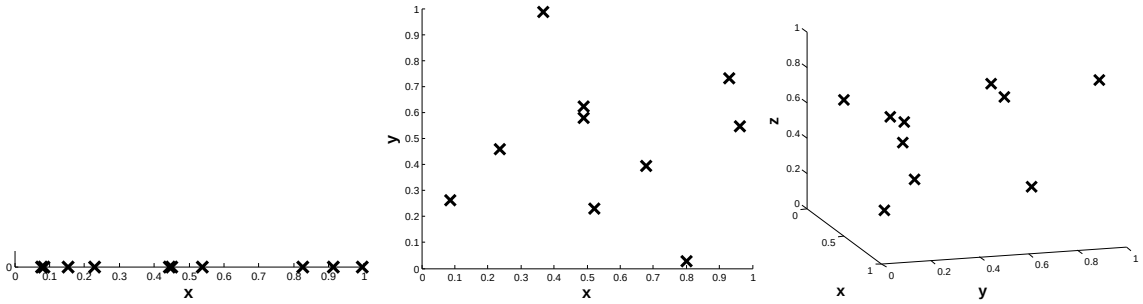

Abbildung 2.11: Veranschaulichung der Verringerung der Punktdichte in höherdimensionalen Räumen. Dargestellt sind zehn zufällig erzeugte Datenpunkte im Bereich $[0, 1]$ in jeweils einem eindimensionalen (links), zweidimensionalen (mitte) und dreidimensionalen Raum (rechts). Der mittlere Abstand zum nächsten Nachbarn beträgt von links nach rechts: 0.0513 , 0.2145 und 0.3560.

Generalisierungsfähigkeit führt.

Dies soll einmal anhand eines Beispiels nachgewiesen werden. Dafür wurde der Datensatz für Problem 1g aus Kapitel 5.1 verwendet. Dieser ist künstlich erzeugt worden, die Struktur der Abhängigkeiten entspricht jedoch der des Beispielproblems aus dem Kraftwerk, welche in Abbildung 2.2 zu sehen ist. Demzufolge lässt sich das Problem in zwei Teilprobleme und zwei neuronale Netze zerlegen, welche in Abbildung 2.12 rechts abgebildet sind. Links daneben ist das zum Vergleich verwendete monolithische neuronale Netz dargestellt.
Beide Architekturen, sowohl die monolithische als auch die zerlegte, wurden mit dem Levenberg-Marquardt Lernverfahren mit Bayes'scher Regularisierung (für Details siehe Kapitel 4.2.1) auf das Problem trainiert, der Root Mean Squared Error (RMSE) berechnet und über 100 Versuche gemittelt. Die Ergebnisse sind in Tabelle 2.1 zu sehen. Der Root Mean Squared Error ist mit den getrennten neuronalen Netzen leicht besser, auch wenn der Unterschied bei diesem recht kleinen Problem relativ gering ausfällt. Der Zeitaufwand für das Training beträgt jedoch nur etwa 40% der eines monolithischen neuronalen Netzes.
Die oben genannten Vorteile resultieren demnach in einem schnelleren Training und

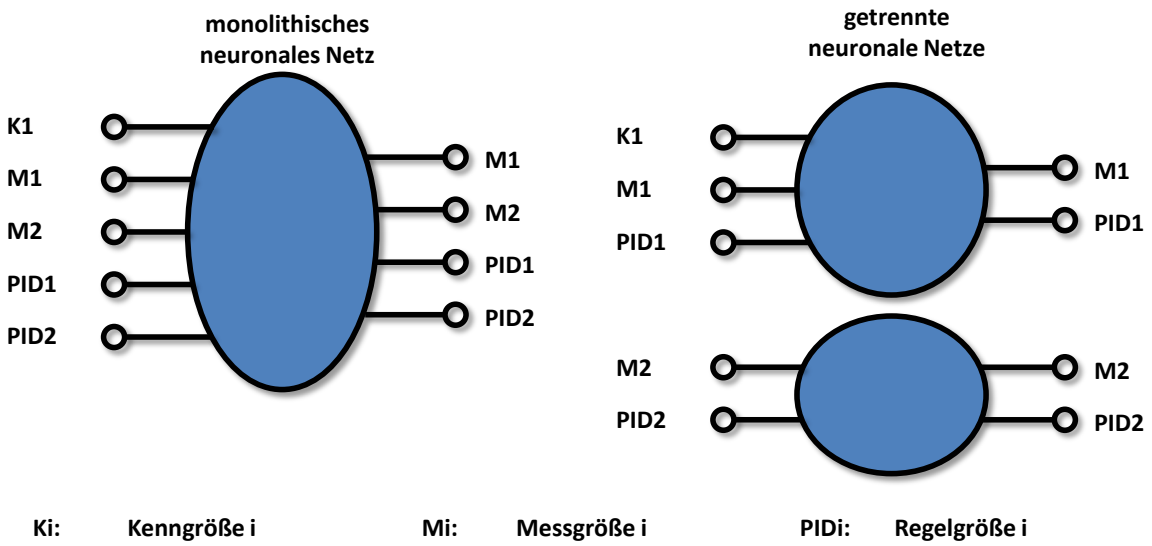

Abbildung 2.12: Veranschaulichung des Nutzens einer Problemzerlegung anhand der Gegenüberstellung eines monolithischen neuronalen Netzes und zweier, entsprechend den Abhängigkeiten zerlegter, neuronaler Netze für Beispielproblem 1g.

besseren Lösungen für das zu optimierende Problem. Darüber hinaus können von den gefundenen Strukturen der Teilprobleme auch die menschlichen Experten profitieren. Sie können zu einem besseren Verständnis des Prozesses beitragen und, durch den nun zum Teil modularen Aufbau, selbst Expertenwissen einbringen.

2.5 Zusammenfassung Problemstellung

In diesem Kapitel wurde ein Beispielproblem eingeführt, anhand dessen die Problemstellung, die Notationen und die Methoden der Problemzerlegung erläutert wurden.

Man kann ein Problem *zeitlich* zerlegen, wobei weiterhin das gesamte Problem von einem neuronalen Netz approximiert wird, jedoch, je nach Systemzustand, von einem anderen neuronalen Netz. Die Netze spezialisieren sich dabei auf die Regelung des Systems in unterschiedlichen Situationen. Eine weitere Möglichkeit ist die *nichthierarchisch funktionale* Problemdekomposition, bei der die Problemgrößen so partitioniert

	monolithisches neuronales Netz	getrennte neuronale Netze
mittlerer RMSE:	0.0221	0.0212
mittlerer Zeitaufwand für das Training:	0.506s	0.206s

Tabelle 2.1: Ergebnisse der Gegenüberstellung eines monolithischen mit zwei, entsprechend den Teilproblemen zerlegten, neuronalen Netzen. Gezeigt wird der über 100 Versuche gemittelte Root Mean Squared Error (RMSE) und der mittlere Zeitaufwand für ein komplettes Training.

werden, dass sie keine Abhängigkeit zu Größen anderer Partitionen besitzen. Die neuronalen Netze können dabei auf jede Partition einzeln trainiert werden. Schließlich gibt es weiterhin die *hierarchisch funktionale* Problemdekomposition, bei der einzelne Problemgrößen in ihre funktionalen Bestandteile zerlegt werden. Die Approximation dieser funktionalen Teilprobleme geschieht durch sogenannte Modulnetze, deren Outputs durch ein Kombinationsnetz zur Gesamtapproximation der Problemgröße zusammengesetzt werden. In dieser Arbeit wird sich nachfolgend allerdings ausschließlich auf die nichthierarchisch funktionale Problemdekomposition beschränkt.

Die *Vorteile einer Problemdekomposition* sind vor allem die Verringerung der Komplexität und die geringere Anzahl an Neuronen und Gewichten für die neuronalen Netze. Dies resultiert in einem deutlich schnelleren Lernvorgang und besseren Lösungen für das Problem.

Kapitel 3

Grundlagen

In diesem Kapitel werden einige für die weitere Arbeit wichtige Grundlagen erläutert und Grundbegriffe geklärt. Dazu gehört zum Einen die Evolution bzw. das Gebiet der evolutionären Algorithmen, welche von dieser abgeleitet sind, und zum Anderen das Levenberg-Marquardt Lernverfahren, welches im Kapitel 4 zum Training der neuronalen Netze eingesetzt wird. Außerdem wird der Begriff der Mutual Information, welche für die Time Delay Estimation in Kapitel 4.4.1 benötigt wird, erläutert und eine Methode zu deren Abschätzung vorgestellt.

3.1 Evolution

Der Lösungsansatz zur Problemdekomposition, welcher Gegenstand dieser Arbeit ist, verwendet evolutionäre Algorithmen um ein großes Problem in mehrere kleine Teilprobleme zu zerlegen. Zum Verständnis der evolutionären Algorithmen, welche im Kapitel 3.2 besprochen werden, wird im Folgenden zunächst einmal kurz auf die Evolution im Allgemeinen eingegangen.

Die Theorie zur Evolution wurde erstmals von Charles Darwin in seinem Werk *On the Origin of Species* [Darwin, 1859] vorgestellt und ist in [Mayr, 2005] in etwas modernerer Form zu finden. Sie stellt ein Prinzip aus der Biologie dar, nach welchem die Entwicklung der Arten beschrieben wird, also die Veränderung ihrer Merkmale

über mehrere Generationen hinweg.
Nach dieser Theorie findet die Evolution hauptsächlich durch natürliche Selektion statt, welche auf drei, von Darwin aufgestellten Prinzipien über Populationen beruht:

1. **Reproduktion**
   Eine Population erzeugt immer mehr Nachkommen als jemals überleben könnten. Es findet also, vor allem aufgrund des beschränkten Platz- und Nahrungsangebotes, ein Konkurrenzkampf statt.

2. **Variation**
   Individuen sind niemals gleich. Sie unterscheiden sich, selbst innerhalb einer Population, was u.a. zu unterschiedlichen Überlebenswahrscheinlichkeiten und Reproduktionsraten führt.

3. **Vererbung**
   Die Merkmale der Individuen sind, zumindest bis zu einem gewissen Grad, vererbbar.

Aus diesen Prinzipien ensteht der Mechanismus der Evolution durch natürliche Selektion, welchen Darwin mit „Survival of the Fittest“ (dt. „Überleben des Bestangepassten“) zusammengefasst hat. Individuen mit Merkmalen, welche für das Überleben und die Fortpflanzung vorteilhaft sind, beispielsweise erhöhte Fluchtchancen gegen Raubtiere, Resistenz gegen Krankheiten usw., produzieren viele Nachkommen und geben ihre Merkmale an ihre Nachkommen weiter. Man sagt, sie haben eine hohe Fitness, also hohe Chancen im Konkurrenzkampf. Schlecht angepasste Individuen hingegen gehen im Konkurrenzkampf unter, sie sterben frühzeitig und/oder produzieren keine oder weniger Nachkommen. Sie haben somit eine geringe Fitness. Dies führt auf lange Sicht zu einer Verbreitung der vorteilhaften Merkmale unter der Art.

Dieser Prozess wird in Abbildung 3.1 in einem vereinfachten Beispiel illustriert. Zu sehen sind mehrere kurzhalsige und eine langhalsige Giraffe. Die langhalsige Giraffe befindet sich gegenüber ihren Artgenossen im Vorteil, weil sie Blätter auch von hohen Bäumen fressen kann. Wenn für dieses Beispiel angenommen wird, dass die Länge des

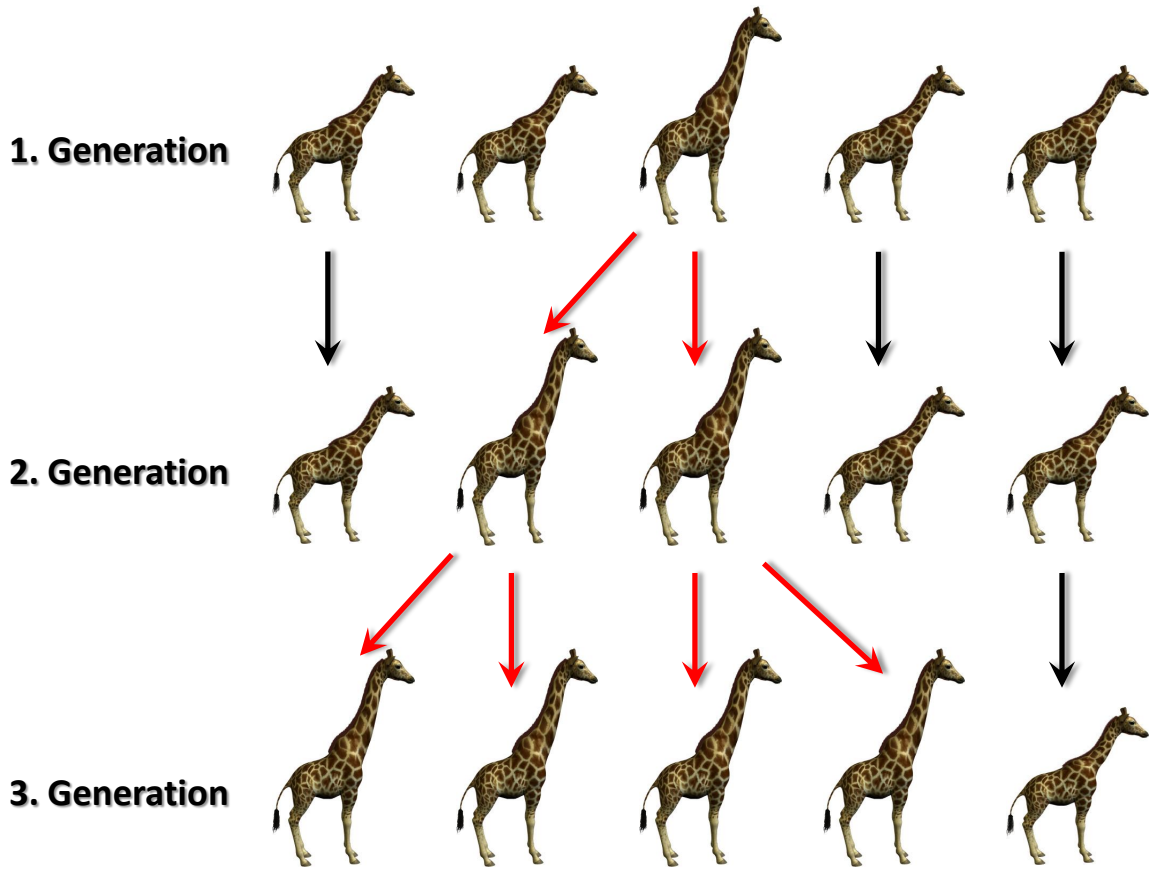

Abbildung 3.1: Darstellung der Ausbreitung vorteilhafter Merkmale über mehrere Generationen anhand von langhalsigen und kurzhalsigen Giraffen. Die Pfeile zeigen die Abstammung, wobei ein roter Pfeil die Vererbung einer langen Halslänge anzeigt.

Halses das einzige für das Überleben und die Fortpflanzung wichtige Merkmal ist, setzt sich die langhalsige Giraffe im Konkurrenzkampf gegenüber ihren Artgenossen durch. Sie lebt länger und produziert mehr Nachwuchs, während die Anderen, aufgrund ihres kurzen Halses, verhungern und wenig bis gar keinen Nachwuchs hervorbringen. Das Merkmal der Halslänge wird dabei an die Nachkommen vererbt und schon in der dritten Generation ist zu sehen, dass bereits die Mehrzahl der überlebenden Giraffen einen langen Hals besitzt. Die roten Pfeile zeigen dabei, wie sich das Merkmal der langen Halslänge über die Zeit ausbreitet.

3.2 Evolutionäre Algorithmen

Die evolutionären Algorithmen, welche bei den Verfahren in Kapitel 4 verwendet werden, gehören zur Klasse der sogenannten naturanalogen Verfahren. Sie lehnen sich an der biologischen Evolution an, welche im letzten Kapitel 3.1 beschrieben wurde, und dienen zur Lösung von Optimierungsproblemen. Eine umfassende Einführung in dieses Gebiet ist in [Jacob, 1997] zu finden.

Evolutionäre Algorithmen haben den Vorteil, dass sie praktisch kein Vorwissen über das zu optimierende Problem brauchen. Nötig sind nur zwei Dinge:

1. Eine *Fitnessfunktion*, welche die Eignung der Individuen als Lösung beschreibt. Wenn in der Welt der evolutionären Algorithmen Individuen mögliche Lösungsvorschläge für das Optimierungsproblem sind, dann stellt die Fitnessfunktion die Entsprechung zur Umgebung dar. Während bei der biologischen Evolution ungeeignete Individuen erkranken, verhungern oder Raubtieren zum Opfer fallen, gehen evolutionäre Algorithmen den humaneren Weg und geben schlechten Lösungen einfach einen geringen numerischen Fitnesswert, welcher die Güte des Individuums als Lösung beschreibt.

2. Eine Möglichkeit *Nachkommen* zu erzeugen, also aus einer oder mehreren Lösungen eine neue Lösung zu erzeugen, welche ihrem Vorgänger/ihren Vorgängern ähnelt, aber nicht genau gleicht. Dies geschieht durch *Rekombination*, also die Kombination der Merkmale der Elternindividuen, und durch *Mutation*, welche die zufällige Veränderung eines Merkmals bewirkt. Durch Erstere könnte, um wieder auf das Beispiel mit den Giraffen aus Kapitel 3.1 zurückzugreifen, aus einer langhalsigen, kurzbeinigen Giraffe und einer kurzhalsigen, langbeinigen Giraffe eine langhalsige, langbeinige Giraffe, wie in Abbildung 3.2 dargestellt, entstehen, oder aber auch eine kurzhalsige, kurzbeinige Giraffe. Bei der Mutation wiederum ist es möglich, dass aus einer mäßig langhalsigen Giraffe eine langhalsige Giraffe hervorgebracht werden kann, wie in Abbildung 3.3 zu sehen. Sowohl bei der Mutation als auch bei der Rekombination kann sowohl eine Verbesserung als auch

eine Verschlechterung der Merkmale auftreten. Man beachte, dass für die Rekombination mindestens zwei Elternindividuen vonnöten sind, deren Merkmale kombiniert werden, während die Mutation auch mit lediglich einem Elternteil möglich ist.

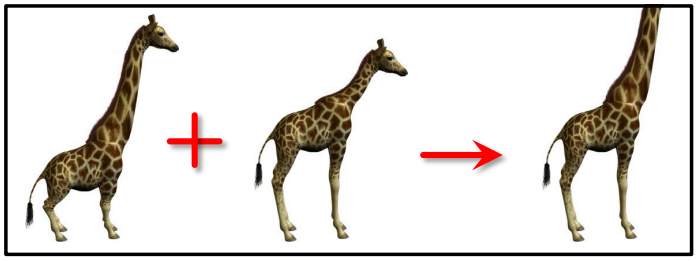

Abbildung 3.2: Veranschaulichung der Rekombination. Aus einer langhalsigen, kurzbeinigen Giraffe und einer kurzhalsigen, langbeinigen Giraffe kann eine langhalsige, langbeinige Giraffe hervorgehen. Die positiven Merkmale der beiden Eltern wurden an das Kind weitervererbt.

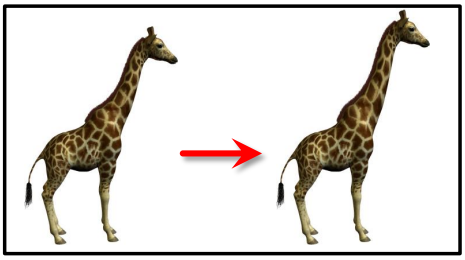

Abbildung 3.3: Veranschaulichung der Mutation. Aus einer mäßig langhalsigen Giraffe kann eine langhalsige Giraffe hervorgehen. Das Merkmal der Halslänge wurde zwar an das Kind weitergegeben, war allerdings einer zufälligen Veränderung unterworfen.

Die evolutionären Algorithmen machen sich zunutze, dass Individuen, welche nahe am Optimum liegen oder auch nur in einigen Merkmalen optimal sind ebenfalls schon eine recht hohe Fitness besitzen. Demnach haben Giraffen mit einem mäßig langen Hals eine erhöhte Überlebenschance, weil sie bereits an höher gelegene Nahrungsquellen

herankommen als ihre Artgenossen mit einem kurzen Hals. Das Gleiche gilt für Giraffen mit einem kurzen Hals, aber dafür beispielsweise sehr langen Beinen. Auf der Suche nach der „optimalen Giraffe“, wenn man bemerkt, dass Giraffen mit mäßig langen Hälsen und Giraffen mit langen Beinen schon eine recht gute Fitness besitzen, weiß man, dass die „optimale Giraffe“ diesen beiden Varianten wahrscheinlich deutlich ähnlicher ist, als den Giraffen mit kurzen Hälsen und kurzen Beinen.
Durch die Weitergabe von Merkmalen an ihre Nachkommen, welche ihren Eltern sehr ähnlich sind und damit, wie oben beschrieben, ebenfalls eine gute Fitness besitzen, und die Tatsache, dass geeignete Individuen einen deutlichen Vorteil bei der Überlebenswahrscheinlichkeit und der Vermehrungsrate besitzen, ist es wahrscheinlich, dass die Population sich im Laufe der Zeit immer weiter dem Optimum nähert.

Es gibt eine ganze Reihe verschiedener Varianten einen evolutionären Algorithmus umzusetzen, hier soll sich aber auf die Grundvariante mit einer $(\mu + \lambda)$-Strategie beschränkt werden. μ gibt dabei die Größe der Population an, während λ die Anzahl der Nachkommen in jeder Generation darstellt. Um die Größe der Population konstant auf genau μ Individuen zu halten, beschreibt sie gleichzeitig die Anzahl der aussterbenden Individuen. Das $+$ weist darauf hin, dass die Folgegeneration sich aus den besten Individuen sowohl der Nachkommen, als auch ihren Eltern zusammensetzen kann. Die Elternindividuen in der Folgegeneration müssen also, anders als beispielsweise bei einer (μ, λ)-Strategie, nicht zwangsweise sterben.

Der Ablauf eines evolutionären Algorithmus wird in Abbildung 3.4 schematisch in einem Diagramm veranschaulicht. Zu Beginn wird eine Startpopulation von μ Individuen erstellt, für welche meist zufällig ausgewählte Lösungen über den gesamten Lösungsraum ausgewählt werden. Diese Individuen werden dann mit Hilfe der Fitnessfunktion auf ihre Eignung als Lösung des Problems getestet. Durch die so berechneten Fitnesswerte können λ zufällige Individuen selektiert werden, aus welchen Nachkommen hervorgehen sollen. Dabei werden gewöhnlich Individuen mit einer hohen Fitness bevorzugt ausgewählt. Die Nachkommen werden durch Mutation und Rekombination wie oben beschrieben aus ihren Elternindividuen erzeugt und

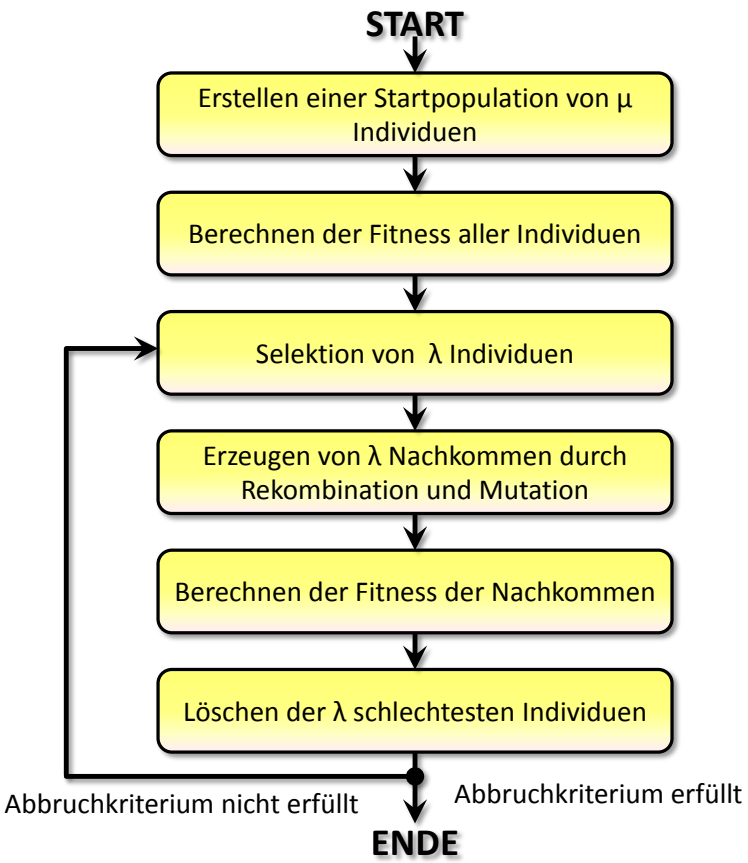

Abbildung 3.4: Darstellung des grundlegenden Ablaufs eines evolutionären Algorithmus mit einer $(\mu + \lambda)$-Strategie als Diagramm.

ihre Fitness berechnet. Aus der Population der Eltern- und Kinderindividuen werden schließlich, um die Größe der Population konstant zu halten, die λ schlechtesten Individuen wieder gelöscht und erneut mit der Selektion fortgefahren. Dieser Vorgang wird so lange wiederholt, bis ein vorher definiertes Abbruchkriterium erfüllt ist, beispielsweise eine vorgegebene Anzahl an Generationen oder eine bestimmte Lösungsgüte. Die Lösungen werden während dieser Zeit immer besser und nähern sich, mit genügend Iterationen, im Idealfall schließlich dem Optimum.

Im Folgenden soll auf einige wichtige Teilaspekte der evolutionären Algorithmen detaillierter eingegangen werden. Dabei wird weniger eine Übersicht über alle vorhandenen

Methoden gegeben, sondern es werden nur die in dieser Arbeit verwendeten und bei den Tests implementierten Varianten vorgestellt:

3.2.1 Selektion

Ein wichtiger Aspekt der Evolution ist die Selektion, also die Auswahl der Individuen, welche Nachkommen produzieren. Hierbei sollten Individuen mit hoher Fitness bevorzugt werden. Im einfachsten Fall werden hierfür einfach die λ Individuen mit der höchsten Fitness ausgewählt. Individuen mit annähernd gleicher Fitness liegen jedoch oftmals nahe beieinander. Auch ist die Wahrscheinlichkeit relativ hoch, dass die Individuen mit der höchsten Fitness voneinander abstammen. Werden aber ausschließlich ähnliche Individuen vermehrt, verringert sich die Varianz der Individuen, der Suchraum wird eingeschränkt und es findet keine Exploration mehr statt, was vor allem zu Beginn der Evolution von Nachteil sein kann. Die Individuen sammeln sich gleich zu Anfang um ein lokales Optimum und versäumen es, den Lösungsraum nach besseren Optima zu durchsuchen. Deshalb gibt es eine Reihe von weiteren Selektionsstrategien, welche unter Anderem in [Wakunda, 1995] beschrieben werden. Die Variante, die für diese Arbeit verwendet wird, ist die lineare Rangauswahl.

Bei der Rangauswahl werden die Individuen nach ihrer Fitness absteigend sortiert. Über diese Rangordnung wird eine Wahrscheinlichkeitsverteilung definiert, welche aussagt, mit welcher Wahrscheinlichkeit das Individuum Nachkommen erzeugen darf. Je kleiner dabei die Rangnummer, desto geringer ist der Wert dieser Wahrscheinlichkeitsverteilung. Dadurch, dass jedoch auch Individuen mit schlechter Fitness eine geringe Chance haben ausgewählt zu werden, erhöht sich die Varianz der Nachkommen. Bei der linearen Rangauswahl ist die Wahrscheinlichkeitsverteilung für Rang i definiert durch:

$$p_{Sel}(i) = \frac{\lambda - i + 1}{\sum_{j=1}^{\lambda} (\lambda - i + 1)} \tag{3.1}$$

Dies resultiert in einer linear abfallenden Wahrscheinlichkeit, wobei das Individuum mit der höchsten Fitness eine λ-mal höhere Wahrscheinlichkeit besitzt als das schwächste Individuum. In Abbildung 3.5 ist die Wahrscheinlichkeitsverteilung einmal für $\lambda = 10$ abgebildet. Die Wahrscheinlichkeit für das Individuum mit Rang 1 ist demnach 10-mal so hoch, wie für das mit Rang 10.

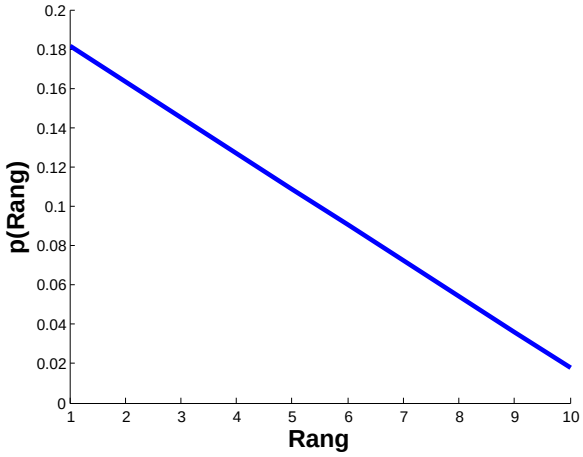

Abbildung 3.5: Darstellung der Wahrscheinlichkeitsverteilung bei der linearen Rangauswahl am Beispiel von $\lambda = 10$.

3.2.2 Mutation

Die Mutation ist, wie bereits beschrieben, die zufällige Änderung von Merkmalen bei der Reproduktion. In evolutionären Algorithmen wird dies meist durch die Addition eines gaußverteilten Zufallsvektors mit dem Erwartungswert 0 realisiert. Kritisch ist dabei jedoch die Wahl der Varianz der verwendeten Gaußverteilung. Eine hohe Varianz resultiert in stark veränderten Merkmalen und damit in einer Durchsuchung des Lösungsraumes. Eine geringe Varianz jedoch, und damit eine geringe Veränderung der Merkmale, hilft guten Individuen, ein eventuell in der Nähe befindliches Optimum zu finden. Die beiden Varianten schließen sich gegenseitig aus, sind jedoch beide für eine erfolgreiche Evolution vonnöten. Dieses Problem ist als Exploration-Exploitation-Dilemma bekannt. Erstrebenswert ist es, zu Beginn der Evolution den Lösungsraum zu durchsuchen, später aber zunehmend die bis dahin gefundenen guten Lösungen aus-

zunutzen und die Optima in deren Nähe zu finden.
Eine Möglichkeit dies zu tun ist die lokale Varianzanpassung nach Nissen [Nissen, 1997]. Die Idee dabei ist es, die Varianz des gaußverteilten Zufallsvektors bei der Mutation nur als einen weiteren Parameter zu betrachten, welcher durch die Evolution selbst optimiert werden kann. Die Varianz wird also mit in die Chromosomen aufgenommen, wobei gewöhnlich für jedes Gen eine eigene Varianz gespeichert wird. Entsprechend der Wirkungsweise der Evolution sterben dadurch Individuen mit unpassender Varianz mit der Zeit aus. Ist die Varianz zu klein, durchsuchen sie den Lösungsraum nicht ausreichend und entwickeln sich zu langsam. Ist sie dagegen zu groß, entfernen die Individuen sich zu weit von ihren Eltern und nutzen deren „Wissen“ über ein eventuell in der Nähe befindliches Optimum nicht aus. Individuen mit optimaler Varianz hingegen produzieren auch gute Nachkommen und geben ihre Varianz an diese weiter.
Die Mutation des Vektors der Varianzen für die Gene selbst kann nach Nissen wie folgt vorgenommen werden:

$$\sigma_i^{neu} = \sigma_i^{alt} \cdot exp\,(\,r_1 \cdot N(0,1) + r_2 \cdot N_i(0,1)\,) \tag{3.2}$$

Wobei σ_i die Standardabweichung für Gen i darstellt. $N(0,1)$ ist eine normalverteilte Zufallszahl, welche einmal pro Chromosom bestimmt wird, während $N_i(0,1)$ zusätzlich für jedes Gen i einzeln ausgewürfelt wird. r_1 und r_2 sind Parameter, welche Nissen mit $r_1 = 0,1$ und $r_2 = 0,2$ empfiehlt.

3.2.3 Rekombination

Die Rekombination ist die Kombination der Merkmale zweier oder mehrerer Elternindividuen. Die Hoffnung dabei ist, dass dadurch Nachkommen entstehen, welche die besten Merkmale der Eltern in sich vereinen.
In dieser Arbeit wurde der sogenannte Uniform-Crossover [Debes, 2010] verwendet, welcher gewöhnlich die Merkmale zweier Elternindividuen kombiniert und zwei Nachkommen hervorbringt. Dafür werden die beiden Eltern erst kopiert und anschließend wird ein Zufallsvektor erzeugt, welcher für jedes Gen einzeln auswürfelt, ob das

Merkmal zwischen den beiden kopierten Individuen ausgetauscht wird oder nicht. Die Austauschwahrscheinlichkeit wird dabei gewöhnlich zwischen $0,5$ und $0,8$ festgelegt. Abbildung 3.6 illustriert dies anhand eines Beispiels mit zwei Individuen, welche jeweils fünf binäre Merkmale besitzen (links). Darunter wird der ausgewürfelte Zufallsvektor gezeigt, welcher angibt, ob der Austausch stattfinden soll oder nicht. Rechts sind die beiden daraus entstehenden Nachkommen zu sehen.

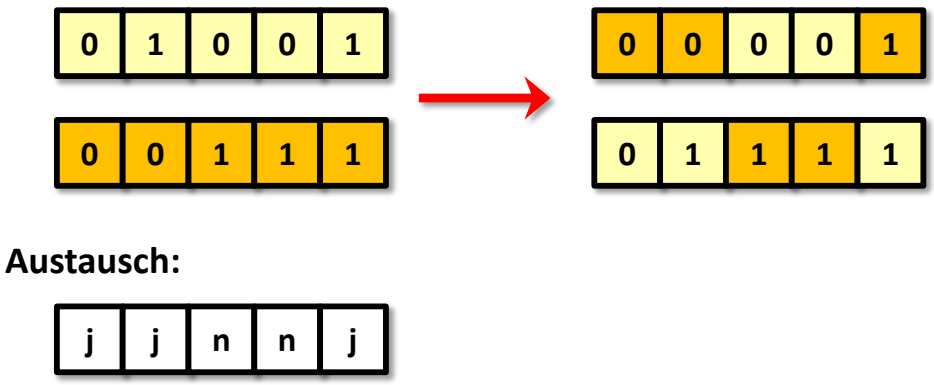

Abbildung 3.6: Darstellung des Uniform-Crossovers. Links sind die beiden Elternindividuen zu sehen, zusammen mit dem Zufallsvektor, welcher bestimmt, ob ein Austausch der Gene stattfinden soll (j) oder nicht (n). Rechts abgebildet sind die beiden daraus entstandenen Nachkommen.

3.3 Levenberg-Marquardt Lernverfahren

Für die Approximation der Problemgrößen und damit auch zur Fitnessberechnung bei der Evolution für die Verfahren zur Problemzerlegung (siehe Kapitel 4.1) sollen in dieser Arbeit neuronale Netze verwendet werden. Zum Training solcher neuronaler Netze wird meist das Backpropagation of Error Verfahren verwendet. Das Levenberg-Marquardt Lernverfahren zeichnet sich jedoch im Vergleich zum Backpropagation of Error Verfahren gewöhnlich durch ein schnelleres Konvergieren, damit auch eine kürzere Laufzeit und durch einen geringeren Approximationsfehler bei den Lösungen aus. Da außerdem einige der in dieser Arbeit verwendeten Verfahren, beispielsweise

der Optimal Brain Surgeon Algorithmus in Kapitel 4.2.2 und die Bayes'sche Regularisierung aus Kapitel 4.2.1, die Inverse der Hessematrix benötigen, welche vom Levenberg-Marquardt Lernverfahren quasi als Nebeneffekt automatisch abfällt, soll im Folgenden das Levenberg-Marquardt Lernverfahren zum Training der neuronalen Netze verwendet werden.

Vorgestellt wird das Levenberg-Marquardt Verfahren unter Anderem in [Bishop, 1995]. Es ist eine Variante des Newton-Verfahrens und speziell dazu entworfen, quadratische Fehlerfunktionen, zum Beispiel die Summe der quadratischen Fehler (SSE), zu minimieren:

$$E = SSE = \frac{1}{2}\sum_{i=1}^{N}\left(y_(x_i) - t_i\right)^2 = \frac{1}{2}\left\|\epsilon\right\|^2 \tag{3.3}$$

Mit $\epsilon = y - t$. Wobei E der Netzwerkfehler ist, N die Anzahl der Pattern, y beschreibt einen Vektor mit den Netzausgaben und t einen Vektor mit den Trainingspattern. Weiterhin stellt $\|\cdot\|$ die euklidische Vektornorm dar. Wenn man nun annimmt, dass man momentan die Gewichte w_{old} in seinem neuronalen Netz besitzt, kann man eine Taylorreihenentwicklung der Fehlerfunktion ϵ bis zum linearen Term durchführen:

$$\epsilon\left(w_{new}\right) = \epsilon\left(w_{old}\right) + J\left(w_{new} - w_{old}\right) \tag{3.4}$$

wobei

$$J = \frac{\partial \epsilon_i}{\partial w_j}$$

die Jacobi-Matrix darstellt. Eingesetzt ergibt sich somit die neue quadratische Fehlerfunktion:

$$E = \frac{1}{2}\left\|\epsilon\left(w_{old}\right) + J\left(w_{new} - w_{old}\right)\right\|^2 \tag{3.5}$$

Nach einem Ableiten und Null setzen, kann man die Formel nach w_{new} umstellen und erhält so eine Updateregel für die Gewichte des Netzes:

$$w_{new} = w_{old} - \left(J^T J\right)^{-1} J^T \epsilon\left(w_{old}\right) \tag{3.6}$$

$J^T J$ ist dabei eine Approximation der Hessematrix H:

$$H = \frac{\partial^2 E}{\partial w_i \, \partial w_j}$$

Die Jacobimatrix J selbst lässt sich recht einfach durch Backpropagation of Error ermitteln.

Theoretisch kann man mit Hilfe dieser Formel iterativ die Gewichte eines neuronalen Netzwerkes minimieren. Praktisch jedoch wird nur eine lineare Approximation der einfachen Fehlerfunktion ϵ in Formel 3.4 verwendet, welche sich bei zu großen Schrittweiten oftmals als sehr ungenau erweist. Da diese Approximation bei kleinen Schrittweiten zunehmend genauer wird, versucht man, den Fehler nicht global, sondern lokal, innerhalb einer kleinen Umgebung um w_{old} zu minimieren und sich so mit Hilfe kleinerer Schrittweiten, Stück für Stück, dem Optimum anzunähern. Dies kann errreicht werden, indem man die Fehlerfunktion durch einen weiteren Term ergänzt:

$$E = \frac{1}{2} \left\|\epsilon\left(w_{old}\right) + J\left(w_{new} - w_{old}\right)\right\|^2 + \lambda \left\|w_{new} - w_{old}\right\|^2 \tag{3.7}$$

Die Wirkung dieser neuen Fehlerfunktion wird durch Abbildung 3.7 illustriert. Die schwarze Kurve zeigt eine beispielhafte SSE-Fehlerfunktion, während der zusätzliche Term von Formel 3.7 blau dargestellt ist. Dieser hat eine quadratische Form mit einem Minimum bei w_{old} (gestrichelte Linie). Im rechten Bild ist mit der roten Kurve die Summe der beiden Kurven dargestellt, welche der neuen Fehlerfunktion von Formel 3.7 entspricht. Die Struktur dieser neuen Fehlerfunktion ist in der Nähe von w_{old} gleich der SSE-Fehlerfunktion, steigt jedoch mit zunehmender Entfernung zu w_{old} zunehmend an. Ihr Minimum liegt somit nahe an w_{old}, nähert sich jedoch ebenfalls dem Optimum. Der Parameter λ steuert die Breite des quadratischen Terms und damit die Schrittweite. Ein großes λ ergibt eine steile quadratische Kurve, damit eine kleine

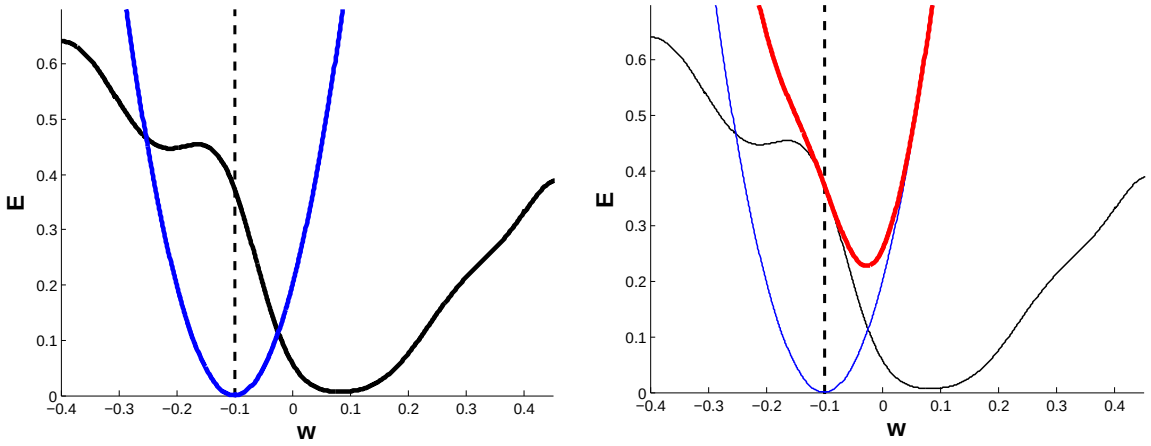

Abbildung 3.7: Darstellung der Zusammensetzung der Fehlerfunktion des Levenberg-Marquardt Algorithmus. Die schwarze Kurve zeigt die ursprüngliche SSE-Fehlerfunktion, während der zusätzliche Term zur Steuerung der Schrittweite blau dargestellt ist. Das rechte Bild zeigt die neue Fehlerfunktion, welche aus der Summe der beiden besteht (rot). Die gestrichelte Linie kennzeichnet die aktuellen Gewichte.

Schrittweite, aber auch eine genaue Approximation der Fehlerfunktion. Ein kleines λ dagegen resultiert in einer relativ flachen quadratische Kurve, damit auch einer großen Schrittweite und einer weniger genauen Approximation der Fehlerfunktion.
In der Praxis sollte dieser Parameter laufend entsprechend der aktuellen Situation angepasst werden. Eine verbreitete Möglichkeit ist es laut [Bishop, 1995], λ anfangs mit einem relativ beliebigen Wert, beispielsweise 0.1, zu initialisieren und später, abhängig von der Entwicklung der Fehlerfunktion, anzupassen. Steigt der Netzwerkfehler nach einer Iteration an, wird angenommen, dass die Schrittweite noch zu groß ist und der Parameter λ sollte um einen Faktor, beispielsweise 10, erhöht werden. Gleichzeitig wird das Gewichtsupdate, um keine Verschlechterung zuzulassen, wieder verworfen. Ist der Fehler nach einem Gewichtsupdate gesunken, wird dieses als erfolgreich angesehen, die Gewichte werden übernommen und λ um einen Faktor verringert.

Durch ein Ableiten und Nullsetzen der neuen Fehlerfunktion von Formel 3.7 erhält

man die Levenberg-Marquardt Updateregel für die Gewichte:

$$w_{new} = w_{old} - \left(J^T J + \lambda I\right)^{-1} J^T \, \epsilon \left(w_{old}\right) \tag{3.8}$$

wobei I die Einheitsmatrix darstellt.

Algorithmus 3.1 Levenberg-Marquardt Lernverfahren

Input e, $nmax$ ▷ e: Schranke für den Fehler ,$nmax$: maximale Anzahl Iterationen
Input t_i ▷ t_i Trainingspattern
$w_{old} \leftarrow$ INITIALISIERENEURONALESNETZ() ▷ zufällige Initialisierung der Gewichte
$SSE_{old} \leftarrow \frac{1}{2} \sum (\, y_i(w_{old}) - t_i)^2$ ▷ Summe der quadratischen Fehler
$\lambda \leftarrow 0.1$
$n \leftarrow 0$ ▷ Zähler für die Anzahl Durchläufe
repeat
  $n \leftarrow n + 1$
  $J \leftarrow$ BERECHNEJACOBIMATRIX() ▷ berechne Jacobimatrix mit Hilfe des
  ▷ Backpropagation of Error Algorithmus
  $w_{new} \leftarrow w_{old} - \left(J^T J + \lambda I\right)^{-1} J^T \, \epsilon \left(w_{old}\right)$
  $SSE_{new} \leftarrow \frac{1}{2} \sum (\, y_i(w_{new}) - t_i)^2$ ▷ Summe der quadratischen Fehler
  if $SSE_{new} \geq SSE_{old}$ **then** ▷ Fehler hat sich verschlechtert
    $\lambda \leftarrow \lambda \cdot 10$ ▷ erhöhe λ, verwirf Gewichtsupdate
  else ▷ Fehler hat sich verbessert
    $w_{old} \leftarrow w_{new}$ ▷ übernimm Gewichtsupdate
    $SSE_{old} \leftarrow SSE_{new}$
    $\lambda \leftarrow \lambda / 10$ ▷ verringere λ
  end if
until $SSE_{old} < e$ **or** $n \geq nmax$ ▷ bis Abbruchkriterium erreicht
return w_{old}

Algorithmus 3.1 fasst den Levenberg-Marquardt Algorithmus wie er hier beschrieben ist noch einmal als Pseudocode zusammen.

3.4 Mutual Information

Die Mutual Information $I(X;Y)$, im Deutschen auch Transinformation genannt, ist ein Maß für die Abhängigkeit von zwei Zufallsvariablen, im Folgenden X und Y genannt. Als solche ist sie in dieser Arbeit vor allem bei der Time Delay Estimation in Kapitel 4.4.1 von Nutzen. Sie wurde erstmals von Shannon [SHANNON, 1948] eingeführt und ist definiert als:

$$I(X;Y) = \int_x \int_y p(x,y)\, log \frac{p(x,y)}{p(x)\, p(y)}\, dx\, dy \tag{3.9}$$

Wobei $p(x)$ die Wahscheinlichkeitsdichtefunktion von X angibt.

Hergeleitet werden kann sie über die Entropien von $H(X)$, $H(Y)$ und die Verbundentropie $H(X,Y)$, wobei die Entropie bekanntlich ein Maß für die Unsicherheit einer Zufallsgröße X ist und damit auch ein Maß für den mittleren Informationsgehalt dieser Zufallsgröße. Sie wird definiert wird als:

$$H(X) = -\int_x p(x)\, log\, p(x)\, dx \qquad , H(X) \geq 0 \tag{3.10}$$

Analog definiert ist die Verbundentropie $H(X,Y)$, welche die Verallgemeinerung der Entropie für den bivariaten Fall darstellt:

$$H(X,Y) = -\int_x \int_y p(x,y)\, log\, p(x,y)\, dx\, dy \tag{3.11}$$

Und die bedingte Entropie $H(X|Y)$, welche die Unsicherheit einer Zufallsgröße X ist, wenn der Wert einer anderen Zufallsgröße bekannt ist:

$$H(X|Y) = -\int_x \int_y p(x,y)\, log\, p(x|y)\, dx\, dy \tag{3.12}$$

Mit Hilfe dieser Entropien kann die Mutual Information nach Shannon folgendermaßen dargestellt werden:

$$I(X;Y) = H(X) + H(Y) - H(X,Y) \tag{3.13}$$

Grafisch veranschaulichen lässt sich dieser Zusammenhang mit einem Venn-Diagramm, wie in Abbildung 3.8 zu sehen. Die Mutual Information liegt hier im Schnitt der beiden Kreise, welche die Entropien von X und Y darstellen und ist damit die mittlere Menge an Information, den eine Größe über die jeweils andere besitzt.

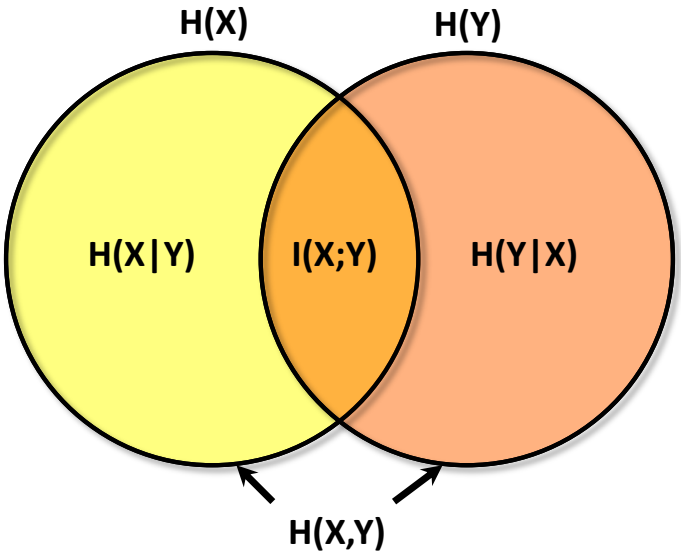

Abbildung 3.8: Darstellung des Zusammenhangs zwischen Entropie und Mutual Information.

3.4.1 Abschätzung der Mutual Information aus Daten

Leider ist es in der Praxis meist nicht möglich, die Mutual Information nach Formel 3.9 direkt zu berechnen, da die Wahrscheinlichkeitsdichtefunktionen $p(x)$, $p(y)$ und $p(x,y)$ gewöhnlich nicht bekannt sind und deshalb mit Hilfe der vorhandenen Daten erst abgeschätzt werden müssen. In [Kaltenhäuser, 2010] werden mehrere

Möglichkeiten vorgestellt eine solche Abschätzung entweder der Wahrscheinlichkeitsdichtefunktionen, oder der Mutual Information direkt vorzunehmen. Hier soll jedoch nur eine davon in Anwendung kommen, welche sich durch ihre Einfachheit und geringen Rechenaufwand auszeichnet und in der Praxis wohl am verbreitetsten ist: die Abschätzung der Wahrscheinlichkeisdichtefunktionen durch Histogramme nach Scott.

Histogramme werden konstruiert, indem der Wertebereich der Datenpunkte in k Bereiche, sogenannte Bins, unterteilt wird. Anschließend wird über jedem Bin ein Rechteck (bzw. ein Hypercube für mehrdimensionale Verteilungen) konstruiert, dessen Höhe sich aus der Anzahl der Datenpunkte $N(B(x))$ innerhalb des Bins, in welchem der Datenpunkt x liegt $B(x)$, geteilt durch die Gesamtanzahl der Datenpunkte N und die Binbreite W (bzw. Hypervolumen für mehrdimensionale Verteilungen) ergibt.
Als Ergebnis erhält man dann eine mehr oder minder exakte Approximation der Verteilung. Für die Berechnung der Mutual Information sind nur die ein- und zweidimensionalen Verteilungen für X und Y notwendig und es ergibt sich als Abschätzung:

$$\hat{p}(x) = \frac{N(\,B(x)\,)}{N \cdot W_X} \tag{3.14}$$

$$\hat{p}(y) = \frac{N(\,B(y)\,)}{N \cdot W_Y} \tag{3.15}$$

$$\hat{p}(x,y) = \frac{N(\,B(x,y)\,)}{N \cdot W_X \cdot W_Y} \tag{3.16}$$

Dabei stellt W_X die Binbreite für die Achse X dar.
Ein Beispiel für eine solche Abschätzung durch Histogramme wird in Abbildung 3.9 gezeigt, welcher eine ringförmige Verteilung zugrunde liegt.

Die einzige offene Größe, die verbleibt, ist die der Anzahl der Bins k, bzw. deren Breite W, deren Wahl für die Güte der Approximation entscheidend ist. Sie sollte

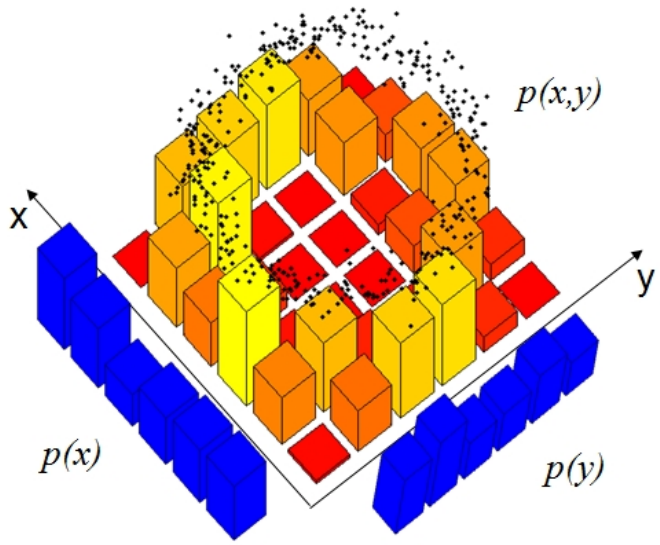

Abbildung 3.9: Approximation einer Beispielverteilung durch Histogramme. Die Randverteilungen sind durch blaue Bins approximiert worden, die Verbundverteilung über gelbe bis rote Bins. Bildquelle: [Gross und Schaffernicht, 2009]

einerseits klein genug sein, um die Feinheiten der zugrunde liegenden Verteilung gut approximieren zu können, anderseits aber auch breit genug, dass, aufgrund der in der Praxis beschränkten Anzahl Datenpunkte, in jedes Bin genügend Datenpunkte fallen, um die Verteilung statistisch korrekt wiedergeben zu können.
Für die Binbreite bietet sich die Regel nach Scott aus [Scott, 1992] an:

$$W = \frac{3,49 \cdot \sigma}{\sqrt[3]{N}} \tag{3.17}$$

Dabei gibt σ die Standardabweichung der Datenpunkte an.

Mithilfe dieser Formeln lassen sich die benötigten Histogramme erstellen, die Wahrscheinlichkeitsdichteverteilungen approximieren und somit die Mutual Information berechnen. Als Folge der Diskretisierung des Wertebereiches können letztendlich auch die Integrale zur Berechnung der Mutual Information weggelassen und durch einfache

Summen ersetzt werden:

$$I(X;Y) = \sum_x \sum_y P(x,y)\ log \frac{P(x,y)}{P(x)\,P(y)} \tag{3.18}$$

Wobei P die Wahrscheinlichkeit für ein Bin angibt und abhängig von der Fläche unter dem Bin ist: $P(x) = \widehat{p}(x) \cdot W_X$, bzw. im zweidimensionalen Fall: $P(x,y) = \widehat{p}(x,y) \cdot W_X \cdot W_Y$.

Algorithmus 3.2 zeigt dieses Vorgehen als Pseudocode. Anfangs werden verschiedene Kennwerte berechnet, welche zur Konstruktion des Histogramms wichtig sind, so die Binbreite W, die Anzahl der Bins k und die oberen und unteren Endpunkte des Histogramms, hier als MIN bzw. MAX bezeichnet. Anschließend wird in einer for-Schleife für jeden Datenpunkt berechnet, in welchem Bin sich dieser befindet und der Wert dieses Bins für jedes Histogramm um 1 erhöht. Nach der Normierung, so dass die Wahrscheinlichkeiten der Histogramme aufsummiert wieder 1 ergeben, wird die Mutual Information nach Formel 3.18 berechnet.

Algorithmus 3.2 Mutual Information Berechnung mit Hilfe von Histogrammen

Input $x[1..N]$, $y[1..N]$ ▷ N Datenpunkte
$W_x \leftarrow 3.49 \cdot \sigma_x / \sqrt[3]{N}$ ▷ Binbreite nach Scott
$W_y \leftarrow 3.49 \cdot \sigma_y / \sqrt[3]{N}$
$Min_x \leftarrow \bar{x} - \lceil (\bar{x} - min(x)) / W_x \rceil \cdot W_x$ ▷ Minimum des Histogramms
$Min_y \leftarrow \bar{y} - \lceil (\bar{y} - min(y)) / W_y \rceil \cdot W_y$
$Max_x \leftarrow \bar{x} + \lceil (max(x) - \bar{x}) / W_x \rceil \cdot W_x$ ▷ Maximum des Histogramms
$Max_y \leftarrow \bar{y} + \lceil (max(y) - \bar{y}) / W_y \rceil \cdot W_y$
$k_x \leftarrow (Max_x - Min_x)/W_x$ ▷ Anzahl der Bins
$k_y \leftarrow (Max_y - Min_y)/W_y$
$P_x[1..k_x]$, $P_y[1..k_y]$, $P_{xy}[1..k_x, 1..k_y] \leftarrow 0$ ▷ initialisieren der Histogramme
for $i = 1, ..., N$ **do** ▷ für alle Datenpunkte
  $bin_x \leftarrow \lceil (x[i] - Min_x)/W_x \rceil$ ▷ Bin für den aktuellen Datenpunkt
  $bin_y \leftarrow \lceil (y[i] - Min_y)/W_y \rceil$
  $P_x[bin_x] \leftarrow P_x[bin_x] + 1$
  $P_y[bin_y] \leftarrow P_y[bin_y] + 1$
  $P_xy[bin_x, bin_y] \leftarrow P_xy[bin_x, bin_y] + 1$
end for
$P_x \leftarrow P_x/N$ ▷ normieren der Histogramme
$P_y \leftarrow P_y/N$
$P_{xy} \leftarrow P_{yx}/N$
$I(X;Y) \leftarrow 0$ ▷ berechnen der Mutual Information
for $i = 1, ..., k_x$ **do** ▷ für alle Bins
  for $i = 1, ..., k_x$ **do**
    $I(X;Y) \leftarrow I(X;Y) + P_{xy}[i,j] \; log\,(P_{xy}[i,j]/(P_x[i] \cdot P_y[j]))$
  end for
end for
return $I(X;Y)$

3.5 Zusammenfassung der Grundlagen

Die *biologische Evolution* beschreibt die Veränderung von Merkmalen von Lebewesen über mehrere Generationen. Diese entsteht durch die natürliche Selektion. Individuen treten in einen Konkurrenzkampf zueinander. Individuen mit unvorteilhaften Merkmalen gehen in diesem Konkurrenzkampf unter und sterben letztendlich aus, während Individuen mit vorteilhaften Merkmalen bestehen und Nachkommen erzeugen, wobei sie ihren Nachkommen, zumindest teilweise, ihre vorteilhaften Merkmale vererben. Dies wiederum geschieht durch *Mutation*, die zufällige Veränderung eines oder einiger Merkmale, und durch *Rekombination*, die Kombination der Merkmale mehrerer Elternteile.

Evolutionäre Algorithmen sind eine Klasse von Optimierungsverfahren, welche sich an der biologischen Evolution orientieren. Sie erzeugen dafür mehrere, anfangs zufällige Lösungen, hier Individuen genannt, bewerten deren *Fitness*, also deren Tauglichkeit als Lösung, und vermehren bevorzugt die Individuen mit der höchsten Fitness, wobei wiederum Mutation und Rekombination zum Einsatz kommen. Die Individuen mit einer niedrigen Fitness werden durch diese Nachkommen verdrängt. Dieser Vorgang wird bis zu einem vorher definierten Abbruchkriterium wiederholt.
Dabei bietet sich die $\mu + \lambda$ - Strategie an, wobei μ die Größe der Gesamtpopulation und λ die Anzahl der Nachkommen in jeder Generation angibt. Die μ überlebenden Individuen der Folgegeneration werden dabei sowohl aus der Menge der Individuen der letzten Generation als auch ihrer Kinderindividuen ausgewählt.
Als Selektionsstrategie, der Strategie, welche bestimmt, welche Individuen Nachkommen erzeugen dürfen und welche aussterben, kann beispielsweise die lineare Rangauswahl verwendet werden. Dabei werden zuerst alle Individuen nach ihrem Rang, also absteigend nach ihrer Fitness sortiert. Die Wahrscheinlichkeit, dass das Individuum mit Rang i Nachkommen produzieren darf, errechnet sich dann durch:

$$p(i) = \frac{\lambda - i + 1}{\sum_{j=1}^{\lambda} (\lambda - i + 1)}$$

Die Erzeugung von Nachkommen geschieht dabei entweder durch Mutation oder Rekombination. Die Mutation ist die zufällige Veränderung einzelner Merkmale, was durch die Addition eines gaußverteilten Zufallsvektors mit dem Erwartungswert 0 realisiert wird. Die Varianz dieses Zufallsvektors kann dabei durch die lokale Varianzanpassung bestimmt werden, welche die Varianz jeweils für jedes Gen einzeln mit in das Chromosom aufnimmt und durch die Evolution optimiert. Die Mutation dieser Varianzen σ_i^2 geschieht nach Nissen durch:

$$\sigma_i^{neu} = \sigma_i^{alt} \cdot exp\,(\,r_1 \cdot N(0,1) + r_2 \cdot N_i(0,1)\,)$$

Wobei $N(0,1)$ eine gaußverteilte Zufallszahl ist, welche einmal pro Chromosom ausgewürfelt wird, während $N_i(0,1)$ für jedes Gen i extra bestimmt wird. r_1 und r_2 sind Parameter, welche von Nissen mit $r_1 = 0,1$ und $r_2 = 0,2$ empfohlen werden.
Bei der Rekombination werden die Merkmale zweier oder mehrerer Individuen miteinander kombiniert. Dies geschieht zum Beispiel beim Uniform-Crossover durch das Auswürfeln eines Zufallsvektors, welcher, mit Wahrscheinlichkeiten, welche gewöhnlich zwischen $0,5$ und $0,8$ liegen, für jedes Gen einzeln bestimmt, ob bei den Nachkommen ein Austausch der Merkmale mit dem jeweils anderen Individuum stattfindet oder nicht.

Beim Levenberg-Marquardt Lernverfahren werden die Gewichte in einer Iteration durch:

$$w_{new} = w_{old} - \left(J^T J + \lambda I\right)^{-1} J^T \,\epsilon\,(w_{old})$$

aktualisiert. Wobei J die Jacobi-Matrix darstellt, welche durch Backpropagation of Error ermittelt werden kann. I ist die Einheitsmatrix und ϵ ist der einfache Fehler des Netzoutputs $y - t$. λ stellt einen Parameter dar, welcher beliebig initialisiert werden kann. Verschlechtert sich nach dem Gewichtsupdate der Fehler, sollte λ um einen Faktor vergrößert und die neuen Gewichte verworfen werden. Verbessert er sich, wird λ um einen Faktor verringert.

Die Mutual Information ist ein Maß für die Abhängigkeit von zwei Zufallsvariablen voneinander und kann über:

$$I(X;Y) = \int_x \int_y p(x,y)\, log \frac{p(x,y)}{p(x)\, p(y)}\, dx\, dy$$

berechnet werden, wobei p die entsprechenden Wahrscheinlichkeitsdichteverteilungen kennzeichnet. Diese können über Histogramme aus den Daten abgeschätzt werden, wobei sich die Binbreite W über die Regel nach Scott:

$$W = \frac{3,49 \cdot \sigma}{\sqrt[3]{N}}$$

berechnen lässt. σ gibt dabei die Standardabweichung der Datenpunkte und N deren Anzahl an.

Kapitel 4

Methoden zur Problemzerlegung

Dieses Kapitel stellt die während dieser Masterarbeit recherchierten und entworfenen Verfahren vor. Dafür ist das Kapitel in vier Teile untergliedert. Zu Beginn werden mehrere Möglichkeiten der Fitnessberechnung aufgezeigt, die für die evolutionäre Problemzerlegung notwendig ist. Danach werden die Bayes'sche Regularisierung und das Optimal Brain Surgeon Verfahren vorgestellt, welche ein Network Pruning durchführen und damit die optimale Größe und Anzahl der Gewichte für die verwendeten neuronalen Netze ermitteln sollen. Diese werden von allen problemzerlegenden Verfahren gleichermaßen angewendet. Die drei grundlegenden Verfahren zur Problemzerlegung sind im dritten Teil zu finden, während gegen Ende des Kapitels verschiedene Erweiterungen für diese vorgestellt werden, welche zum Beispiel eine Time Delay Estimation, die automatische Ermittlung der Anzahl der Teilprobleme oder eine nicht exklusive Problemdekomposition ermöglichen.

4.1 Fitnessberechnung

Da die Problemzerlegung in dieser Arbeit evolutionär vonstatten gehen soll, ist es notwendig, eine geeignete Fitnessfunktion zu finden, welche es ermöglicht, jedes Individuum auf seine Eignung als Lösung zu prüfen, also für jede potenzielle Problemzerlegung zu ermitteln, wie tauglich diese Problemzerlegung ist, in wie viele Teilprobleme im Vergleich zur tatsächlichen Anzahl zerlegt wurde und ob eventuell

noch Abhängigkeiten zwischen Größen verschiedener Teilprobleme bestehen.

4.1.1 Fitnessberechnung über die Zugehörigkeit der Problemgrößen

Vorgreifend sei an dieser Stelle erwähnt, dass zwei der drei Verfahren zur Problemzerlegung diese im Sinne einer Partitionierung der Problemgrößen bewerkstelligen. Die erste und wahrscheinlich intuitivste Möglichkeit zur Bewertung der Partitionierung wäre deshalb nach dem Schema: „Wie gut entspricht die Partitionierung den realen Teilproblemen?“ Um diese Frage zu beantworten, kann man den Prozentsatz der Problemgrößen verwenden, welche zur korrekten Partition und damit zum korrekten Teilproblem zugeordnet wurden. Da es natürlich vorkommen kann, dass einer Partition Problemgrößen mehrerer Teilprobleme angehören und damit nicht immer klar ist, welchem Teilproblem die aktuelle Partition entspricht, soll die Abbildung von Partitionen zu Teilproblemen verwendet werden, welche den maximalen Prozentsatz und damit die maximale Fitness ergibt.
In Abbildung 4.1 wird dies anhand des Beispielproblems erläutert. Links ist das zu bewertende Individuum in Form einer Partitionierung der Problemgrößen zu sehen. Im rechten Bild ist zum Vergleich die korrekte Zerlegung in Teilprobleme dargestellt. Beide Partitionen des Individuums enthalten Größen aus beiden Teilproblemen. Nimmt man nun an, dass die obere Partition dem Teilproblem 1 entspricht und die untere Teilproblem 2, wurden 3 der 5 Größen, also 60% korrekt zugeordnet. Nimmt man umgekehrt an, dass die obere Partition Teilproblem 2 und die untere Teilproblem 1 entspricht, wurden nur 2 der 5 Problemgrößen, nämlich 40% richtig zugeordnet. Die Fitness des Individuums entspricht dem Maximum der beiden Werte.
Formel 4.1 drückt die Fitnessberechnung als mathematische Formel aus:

$$F_{PG} = \max \left(\frac{\sum_{i=1}^{N_P} |P_{Ind_\sigma(i)} \cap P_{true_i}|}{|M|} \;\middle|\; \sigma : \{1, .., N_P\} \rightarrow \{1, ..., N_P\} \text{Permutationsfunktion} \right) \quad (4.1)$$

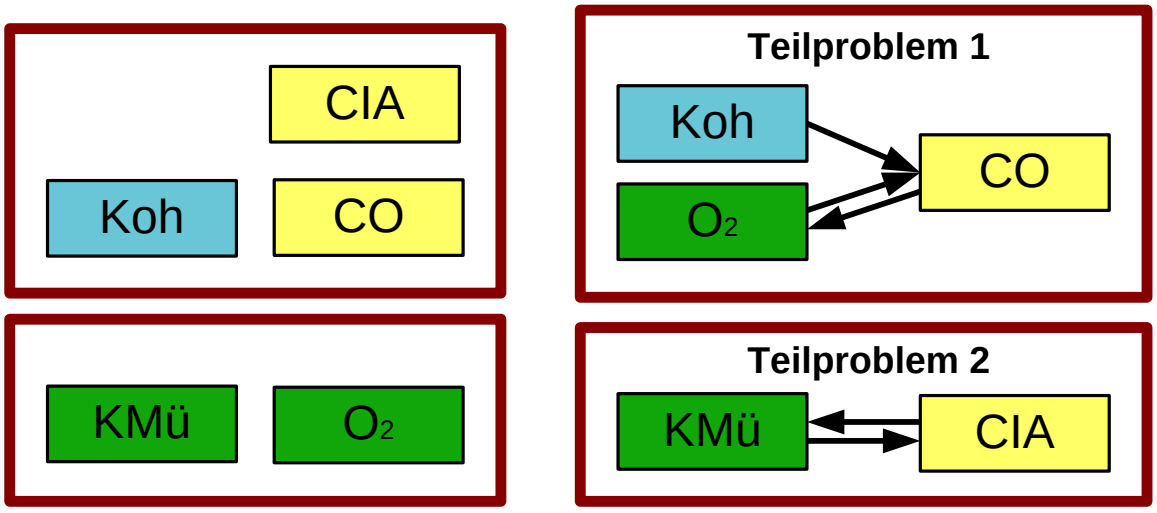

Abbildung 4.1: Veranschaulichung der Fitnessberechnung über die Zugehörigkeit der Problemgrößen anhand des Beispielproblems. Rechts ist die korrekte Zerlegung dargestellt, während das linke Bild die zu bewertende Partitionierung zeigt. Wird die obere Partition als Teilproblem 1 und die untere als Teilproblem 2 angesehen, ergibt sich ein Prozentsatz von 60% korrekt zugeordneter Problemgrößen, im umgekehrten Fall ein Prozentsatz von 40%. Die Problemzerlegung besitzt deshalb eine Fitness von 60%.

Wobei P_{Ind_i} die i. Partition der zu prüfenden Partitionierung und entsprechend P_{true_i} die i. Partition der Partitionierung, welche den realen Teilproblemen entspricht, darstellen. Außerdem gibt M die Menge der Problemgrößen und N_p die Anzahl der Partitionen an.

Diese Form der Fitnessberechnung hat jedoch auch Nachteile. So wird zum Beispiel ausschließlich danach geschaut, ob eine Problemgröße in der korrekten Partition liegt oder nicht. Es ist aber durchaus auch von Bedeutung, ob Problemgrößen, welche ursprünglich alle zu einem Teilproblem gehörten, aber nicht in der zugehörigen Partition liegen, alle in einer Partition liegen, womit zumindest einige der Abhängigkeiten erhalten bleiben würden, oder ob sie über mehrere andere Partitionen verteilt sind. Weiterhin wird die Menge der Abhängigkeiten einer Größe nicht beachtet. Eine Problemgröße

mit sehr vielen Abhängigkeiten, welche auf viele andere Größen Einfluss nimmt oder auf welche umgekehrt viele Größen Einfluss besitzen und die einem falschen Teilproblem bzw. einer falschen Partition zugeordnet wurde, ist kritischer als eine mit einer geringen Anzahl eingehender oder ausgehender Abhängigkeiten. Auch ist es so nicht möglich, Partitionierungen mit einer höheren oder niedrigeren Anzahl an Partitionen als real vorhandene Teilprobleme zu bewerten, die Anzahl der Partitionen muss für diese Art der Fitnessberechnung der Anzahl der vorhandenen Teilprobleme entsprechen. Am schwersten wiegt allerdings, dass die korrekte Problemzerlegung für diese Methode bekannt sein muss, weshalb sie höchstens für simulierte Datensätze, z.B. zum Test der Verfahren im späteren Teil der Arbeit, verwendet werden kann, nicht aber in der Praxis, wo die korrekte Problemzerlegung erst ermittelt werden soll.

4.1.2 Fitnessberechnung über die Abhängigkeiten

Um zumindest einige der Nachteile der Fitnessberechnung über die Zugehörigkeit der Problemgrößen zu den Partitionen zu umgehen, kann man anstelle der Problemgrößen die Abhängigkeiten verwenden. Nach der Definition der Problemzerlegung in dieser Arbeit dürfen bei einer korrekten Problemzerlegung keine Abhängigkeiten zwischen Problemgrößen verschiedener Teilprobleme bestehen. Eine Partitionierung also, bei der viele Abhängigkeiten zwischen Größen mehrerer Partitionen bestehen, kann somit keine brauchbare Problemzerlegung darstellen. Bei einer Approximation durch neuronale Netze beispielsweise würden durch diese Abhängigkeiten Informationen fehlen. Deshalb wird hier für die Fitnessberechnung der Prozentsatz der korrekten Abhängigkeiten berechnet, wobei eine Abhängigkeit als „korrekt“ angesehen wird, wenn sie zwischen zwei Problemgrößen der selben Partition, und umgekehrt als „nicht korrekt“, wenn sie zwischen Problemgrößen verschiedener Partitionen besteht.

Abbildung 4.2 veranschaulicht die Fitnessberechnung über die Abhängigkeiten anhand des Beispielproblems. Im rechten Bild ist die korrekte Zerlegung abgebildet, während links die zu bewertende Partitionierung zu sehen ist. Nur eine einzige der fünf Abhängigkeiten befindet sich innerhalb einer der Partitionen. Die Fitness des Individuums

beträgt daher nur 20%. Der große Unterschied zum Wert der Fitness, welcher über die Zugehörigkeit der Problemgrößen berechnet wurde, verdeutlicht noch einmal dessen Nachteile, da dort nur die Problemgrößen selbst, nicht aber die eigentlich wichtigen Abhängigkeiten betrachtet werden.

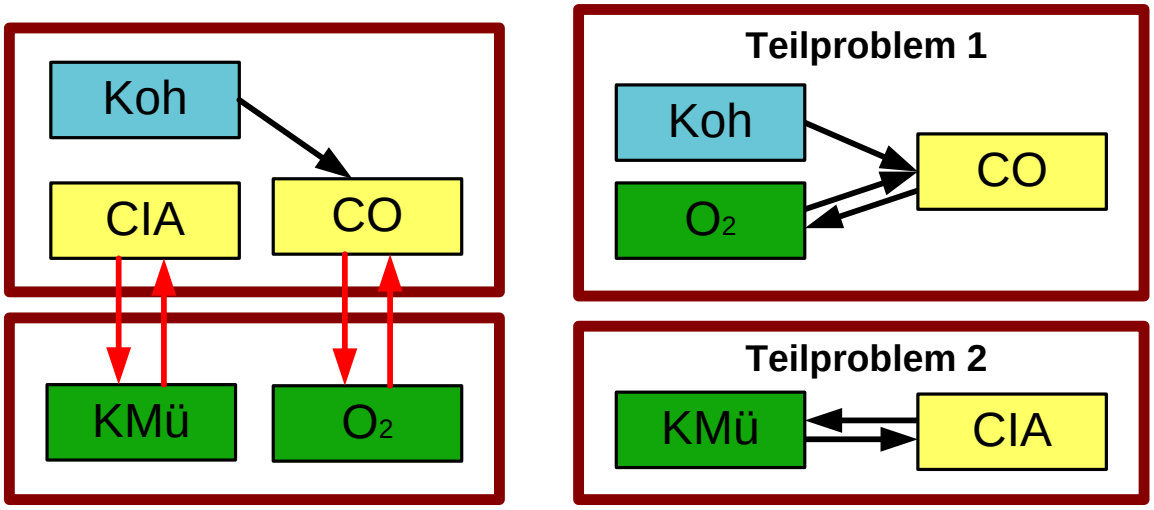

Abbildung 4.2: Veranschaulichung der Fitnessberechnung über die Abhängigkeiten anhand des Beispielproblems. Rechts ist die korrekte Zerlegung dargestellt, während das linke Bild die zu bewertende Partitionierung zeigt. Nur eine der fünf Abhängigkeiten besteht zwischen Problemgrößen der selben Partition. Die Fitness des Individuums beträgt daher 20%.

In Formel 4.2 ist die Fitnessberechnung über die Abhängigkeiten mathematisch ausgedrückt:

$$F_{Abh} = \frac{C}{|A|} \tag{4.2}$$

Mit:

$$C = \{(a_1, a_2) \in A \mid \exists j \in \{1, .., N_P\} : \exists (p_1, p_2) \in P_j \times P_j : p_1 = a_1, p_2 = a_2\}$$

Dabei ist A die Menge der Abhängigkeiten und besteht aus den Tupeln zweier Problem-

größen (a_1, a_2), bei welchen eine Abhängigkeit von Problemgröße a_1 zu a_2 vorhanden ist. Weiterhin stellt P_j die Partition j dar und enthält alle Problemgrößen, welche dieser Partition zugeordnet wurden. Schließlich gibt N_P die Anzahl der Partitionen an.

Vor allem in Hinblick auf die Problemzerlegung stellt die Fitnessberechnung über die Abhängigkeiten die bessere Alternative im Vergleich zur Fitnessberechnung über die Zugehörigkeit der Problemgrößen dar, da hier die vorhandenen oder nicht vorhandenen Informationen in Form der Abhängigkeiten betrachtet werden. Sie spiegelt somit wieder, wie viele Informationen, beispielsweise für die Approximation der Problemgrößen durch ein neuronales Netz, bei der Problemzerlegung verloren gehen würden.
Nachteilig ist allerdings, dass weiterhin zur Berechnung der Fitness die korrekte Partitionierung und mehr noch, auch die tatsächlich vorhandenen Abhängigkeiten der Problemgrößen zueinander komplett bekannt sein müssen, was, vor allem da offenbar mit Hilfe der in dieser Arbeit vorgestellten Verfahren erst eine Problemzerlegung stattfinden soll, in der Praxis höchst unwahrscheinlich ist. Dieses Fitnessmaß ist daher hier nur für die später in dieser Arbeit durchgeführten Tests der problemzerlegenden Verfahren von Bedeutung. Ein weiterer Nachteil ist außerdem, dass die Anzahl der Teilprobleme nicht in Betracht gezogen wird. Eine Zerlegung in zu viele Partitionen schlägt sich zwar, durch die dadurch verursachten nicht korrekten Abhängigkeiten, negativ auf die Fitness nieder, eine zu geringe Anzahl Partitionen hingegen hat keinen Einfluss auf die Fitness. So besitzt im Extremfall beispielsweise eine einzige Partition mit allen Problemgrößen, wenn also gar keine Problemzerlegung stattfand, immer die maximale Fitness.

4.1.3 Fitnessberechnung über die Fehlerrate

Für die Anwendung der Verfahren in der Praxis muss für die Evolution eine Fitnessfunktion gefunden werden, bei welcher die reale Problemzerlegung, anders als bei den beiden bisher vorgestellten Fitnessberechnungen, nicht bekannt sein muss, da diese logischerweise erst durch das Verfahren selbst ermittelt werden soll. Dafür bietet sich die Fehlerrate der neuronalen Netze an, welche die Problemgrößen approximieren sollen.

Die Wirkungsweise ist hier ähnlich wie bei der Fitnessberechnung über die Abhängigkeiten, denn sind nicht korrekte Abhängigkeiten vorhanden, bestehen also Abhängigkeiten zwischen Problemgrößen verschiedener Partitionen und damit Teilprobleme, fehlen Informationen für die Approximation eines oder auch mehrerer der neuronalen Netze und die Fitness sinkt entsprechend. Je mehr dieser Abhängigkeiten nicht korrekt sind, desto schwieriger ist es für die neuronalen Netze, die Problemgrößen zu approximieren und desto geringer ist auch die Fitness.
In Abbildung 4.3 wird dies anhand einer Beispielpartitionierung für das Beispielproblem verdeutlicht. Die zu bewertende Partition ist oben links zu sehen. Unten sind die entsprechend der Partition getrennten neuronalen Netze mitsamt ihren Inputgrößen dargestellt. Zur Approximation der Mess- und Regelgrößen fehlen den neuronalen Netzen allerdings Informationen, da diese Größen dem jeweils anderen Netz zugeordnet wurden. Umgekehrt besitzen sie einige Inputgrößen, mit welchen sie nichts anfangen können, die jedoch die Dimensionalität des Inputraumes erhöhen. Beides wirkt sich negativ auf die Fitness aus.
Auch die Anzahl der Partitionen wird hier in Betracht gezogen. Ist die Anzahl der Partitionen größer als die der real vorhandenen Teilprobleme, so erhöht sich damit zwangsweise die Anzahl der nicht korrekten Abhängigkeiten und die Fitness sinkt. Ist die Anzahl der Partitionen dagegen zu gering, wächst die Komplexität für die neuronalen Netze, sie haben mehr Inputs und mehr freie Parameter in Form der Gewichte. Die Approximation wird dadurch schwieriger und es steht zu erwarten, dass das Training mit einer höheren Fehlerrate abgeschlossen wird.

Als Fehlerrate ist nahezu jedes denkbare Fehlermaß geeignet. Hier wurde, um den Fehler auf der gleichen Skala mit den Teachwerten vergleichen zu können, der Root Mean Squared Error (RMSE) verwendet:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y(x_i) - t_i)^2} \tag{4.3}$$

Dabei ist N die Anzahl der Trainingspattern, $y(x_i)$ die Ausgabe des neuronalen Netzes

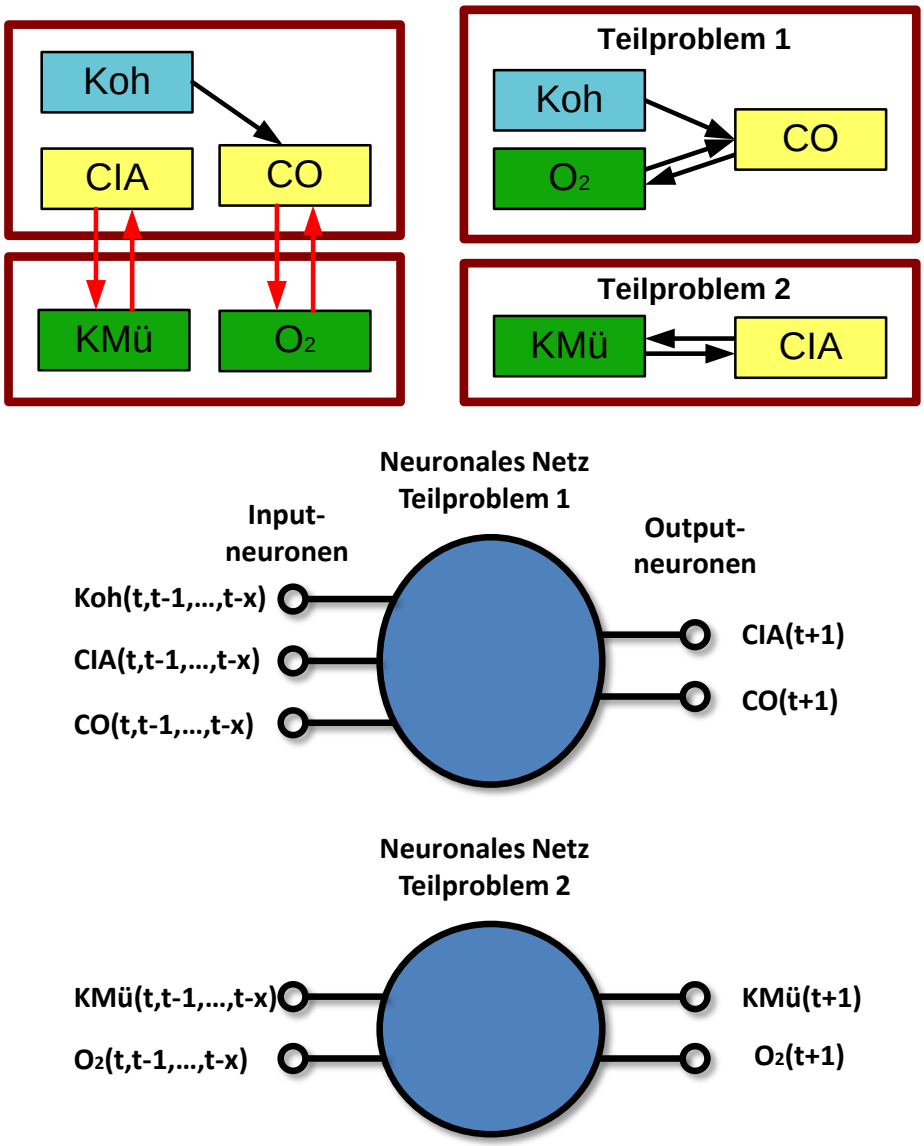

Abbildung 4.3: Veranschaulichung der Fitnessberechnung über die Fehlerrate anhand des Beispielproblems. Das obere linke Bild zeigt die zu bewertende Partitionierung. Unten sind die entsprechend zum Training verwendeten neuronalen Netze zu sehen. Wie in der korrekten Problemzerlegung oben rechts zu sehen ist, fehlen den neuronalen Netzen mehrere zur Approximation benötigte Problemgrößen, beispielsweise die Werte der Luftzufuhr für die Berechnung des Kohlenmonoxidgehaltes. Dies schlägt sich negativ auf die Approximationsgüte und damit die Fitness nieder.

für den aktuellen Vektor der Eingangsgrößen x_i und t_i das Trainingspattern i.
Wird ein neuronales Netz dagegen auf mehr als eine Problemgröße trainiert, muss die Berechnung des Root Mean Sqared Errors entsprechend angepasst werden. Dazu werden die Trainingspattern der Ausgänge einfach aneinander gehängt und bei der Fehlerberechnung als ein einziger Output angesehen:

$$RMSE = \sqrt{\frac{1}{N \cdot N_y} \sum_{i=1}^{N} \sum_{j=1}^{N_y} (y_j(x_i) - t_{ij})^2} \tag{4.4}$$

Dabei stellt N_y die Anzahl der Outputneuronen dar. Außerdem ist $y_j(x_i)$ die Ausgabe des j. Outputneurons und t_{ij} das i. Trainingspattern für Outputneuron j.
Anzumerken ist hierbei allerdings, dass, anders als bisher, eine hohe Fitness, also eine hohe Fehlerrate, in diesem Fall ein schlechtes Individuum bedeutet. Der Begriff „Fitness“ ist hier also eher als „Weakness“ anzusehen. Die Mechanismen der Evolution lassen sich jedoch ohne Weiteres an die umgekehrten Verhältnisse anpassen.

Nachteilig bei dieser Fitnessberechnung ist der Zufallsfaktor, welcher durch das Training der neuronalen Netze verursacht wird. So können zwei eigentlich gleiche Individuen, verursacht durch anders initialisierte Gewichte oder unterschiedliche Trainingsverfahren, zwei unterschiedliche Fitnesswerte besitzen. Auch besteht hier weiterhin das Problem, dass eine zu geringe Partitionszahl bzw. zu geringe Anzahl Teilprobleme zwar in der Theorie durch die höhere Komplexität eine geringere Fitness besitzt, dieser Unterschied in der Praxis aber eher gering ausfällt, da alle bzw. die meisten Abhängigkeiten korrekt sind und den neuronalen Netzen keine Informationen fehlen.

4.2 Network Pruning

In Kapitel 2.4 wurde bei der Herausstellung der Vorteile einer Problemdekomposition unter Anderem die geringere Anzahl der benötigten Gewichte und die nicht zuletzt auch damit verbundene Verkürzung der Lernzeit und das Finden besserer Lösungen für das Problem beschrieben. Ist das neuronale Netz jedoch von vornherein überdi-

mensioniert, werden viele dieser Vorteile wieder zunichte gemacht. Um von den bei der Problemzerlegung eingesparten Gewichten optimal profitieren zu können, sind bereits möglichst kleine neuronale Netze mit einer möglichst geringen Anzahl von Neuronen und Gewichten notwendig, da sonst die gewonnenen Vorteile von der überschüssigen Kapazität des neuronalen Netzes wieder aufgehoben werden.
Wenn man zum Beispiel annimmt, dass sich das Kraftwerks-Beispielproblem aus Kapitel 2.2, so, wie in Abbildung 4.4 links zu sehen, mit vier vollvernetzten Hiddenneuronen und die beiden Teilprobleme mit jeweils zwei Hiddenneuronen approximieren lassen, reduziert sich die Anzahl der Gewichte nach dieser Problemdekomposition von 36 auf 18. Bei einer weniger sorgfältigen Wahl der Größe des neuronalen Netzes, beispielsweise mit zehn vollvernetzten Hiddenneuronen, bzw. zwei Mal fünf im zerlegten Fall, beträgt die Anzahl der Gewichte 90 bzw. 45. Zwar findet auch hier wieder eine Reduktion um 50% statt, jedoch bleibt das Netz weit hinter seinen Möglichkeiten zurück, da die Anzahl der Gewichte im zerlegten Fall hier immer noch höher ist, als mit einem sorgfältiger ausgewählten Netz im unzerlegten Fall.

Um dies zu vermeiden, sollen die zur Problemdekomposition verwendeten neuronalen Netze vor dem Training einem sogenannten Network Pruning unterworfen werden. Ziel dieses Network Prunings ist es, die Struktur des neuronalen Netzwerkes automatisch an das Problem anzupassen. Dafür wird, wie der Name schon sagt (engl. „to prune“: zurechtstutzen, beschneiden) mit einem zu großen neuronalen Netz begonnen und dieses, Schritt für Schritt, durch das Entfernen von als irrelevant klassifizierten Gewichten, verkleinert, bis ein neuronales Netzwerk möglichst geringer Größe erreicht wird.

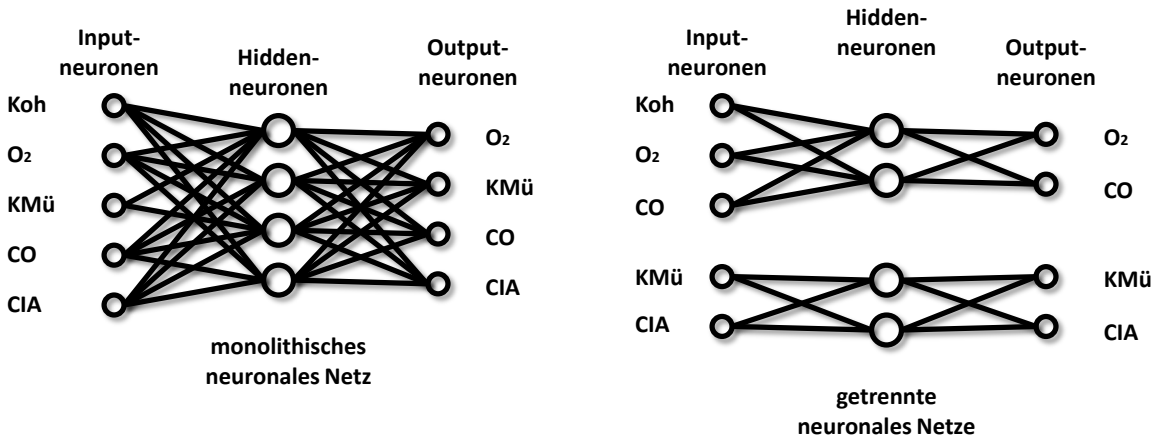

Abbildung 4.4: Veranschaulichung der Reduktion der Anzahl der Gewichte bei einer Problemdekomposition.

4.2.1 Bayes'sche Regularisierung

Die Bayes'sche Regularisierung wird unter anderem in [MacKay, 1991] und [Foresee und Hagan, 1997] beschrieben und ist eine Erweiterung des Lernverfahrens, beispielsweise des Levenberg-Marquardt Verfahrens aus Kapitel 3.3.

Das Grundanliegen dabei ist es, möglichst kleine Gewichte zu bevorzugen, also beim Lernvorgang nicht nur den quadratischen Fehler, sondern auch die Summe der Gewichte zu minimieren. Dies hat zwei Effekte: Erstens wird die Ausgabefunktion auf diese Weise glatter, da so eine geringe Veränderung des Inputs nicht zu einer großen Veränderung des Outputs führen kann, was in Abbildung 4.5 verdeutlicht wird. Gezeigt werden drei Fermifunktionen mit den Gewichten $w = 10$ (blau), $w = 1$ (schwarz) und $w = 0,1$ (rot). Während die rote Funktion mit dem Gewicht von $0,1$ sehr glatt, fast schon eine Gerade ist, ist die blaue Fermifunktion mit $w = 10$ relativ steil und kantig. Eine glatte Ausgabefunktion aber verbessert die Generalisierung des neuronalen Netzes.

Der zweite Effekt ist, dass, durch die Minimierung der Gewichte, für das Problem irrelevante Gewichte gegen Null gedrückt werden, während relevante Gewichte durch die Minimierung relativ unbeeinflusst bleiben und einen Wert deutlich über Null

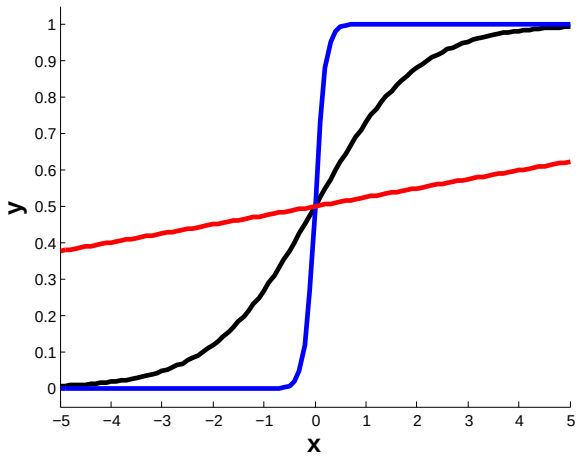

Abbildung 4.5: Veranschaulichung der Glattheit der Ausgabefunktion bei unterschiedlich großen Gewichten. Gezeigt werden Fermifunktionen $y = 1\,/\,(1 + exp(-w \cdot x)\,)$ mit jeweils einem Gewicht: $w = 10$ (blau), $w = 1$ (schwarz) und $w = 0,1$ (rot). Je höher das Gewicht, desto steiler und kantiger ist die Ausgabefunktion.

besitzen. Auf diese Weise wäre eine erste, einfache Version des Network Prunings möglich, indem nach dem Training die Gewichte, welche einen Wert von Null oder nahe Null besitzen, gelöscht werden. Die übrig bleibenden Gewichte wären dann allesamt für die Lösung des Problems relevant und das so entstehende Netz kann, zumindest theoretisch, als minimal angesehen werden.

Die Bayes'sche Regularisierung integriert die Minimierung der Gewichte in die Fehlerfunktion, sodass mit Hilfe des Lernverfahrens nicht nur die Summe der quadratischen Fehler (Sum of Squared Errors, SSE), sondern automatisch auch gleich die Summe der quadrierten Gewichte (Sum of Squared Weights, SSW) minimiert wird. Die so angepasste Fehlerfunktion setzt sich wie folgt zusammen:

$$E = \beta \cdot SSE + \alpha \cdot SSW \tag{4.5}$$

Dabei ist E die Fehlerfunktion und α und β sind Parameter, welche die Wichtungen der beiden Fehlerterme darstellen und im Folgenden noch erläutert werden. Weiterhin

gilt:

$$SSW = \sum_{j=1}^{N_w} w_j^2 \qquad \text{und} \qquad SSE = \sum_{i=1}^{N} \left(y(x_i) - t_i \right)^2 \tag{4.6}$$

Dabei ist N_w die Anzahl der Gewichte und N beschreibt die Anzahl der Datenpunkte.

Zum tieferen Verständnis der Funktionsweise des Algorithmus und der Wirkung der Parameter α und β ist es notwendig, diese Fehlerfunktion im Folgenden einmal genauer herzuleiten.

Herleitung der Fehlerfunktion

In einem Bayes'schen Framework werden, anstelle Fehlerfunktionen zu minimieren, für das Training neuronaler Netze Wahrscheinlichkeiten maximiert. Mit einem gegebenen Trainingsdatensatz D, einem Netzwerkmodell M, bestehend aus einem neuronalen Netz mit seinen Neuronen und Verbindungen, und den, zumindest vorerst als gegeben angenommenen Parametern α und β, lässt sich die Wahrscheinlichkeit $P(w \,|\, D, \alpha, \beta, M)$ der Gewichte über das Bayestheorem ermitteln:

$$P(w \,|\, D, \alpha, \beta, M) = \frac{P(D \,|\, w, \beta, M)\, P(w \,|\, \alpha, M)}{P(D \,|\, \alpha, \beta, M)} \tag{4.7}$$

Es wird also nach den Gewichten gesucht, welche bei dem gegebenen Netzwerkmodell, α und β und den Daten am wahrscheinlichsten sind. Durch die Maximierung dieser Wahrscheinlichkeit $P(w \,|\, D, \alpha, \beta, M)$ lassen sich die für das zugrunde liegende Problem optimalen Gewichte ermitteln. Formel 4.7 setzt sich wiederum aus drei Wahrscheinlichkeiten zusammen, welche im Folgenden einzeln untersucht werden.

Die erste Wahrscheinlichkeit, welche in Formel 4.7 enthalten ist, ist das sogenannte Datenmodell $P(D \,|\, w, \beta, M)$. Es beschreibt die Wahrscheinlichkeit der Daten für ein gegebenes neuronales Netz mitsamt Gewichten und dem Parameter β. Es beantwortet also die Frage: „Wie gut passen die Daten zu meinem trainierten neuronalen Netz?“ Je höher diese Wahrscheinlichkeit für die vorliegenden Daten ist, desto besser appro-

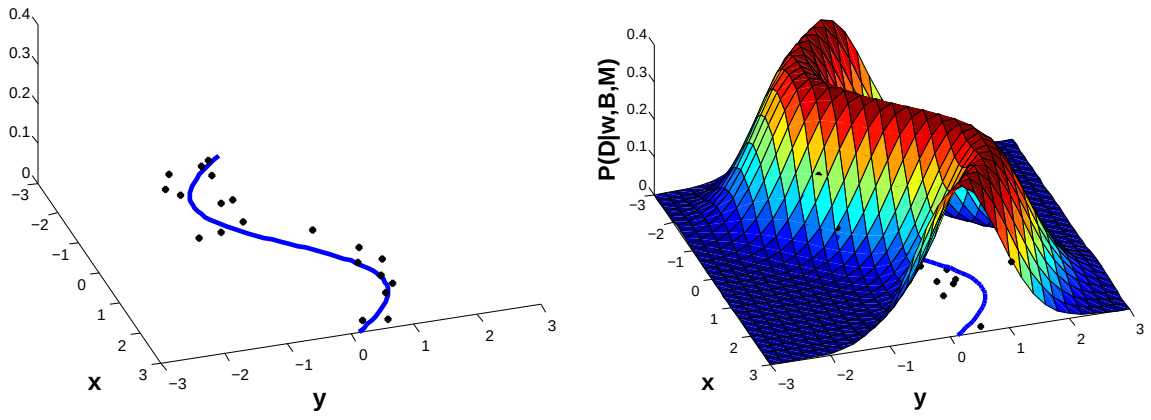

Abbildung 4.6: Illustration des Datenmodells der Bayes'schen Regularisierung. Dargestellt sind die Ausgabefunktion des gegebenen trainierten neuronalen Netzes (blau) und die Datenpunkte (schwarz). Das rechte Bild zeigt zusätzlich die Wahrscheinlichkeitsfunktion, welche sich aus einer über die Ausgabefunktion des Netzes gelegten Gaußkurve ergibt und die Stärke des Rauschens auf den Daten widerspiegelt. Der Wert der Wahrscheinlichkeitsfunktion an den Stellen der Datenpunkte beschreibt die Passgenauigkeit der Daten zu dem neuronalen Netz.

ximiert das Netz diese Daten.

Abbildung 4.6 zeigt dies grafisch. Links sind dort die Ausgabefunktion des trainierten neuronalen Netzes (blau) und die Datenpunkte (schwarz) zu sehen. Es wird in der Bayes'schen Regularisierung außerdem angenommen, dass auf den Daten ein gaußförmiges Rauschen liegt. Dies ist im rechten Bild dargestellt. Es wurde hier eine gaußförmige Kurve über die Ausgabefunktion des neuronalen Netzes gelegt, welche die Wahrscheinlichkeit angibt, dass die Datenpunkte an dieser Stelle, unter Berücksichtigung des Rauschens, mit der Ausgabefunktion des Netzes übereinstimmen. Liegen die Datenpunkte nahe an der Ausgabefunktion, ergibt sich eine hohe Wahrscheinlichkeit, also eine gute Passgenauigkeit der Daten zu dem gegebenen neuronalen Netz. Liegen sie hingegen weiter entfernt, sinkt die Wahrscheinlichkeit entsprechend ab. Die Stärke des Absinkens wird durch die Breite der Gaußfunktion geregelt. Dies liefert nun auch eine Interpretation des Parameters β, denn dieser ergibt sich zu:

$$\beta = \frac{1}{\sigma_D^2} \tag{4.8}$$

Dabei gibt σ_D^2 die Standardabweichung des Rauschens auf den Daten an. β ist also ein Maß für die Höhe des Rauschens auf den Daten und steuert die Breite der Gaußfunktion und damit das Absinken der Wahrscheinlichkeit mit zunehmender Entfernung zur Ausgabefunktion des neuronalen Netzes. Ist β groß, also die Gaußfunktion sehr schmal, wird ein geringes Rauschen auf den Daten angenommen und Ausreißer, welche weit von der Ausgabefunktion entfernt liegen, erhalten eine sehr geringe Wahrscheinlichkeit. Ist β dagegen sehr klein und die Gaußfunktion damit sehr breit, werden auch weit entfernte Datenpunkte toleriert.

Dies lässt sich auch als Formel darstellen. Es wird eine Gaußfunktion mit Standardabweichung σ_D über die Ausgabefunktion $y(x)$ gelegt, wobei x den Vektor der Inputs darstellt, und die Datenpunkte t_i werden entsprechend dieser gewichtet:

$$P(D \,|\, w, \beta, M) = \frac{1}{\sqrt{(2\pi\,\sigma_D^2)^N}}\, exp\left(-\frac{\sum\left(y(x_i) - t_i\right)^2}{\sigma_D^2}\right) \tag{4.9}$$

Mit Hilfe der Definition der Summe der quadratischen Fehler (SSE) aus Formel 4.6 und des Parameters β aus Formel 4.8 ergibt sich:

$$P(D \,|\, w, \beta, M) = \frac{1}{\sqrt{(2\pi\,/\,\beta)^N}}\, exp\left(-\beta \cdot SSE\right) \tag{4.10}$$

Die zweite Wahrscheinlichkeit aus Formel 4.7 ist das Gewichtsmodell $P(w \,|\, \alpha, M)$. Es stellt die Wahrscheinlichkeit für die Gewichte bei einem gegebenen Parameter α und Netzwerkmodell M dar. Ziel der Bayes'schen Regularisierung ist es, wie schon erwähnt, die Gewichte möglichst klein zu halten, um eine möglichst glatte Ausgabefunktion zu erlangen. Dafür wird angenommen, dass die Gewichte normalverteilt sind, mit dem Mittelwert $\mu = 0$ und der Standardabweichung σ_w. Dies ist in Abbildung 4.7 grafisch dargestellt, welche die Wahrscheinlichkeitsfunktion für die Gewichte des Netzes

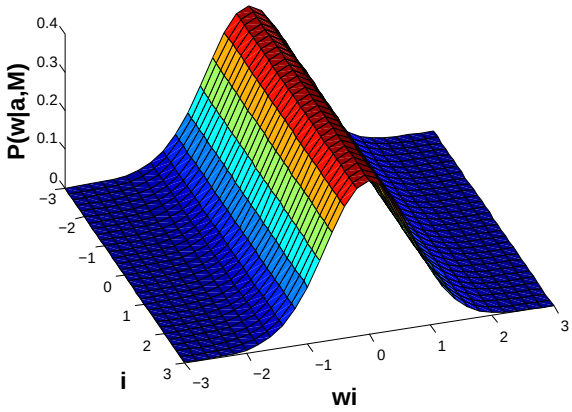

Abbildung 4.7: Illustration des Gewichtsmodells der Bayes'schen Regularisierung. Dargestellt ist die Wahrscheinlichkeitsfunktion der Werte der Gewichte des Netzes. Gewichte mit einem hohen Wert werden, abhängig von der Breite der Wahrscheinlichkeitsfunktion, als unwahrscheinlich erachtet und erhalten einen geringen Wahrscheinlichkeitswert.

zeigt. Gewichte mit einem relativ hohen Wert werden als unwahrscheinlich betrachtet und erhalten einen geringen Wahrscheinlichkeitswert. Wie stark der Abfall der Wahrscheinlichkeit für solche Gewichte mit hohen Werten ist, wird wieder über die Breite der Gaußfunktion geregelt. Es gilt:

$$\alpha = \frac{1}{\sigma_w^2} \tag{4.11}$$

α steuert somit die Breite der Gaußfunktion, welche das Absinken der Wahrscheinlichkeit für hohe Gewichte bestimmt. Ist α groß, also die Gaußfunktion sehr schmal, erhalten hohe Gewichte eine geringe Wahrscheinlichkeit. Die Bayes'sche Reglarisierung zwingt den Gewichten dadurch im Laufe des Trainings einen sehr geringen Wert auf. Ist α hingegen sehr klein und die Gaußfunktion somit eher breit, werden auch höhere Gewichte zugelassen.

Es wird also eine Gaußfunktion mit $\mu = 0$ und der Standardabweichung σ_w über die Werte der Gewichte gelegt und deren Wahrscheinlichkeit entsprechend bewertet. In

einer Formel ausgedrückt sieht dies wie folgt aus:

$$P(w \,|\, \alpha, M) = \frac{1}{\sqrt{(2\pi\,\sigma_w^2)^{N_w}}}\, exp\left(-\frac{\sum w_i^2}{\sigma_w^2}\right) \tag{4.12}$$

Mit Hilfe der Definition der Summe der quadrierten Gewichte (SSW) aus Formel 4.6 und des Parameters α aus Formel 4.11 ergibt sich:

$$P(w \,|\, \alpha, M) = \frac{1}{\sqrt{(2\pi \,/\, \alpha)^{N_w}}}\, exp\,(-\alpha \cdot SSW) \tag{4.13}$$

Die dritte und letzte Wahrscheinlichkeit aus Formel 4.7, nämlich $P(D \,|\, \alpha, \beta, M)$, ist von den Gewichten w unabhängig, also eine Konstante und somit für das Optimierungsproblem irrelevant. Sie kann im Folgenden ignoriert werden.

Nun lassen sich die Ergebnisse von Formel 4.10 und 4.13 in die Formel 4.7 zur Optimierung der Gewichte einsetzen. Es ergibt sich:

$$P(w \,|\, D, \alpha, \beta, M) = \frac{\frac{1}{\sqrt{(2\pi \,/\, \beta)^N}\sqrt{(2\pi \,/\, \alpha)^{N_w}}}\, exp\,(-\,(\beta \cdot SSE + \alpha \cdot SSW))}{C} \tag{4.14}$$

Dabei ist C eine Konstante. Der Term vor der Exponentialfunktion ist aber ebenfalls von den Gewichten unabhängig und somit ebenfalls konstant. Er lässt sich daher mit C zusammenfassen und wird Z_M genannt:

$$P(w \,|\, D, \alpha, \beta, M) = \frac{exp\,(-\,(\beta \cdot SSE + \alpha \cdot SSW))}{Z_M\,(\alpha, \beta)} \tag{4.15}$$

$Z_M(\alpha, \beta)$ ist, bezüglich der Gewichte w, konstant und wird im folgenden Abschnitt noch benötigt. Der übrige Term besteht nun nur noch aus einer Exponentialfunktion $exp(-X)$. Entsprechend der Charakteristik der Exponentialfunktion, welche monoton steigend ist, lässt sich $exp(-X)$ maximieren, indem man X minimiert. Das bisherige

Maximierungsproblem, welches die Wahrscheinlichkeit der Gewichte $P(w \mid D, \alpha, \beta, M)$ maximieren sollte, lässt sich somit in ein Minimierungsproblem transferieren, welches

$$E = \beta \cdot SSE + \alpha \cdot SSW \tag{4.16}$$

minimiert und der Fehlerfunktion für die Bayes'sche Regularisierung aus Formel 4.5 entspricht, die somit hergeleitet wurde. Noch offen ist die Wahl der Parameter α und β.

Wahl der Parameter α und β

Zusammenfassend aus dem letzten Kapitel lässt sich sagen, dass α und β das Vorwissen über die Daten und deren Verlauf sind. α ist dabei das Vorwissen über den Grad der Glattheit der zu approximierenden Funktion. Die Glattheit der Ausgabefunktion des neuronalen Netzes wird dafür über die zulässige Größe der Gewichte gesteuert. Bei einem großen α werden hohe Gewichte stark bestraft. Sie werden als unwahrscheinlich angesehen und, mithilfe der Zuweisung einer niedrigen Wahrscheinlichkeit, verworfen. Dies resultiert in einer sehr glatten Ausgabefunktion. Bei einem niedrigen α hingegen werden auch hohe Gewichte toleriert, was zu einem zunehmend zerklüfteten Verlauf der Ausgabefunktion führen kann. Es ist also wichtig, α entsprechend der Daten zu wählen. Abbildung 4.8 verdeutlicht diesen Umstand. Die gezeigten Grafiken wurden aus [MacKay, 1991] entnommen und mit RBF-Netzen erstellt. Oben links wurde dabei versucht, die Datenpunkte mit einem zu geringen α zu approximieren. Die Gewichte wurden deshalb zu klein gehalten, die Ausgabefunktion geriet zu glatt und sie konnte sich dadurch den Datenpunkten nicht ausreichend anpassen. Im oberen rechten Bild ist das Gegenteil der Fall. Die Gewichte wurden nicht ausreichend beschränkt. Die Ausgabefunktion geriet sehr steil und zerklüftet und leidet an Overfitting. Im unteren Bild wurde der richtige Kompromiss gefunden und ein gutes α gewählt.

Ähnliches gilt für den Parameter β, welcher das Vorwissen über die Stärke des Rauschens auf den Daten darstellt. Bei einem großen β wird ein geringes Rauschen

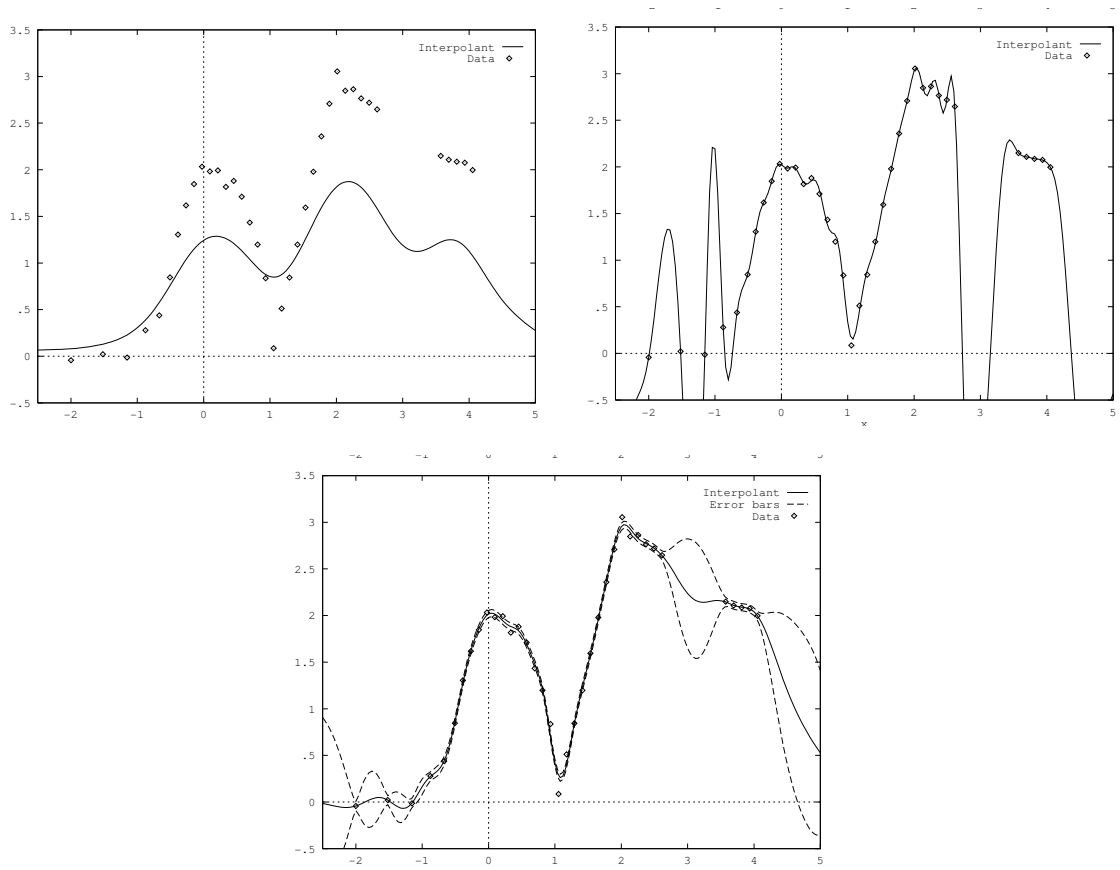

Abbildung 4.8: Illustration der Wahl von Parameter α bei der Bayes'schen Regularisierung. Das obere linke Bild zeigt ein zu großes α, die Ausgabefunktion wird dadurch zu glatt und kann sich den Daten nicht anpassen. Oben rechts ist eine Ausgabefunktion mit einem zu kleinen α zu sehen, was in starkem Overfitting resultiert. Das untere Bild zeigt die Ausgabefunktion mit optimalem α. Bildquelle: [MacKay, 1991]

angenommen und eine Abweichung zwischen Ausgabefunktion und Datenpunkten stark bestraft. Dies resultiert in einer hohen Anpassung an die Daten und eventuellem Overfitting. Bei einem niedrigen β hingegen werden auch höhere Abweichungen zugelassen, was in geringerem Overfitting aber auch in einer ungenügend genauen Approximation der Daten resultieren kann. Auch beim Parameter β muss also der richtige Kompromiss gefunden werden.

Um diese Parameter α und β automatisch aus den vorliegenden Daten zu ermitteln, wird zunächst deren bedingte Wahrscheinlichkeit $P(\alpha, \beta \,|\, D, M)$ betrachtet. Ähnlich wie beim Lernvorgang, bei dem zum Lernen der Gewichte deren Wahrscheinlichkeit maximiert wurde, wird zum Ermitteln der Parameter α und β diese bedingte Wahrscheinlichkeit maximiert. Dafür wird zunächst wieder das Bayestheorem verwendet:

$$P(\alpha, \beta \,|\, D, M) = \frac{P(D \,|\, \alpha, \beta, M)\, P(\alpha, \beta \,|\, M)}{P(D \,|\, M)} \tag{4.17}$$

$P(D \,|\, M)$ ist dabei erneut eine Normalisierungskonstante und kann für das Optimierungsproblem ignoriert werden. Für den Prior $P(\alpha, \beta \,|\, M)$ wird eine Gleichverteilung angenommen. Somit kann die Wahrscheinlichkeit $P(\alpha, \beta \,|\, D, M)$ maximiert werden, indem $P(D \,|\, \alpha, \beta, M)$ maximiert wird. Diese ist aber bereits aus Formel 4.7 bekannt. Formel 4.10, 4.13 und 4.15 in diese eingesetzt und nach der gesuchten Wahrscheinlichkeit $P(D \,|\, \alpha, \beta, M)$ umgestellt, ergibt:

$$P(D \,|\, \alpha, \beta, M) = \frac{\frac{1}{Z_D(\beta) \cdot Z_w(\alpha)} \cdot exp\left(-(\beta \cdot SSE + \alpha \cdot SSW)\right)}{\frac{exp(-(\beta \cdot SSE + \alpha \cdot SSW))}{Z_M(\alpha, \beta)}} \tag{4.18}$$

Und gekürzt:

$$P(D \,|\, \alpha, \beta, M) = \frac{Z_M\,(\alpha, \beta)}{Z_D(\beta) \cdot Z_w(\alpha)} \tag{4.19}$$

Wobei:

$$Z_w(\alpha) = \sqrt{(2\pi \,/\, \alpha)^{N_w}} \qquad \text{und} \qquad Z_D(\beta) = \sqrt{(2\pi \,/\, \beta)^{N}} \tag{4.20}$$

Ermittelt werden muss nun nur noch die frühere Normalisierungskonstante Z_M. Für dieses kann geschlussfolgert werden, da $P(w \,|\, D, \alpha, \beta, M)$ in Formel 4.15 aufintegriert 1 ergeben muss:

$$Z_M\,(\alpha, \beta) = \int\limits_w e^{-E} d^k w \tag{4.21}$$

Die Fehlerfunktion E ist unbekannt, allerdings kann man, wenn man annimmt, dass man sich zumindest im späteren Verlauf des Trainings am oder nahe dem Optimum w_{opt} befindet, für diese eine Taylorreihenentwicklung um w_{opt} durchführen:

$$E = E(w_{opt}) + \frac{1}{2}\,(w - w_{opt})^T\, H\,(w - w_{opt}) \tag{4.22}$$

Dabei stellt w_{opt} der Gewichtsvektor mit den Gewichten am Optimum und H die Hessematrix dar. Der lineare Term konnte dabei, da die erste Ableitung am Optimum gleich 0 ist, weggelassen werden. Dies in Formel 4.21 eingesetzt und integriert ergibt sich:

$$Z_M\,(\alpha, \beta) = e^{-E(w_{opt})}\,(2\pi)^{N_w/2}\, det\left(H^{-1}\right)^{1/2} \tag{4.23}$$

Was wiederum in Formel 4.19 eingesetzt werden kann. Nach Bilden der Log-Likelihood erhält man:

$$\begin{aligned} log\, P(D \,|\, \alpha, \beta, M) = -\alpha \cdot SSW(w_{opt}) - \beta \cdot SSE(w_{opt}) - \frac{1}{2}\, log\, det(H) - \\ - log\, Z_w(\alpha) - log\, Z_D(\beta) + \frac{N_w}{2}\, log\, 2\pi \end{aligned} \tag{4.24}$$

Womit nach einem Ableiten und Nullsetzen schließlich die Werte für die Parameter α und β berechnet werden können:

$$\alpha = \frac{\gamma}{2 \cdot SSW} \tag{4.25}$$

$$\beta = \frac{N - \gamma}{2 \cdot SSE} \tag{4.26}$$

$$\gamma = N_w - 2\alpha \cdot trace\,(H)^{-1} \tag{4.27}$$

γ stellt eine Abschätzung der vom Netz effektiv genutzten Parameter, sprich Gewichte, dar. Es ist für die Bayes'sche Regularisierung selbst nicht von Bedeutung, kann jedoch beispielsweise als Abbruchkriterium für das Network Pruning verwendet werden (siehe Kapitel 4.2.2).
Zur Berechnung von α und β werden diese anfangs mit einem beliebigen Startwert initialisiert und anschließend mit Hilfe der Formeln 4.25 bis 4.27 im Laufe des Trainings des neuronalen Netzes iterativ ihren wahren Werten angenähert.

Bayes'sche Regularisierung und das Levenberg-Marquardt Lernverfahren

Wie bereits oben beschrieben wird die Bayes'sche Regularisierung in das Lernverfahren integriert. Dazu muss dieses jedoch angepasst werden, die neue Fehlerfunktion zu minimieren. Das Levenberg-Marquardt Lernverfahren beispielsweise, welches in dieser Arbeit verwendet wurde, minimiert in seiner Grundvariante aus Kapitel 3.3 ausschließlich die Summe der quadratischen Fehler, nicht aber die Gewichte.
Analog zum Vorgehen in Kapitel 3.3 lautet die angepasste, um die Bayes'sche Regularisierung erweiterte Levenberg-Marquardt Fehlerfunktion:

$$E = \frac{\beta}{2} \left\| \epsilon\,(w_{old}) + J\,(w_{new} - w_{old}) \right\|^2 + \beta \cdot \lambda \left\| w_{new} - w_{old} \right\|^2 + \alpha \left\| w_{new} \right\|^2 \tag{4.28}$$

Wobei J die Jacobimatrix, λ die Lernrate für den Lernalgorithmus und $\epsilon(w_{old}) = y(w_{old}) - t$ ist. $y(w_{old})$ beschreibt dabei den Vektor mit den Netzausgaben und t den Vektor mit den Trainingspattern. Abgeleitet, Null gesetzt und umgestellt ergibt sich daraus die Updateregel des Levenberg-Marquardt Lernverfahrens mit Bayes'scher Regularisierung für die Gewichte des Netzes:

$$w_{new} = w_{old} - \left(\beta \cdot J^T J + (\alpha + \lambda) \cdot I\right)^{-1} \left(\beta \cdot J^T \epsilon\left(w_{old}\right) + \alpha \cdot w_{old}\right) \tag{4.29}$$

Algorithmus 4.1 fasst noch einmal das Vorgehen für das Levenberg-Marquardt Lernverfahren mit Baye'scher Regularisierung zusammen. Es entspricht im Wesentlichen dem Vorgehen des normalen Levenberg-Marquardt Algorithmus, nur dass entsprechend die Fehlerfunktion und die Updateregel für die Gewichte angepasst wurden. Außerdem müssen die Parameter α, β und γ berechnet werden. α und β werden dabei, nach der Empfehlung von [Foresee und Hagan, 1997] mit 0 beziehungsweise 1 initialisiert.

Damit ließe sich nun ein erster Network Pruning Algorithmus realisieren. Wie bereits erwähnt, werden durch die Bayes'sche Regularisierung irrelevante Gewichte gegen Null gedrückt, während für die Lösung des Problems wichtige Gewichte größtenteils einen Wert weit über Null besitzen. Demnach ist es möglich Gewichte mit einem geringen Wert zu löschen, ohne den Approximationsfehler des Netzes signifikant zu verschlechtern. Die entsprechenden Schritte könnten folgendermaßen aussehen:

1. Trainiere ein neuronales Netz mit Hilfe von Algorithmus 4.1

2. Entferne alle Gewichte, welche einen Wert kleiner als eine vorher festgesetzte Schranke e besitzen

3. Entferne alle Hiddenneuronen, welche keine ausgehenden oder keine eingehenden Verbindungen mehr besitzen. Entferne alle Inputneuronen ohne ausgehende Verbindungen.

4. Trainiere das neuronale Netz erneut.

Algorithmus 4.1 Levenberg-Marquardt Lernverfahren mit Bayes'scher Regularisierung

Input e, $nmax$ ▷ e: Schranke für den Fehler ,$nmax$: maximale Anzahl Iterationen
Input t_i ▷ t_i Trainingspattern
$w_{old} \leftarrow$ INITIALISIEREN EURONALES NETZ() ▷ zufällige Initialisierung der Gewichte
$SSE_{old} \leftarrow \frac{\beta}{2} \sum (\, y_i(w_{old}) - t_i)^2 + \alpha \cdot \sum w_{old}^2$ ▷ Summe der quadratischen Fehler
$\lambda \leftarrow 0.1$
$\alpha \leftarrow 0$
$\beta \leftarrow 1$
$n \leftarrow 0$ ▷ Zähler für die Anzahl Durchläufe
repeat
  $n \leftarrow n + 1$
  $J \leftarrow$ BERECHNE JACOBIMATRIX() ▷ berechne Jacobimatrix mit Hilfe des
  ▷ Backpropagation of Error Algorithmus
  $w_{new} \leftarrow w_{old} - \left(\beta \cdot J^T J + (\alpha + \lambda) \cdot I\right)^{-1} \left(\beta \cdot J^T \, \epsilon \, (w_{old}) + \alpha \cdot w_{old}\right)$
  $SSE_{new} \leftarrow \frac{\beta}{2} \sum (\, y_i(w_{new}) - t_i)^2 + \alpha \cdot \sum w_{new}^2$
  if $SSE_{new} \geq SSE_{old}$ **then** ▷ Fehler hat sich verschlechtert
    $\lambda \leftarrow \lambda \cdot 10$ ▷ erhöhe λ, verwirf Gewichtsupdate
  else ▷ Fehler hat sich verbessert
    $w_{old} \leftarrow w_{new}$ ▷ übernimm Gewichtsupdate
    $SSE_{old} \leftarrow SSE_{new}$
    $\lambda \leftarrow \lambda / 10$ ▷ verringere λ
    $\gamma \leftarrow N_w - 2\alpha \cdot trace\,(H)^{-1}$ ▷ aktualisiere BR-Parameter
    $\alpha \leftarrow \gamma \,/\, (2 \cdot SSW)$
    $\beta \leftarrow (N - \gamma) \,/\, (2 \cdot SSE)$
  end if
until $SSE_{old} < e$ **or** $n \geq nmax$ ▷ bis Abbruchkriterium erreicht
return w_{old}

Das Problem eines solchen Algorithmus ist jedoch, dass er das Auswahlkriterium zum Löschen der Gewichte ausschließlich auf der Größe der Gewichte basieren lässt. Aber auch kleine Gewichte können etwas zur Lösung eines Problems beitragen und helfen, die Ausgabefunktion der zu approximierenden Funktion anzupassen. Dies wird in Abbildung 4.9 illustriert. Mithilfe eines RBF-Netzes soll hier eine Funktion (schwarz) abgeschätzt werden. Ein Netz mit nur einem einzigen RBF-Neuron bringt bereits eine relativ brauchbare Abschätzung (blau), für eine optimale Approximation sind jedoch mehr Neuronen erforderlich. Im Vergleich zum ersten Neuron besitzen die folgenden allerdings nur Gewichte von sehr geringer Größe. Ein Network Pruning, welches das Pruning nur aufgrund der Größe der Gewichte vornimmt, würde diese Neuronen eventuell löschen. Aus diesem Grund wurden weitere Network Pruning Algorithmen entwickelt, welche neben der Größe der Gewichte außerdem deren Ableitungen in Betracht ziehen, um zu ermitteln, welchen Einfluss sie auf den Netzwerkfehler besitzen. Einer dieser Algorithmen ist der Optimal Brain Surgeon Algorithmus.

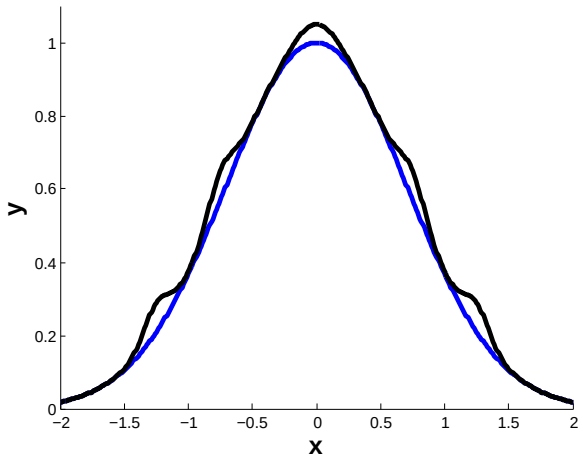

Abbildung 4.9: Illustration des Nutzens kleiner Gewichte in einem neuronalen Netz. Die Funktion ist schwarz dargestellt und soll mit einem RBF-Netz approximiert werden. Die blaue Kurve zeigt die Approximation mit einem RBF-Neuron. Zum optimalen Abschätzen der Funktion sind noch mehr Neuronen notwendig, welche in diesem Fall allerdings im Vergleich zum ersten nur Gewichte von sehr geringer Größe besitzen.

4.2.2 Optimal Brain Surgeon

Der Optimal Brain Surgeon Algorithmus wurde erstmals von [Hassibi et al., 1993] vorgestellt und ist eine Weiterentwicklung des bekannten Optimal Brain Damage Algorithmus [Cun et al., 1990]. Anders als beim ersten naiven Network Pruning Algorithmus aus dem letzten Kapitel, wo die Relevanz der Gewichte ausschließlich über deren Größe bewertet wurde, wird diese hier über die Änderung des Netzwerkfehlers bei Änderung des Gewichtes bewertet. Besitzt das Gewicht keinen oder kaum Einfluss auf den Netzwerkfehler, wird es als irrelevant betrachtet. Das Ziel beim Optimal Brain Surgeon Algorithmus ist es demnach, die Gewichte zu löschen, bei denen sich der Netzwerkfehler am wenigsten ändert.

Ähnlich wie bei seinem Vorgänger, dem Optimal Brain Damage Algorithmus, geht man hier von einem neuronalen Netz aus, welches bis zu einem, zumindest lokalen, Minimum trainiert wurde. Ist dies geschehen, kann eine Taylorreihenentwicklung der Fehlerfunktion bis zum quadratischen Term durchgeführt werden:

$$E(w) = E(w_{Min}) + \left(\frac{\partial E}{\partial w}\right)^T \Delta w + \frac{1}{2}\Delta w^T H w \tag{4.30}$$

Dabei ist w_{Min} der Gewichtsvektor am Minimum. Dadurch, dass bis zu diesem (lokalen) Minimum trainiert wurde, ist bei der Taylorreihenentwicklung der lineare Term gleich Null und damit die Abschätzung der Fehlerfunktion:

$$E(w) = E(w_{Min}) + \frac{1}{2}\Delta w^T H w \tag{4.31}$$

Da für das Verfahren die Gewichte gelöscht werden sollen, welche die geringste Änderung beim Netzwerkfehler bewirken, wird im Folgenden der Netzwerkfehler betrachtet:

$$E(w) - E(w_{Min}) = \frac{1}{2}\Delta w^T H w \tag{4.32}$$

Das Ziel ist es nun, eines der Gewichte Null zu setzen, was, da $\Delta w = w - w_{opt}$, durch

folgendenden Ausdruck erreicht werden kann:

$$e_q^T \cdot \Delta w + w_{Min,q} = 0 \tag{4.33}$$

Dabei stellt e_q den Einheitsvektor dar, dessen q. Element $= 1$ ist und $w_{Min,q}$ bezeichnet das Gewicht, welches Null gesetzt werden soll. Das Problem lässt sich nun als Optimierungsproblem beschreiben:

$$\min_q \left\{ \min_{\Delta w} \left(\frac{1}{2} \Delta w^T H w \right) \mid \quad e_q^T \cdot \Delta w + w_{Min,q} = 0 \right\} \tag{4.34}$$

Es wird also nicht nur nach dem Gewicht gesucht, bei welchem die Änderung des Netzwerkfehlers möglichst minimal ausfällt, sondern gleichzeitig nach dem zugehörigen Gewichtsvektor w.
Gelöst werden kann das Optimierungsproblem mit der Lagrange-Multiplikatorenregel:

$$S = \frac{1}{2} \Delta w^T H w - \lambda \left(e_q^T \cdot \Delta w + w_{Min,q} \right) \tag{4.35}$$

Null gesetzt, jeweils nach Δw und λ abgeleitet und schließlich eingesetzt ergibt sich:

$$\Delta w = -\frac{w_{Min,q}}{[H^{-1}]_{q,q}} H^{-1} \cdot e_q \tag{4.36}$$

und

$$S_q = \frac{1}{2} \cdot \frac{w_{Min,q}^2}{[H^{-1}]_{q,q}} \cdot e_q \tag{4.37}$$

Dabei bezeichnet $[H^{-1}]_{q,q}$ das q. Diagonalelement der invertierten Hessematrix. S_q ist die sogenannte Saliency für Gewicht q. Sie stellt die bereits beschriebene Änderung des Netzwerkfehlers bei Änderung des Gewichtes q dar und kann als Maß für die Relevanz des Gewichtes betrachtet werden. Mit ihrer Hilfe wird bei jedem Schritt das am wenigsten relevante Gewicht des Netzes ermittelt und gelöscht. Die verbliebenen

Gewichte werden mit Hilfe von Formel 4.36 aktualisiert. Als Abbruchkriterium kann beispielsweise der Wert γ, welcher von der Bayes'schen Regularisierung aus Kapitel 4.2.1 geliefert wird, verwendet werden. Dieser stellt eine Abschätzung der effektiv vom Netz verwendeten Gewichte dar. Solange die Anzahl der Gewichte im neuronalen Netz also noch größer als γ ist, können theoretisch noch weitere Gewichte entfernt werden. Da γ allerdings oftmals etwas zu niedrig ausfällt, empfiehlt es sich, noch einen Puffer e von beispielsweise 20% an zusätzlichen Gewichten beizubehalten.

Algorithmus 4.2 Optimal Brain Surgeon Algorithmus

Input e ▷ e: Schranke für die Anzahl Gewichte
$(w, N_w, \alpha, \beta, \gamma) \leftarrow$ TRAINNN_LM_BR() ▷ trainiere ein neuronales Netz mit ▷ Algorithmus 4.1 bis zu einem Minimum
while $(1 + e) \cdot \gamma < N_w$ **do**
  $J \leftarrow$ BERECHNEJACOBIMATRIX() ▷ berechne Jacobimatrix mit Hilfe des ▷ Backpropagation of Error Algorithmus
  for $q = 1 \rightarrow N_w$ **do** ▷ für alle Gewichte
    $S_q \leftarrow \frac{1}{2} \cdot w_{Min,q}^2 \,/\, [H^{-1}]_{q,q}$ ▷ berechne Saliency
  end for
  $Q \leftarrow \min_q(S_q)$ ▷ $w_{Min,Q}$: Gewicht mit minimaler Saliency
  $\Delta w \leftarrow -w_{Min,Q} \,/\, [H^{-1}]_{q,q} \cdot H^{-1} \cdot e_q$
  $w = w + \Delta w$ ▷ Gewichte aktualisieren
  $w_{Min,Q} \leftarrow 0$ ▷ lösche das Gewicht mit minimaler Saliency
  $N_w \leftarrow N_w - 1$
end while
$w \leftarrow$ TRAINNN_LM_BR() ▷ trainiere das neuronale Netz mit Algorithmus 4.1
return w

Algorithmus 4.2 fasst das Verfahren als Pseudocode zusammen. Zu Beginn wird ein neuronales Netz bis zu einem Minimum trainiert und die Inverse der Hessematrix mit Hilfe der vom Backpropagation of Error Algorithmus erhaltenen Jacobimatrix J und den vom Training des neuronalen Netzes erhaltenen Parametern α und β ermittelt.

Damit wird für jedes Gewicht im neuronalen Netz die Saliency S_q berechnet. Das Gewicht mit der geringsten Saliency und damit das am wenigsten relevante Gewicht kann somit gelöscht und die restlichen Gewichte mit Δw aktualisiert werden. Dies wird so lange wiederholt, bis die Anzahl der Gewichte N_w im Netz dem Wert γ, plus einem zusätzlichen Puffer e, entspricht. Anschließend kann das neuronale Netz noch einmal neu trainiert werden, um, durch die nun hoffentlich deutlich niedrigere Anzahl von Gewichten, eventuell eine bessere Lösung zu finden.

4.3 Verfahren zur Problemzerlegung

Mit den Lernverfahren und der Möglichkeit, die Größe eines neuronalen Netzes durch Network Pruning automatisch an das Problem anzupassen, sollen im Folgenden die grundlegenden Verfahren für eine nichthierarchisch funktionale Problemdekomposition, wie sie in Kapitel 2.3.2 beschrieben wurde, vorgestellt werden. Zur Illustration der Verfahren wird dabei das Beispielproblem aus Kapitel 2.2 verwendet. In Abbildung 4.10 links sind noch einmal die dabei verwendeten Problemgrößen mitsamt den Abhängigkeiten (Pfeile) und der korrekten Dekomposition in zwei Teilprobleme (rote Umrandungen) zu sehen.
Das vorrangige Ziel wird es dabei sein, die optimale, dem Problem entsprechende Dekomposition in Teilprobleme vorzunehmen, also die Partitionierung der Problemgrößen so, dass keine der vorhandenen Abhängigkeiten die Partitionsgrenzen überschreitet, wie dies im linken Bild am Beispiel dargestellt ist. Dabei soll angenommen werden, dass kein Vorwissen über das Problem vorhanden ist, insbesondere die Abhängigkeiten der Problemgrößen untereinander sind unbekannt (Abbildung 4.10 rechts). Allerdings wird, zumindest vorerst, die Anzahl der Teilprobleme vorgegeben werden, was entweder durch Vorwissen über das Problem, oder einfach durch eine grobe Schätzung geschehen kann, denn auch eine Zerlegung in weniger als die maximal mögliche Anzahl Teilprobleme bringt bereits Vorteile. Später wird jedoch in Kapitel 4.4.3 auch eine Möglichkeit vorgestellt, diese Anzahl der Teilprobleme automatisch zu ermitteln.

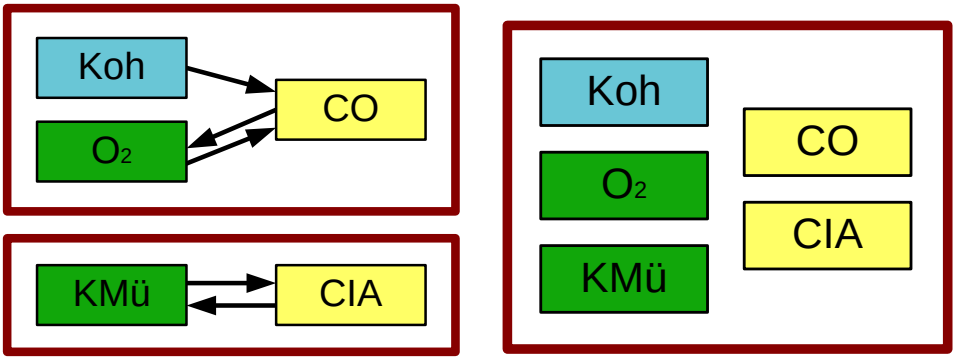

Abbildung 4.10: Veranschaulichung der Aufgabe der problemzerlegenden Verfahren am Beispielproblem. Die linke Abbildung zeigt die Problemgrößen des Beispielproblems mit ihren Abhängigkeiten (Pfeile) und der Zerlegung in die beiden Teilprobleme (rote Umrandung). Weder die Teilprobleme noch die Abhängigkeiten sind jedoch anfangs bekannt (rechte Abbildung).

4.3.1 Problemzerlegung durch Input Pruning

Die erste und wahrscheinlich einfachste Variante eine Problemzerlegung vorzunehmen ist, die Abhängigkeiten der Problemgrößen durch Input Pruning zu ermitteln, um anschließend die Zusammenhangskomponenten, beispielsweise durch eine Variante von Tiefensuche, zu bestimmen und damit eine Problemzerlegung durchzuführen.

Die Idee des Input Prunings ist, die Eingangsgrößen eines neuronalen Netzes von allen irrelevanten, welche nicht zur Lösung des Problems benötigt werden, zu befreien. Die dadurch ermöglichte geringere Anzahl von Eingangsgrößen verringert die Komplexität, man erhält ein niedrigdimensionaleres Fehlergebirge und bekommt allgemein bessere Lösungen, ähnlich, wie dies schon in Kapitel 2.4 zur Beschreibung der Vorteile einer Problemzerlegung erläutert wurde.
Das Input Pruning kann beispielsweise durch Evolution geschehen. Die Individuen stellen in diesem Fall neuronale Netze dar, welche das Problem approximieren. Bei der Reproduktion werden die verwendeten Eingangsgrößen durch Mutation und Rekombination (siehe Kapitel 3.2) verändert und ihr Approximationsfehler berechnet.

Dabei ist zu erwarten, dass neuronale Netze mit einer möglichst kleinen Menge an Eingangsgrößen, welche dennoch alle zur Lösung des Problems wichtigen Größen enthalten, aus oben genannten Gründen im Schnitt einen geringeren Approximationsfehler besitzen und sich im Laufe der Evolution durchsetzen.
Die Mutation der Inputs geschieht dabei dadurch, dass jeder Input mit einer bestimmten Wahrscheinlichkeit, der sogenanten Mutationsrate, invertiert wird, also deaktiviert, falls er aktiviert war und entsprechend wieder aktiviert, falls er deaktiviert war. Die Mutationsrate selbst kann dabei mit in das Chromosom aufgenommen und mit der in Kapitel 3.2.2 beschriebenen lokalen Varianzanpassung evolviert werden.

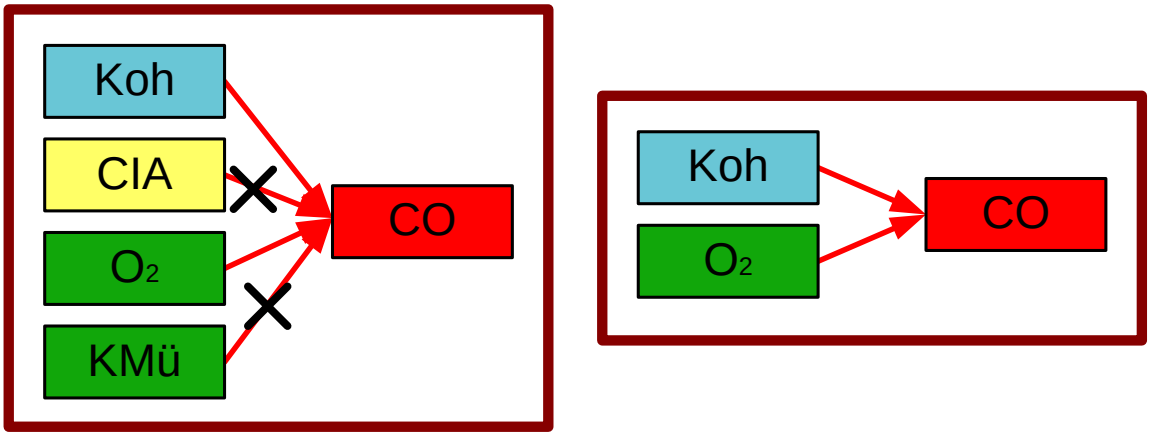

Abbildung 4.11: Illustration der Problemzerlegung durch Input Pruning für die Problemgröße des Kohlenmonoxidgehaltes aus dem Beispielproblem. Das linke Bild zeigt die Problemgröße Kohlenmonoxid (rot) mit den zu Beginn des Verfahrens angenommenen Abhängigkeiten zu allen anderen Problemgrößen (rote Pfeile). Durch Input Pruning lassen sich die tatsächlich vorhandenen Abhängigkeiten auf zwei der Problemgrößen reduzieren (rechtes Bild).

Bei der Problemzerlegung lässt sich das Input Pruning dazu einsetzen, die Abhängigkeiten der einzelnen Problemgrößen zu ermitteln. Dazu wird auf jeweils jede Mess- und Regelgröße ein neuronales Netz trainiert, welches potenziell alle anderen Problemgrößen als Eingangsgrößen besitzt. Dies ist in Abbildung 4.11 links für die Größe Kohlenmonoxid aus dem Beispielproblem zu sehen. Wendet man darauf

ein Input Pruning Verfahren an, steht zu erwarten, dass alle in der Realität nicht vorhandenen Abhängigkeiten, welche schließlich nichts zur Approximation der Größe Kohlenmonoxid beitragen und im Gegenzug aber die Komplexität für das neuronale Netz erhöhen, entfernt werden (Abbildung 4.11 rechts). Dies führt man für alle Problemgrößen des Gesamtproblems durch, wie dies der Vollständigkeit halber in Abbildung 4.12 für das Beispielproblem getan wurde. Sind alle Abhängigkeiten bestimmt, wie dies in Abbildung 4.12 unten rechts zu sehen ist, kann man, beispielsweise durch Tiefensuche, die Zusammenhangskomponenten im Abhängigkeitsgraphen ermitteln. Da jede Zusammenhangskomponente ein Teilproblem darstellt, kann auf diese Weise eine Problemdekomposition vorgenommen werden und man erhält für das Beispielproblem eine Problemzerlegung wie in Abbildung 4.10 links dargestellt.

Algorithmus 4.3 fasst die gesamte Problemzerlegung durch Input Pruning als Pseudocode zusammen. Das Grundschema entspricht dabei dem des evolutionären Algorithmus, welcher in Kapitel 3.2 vorgestellt wurde. Es wird für jede Mess- und Regelgröße eine eigene Evolution gestartet. Dabei wird jedes Mal zuerst eine Population von μ Individuen zufällig initialisiert und jedes davon mit dem Levenberg-Marquardt Lernverfahren mit Bayes'scher Regularisierung trainiert und dem Optimal Brain Surgeon Algorithmus gepruned (Algorithmus 4.2). Jedes der neuronalen Netze kann dabei potentiell alle anderen Problemgrößen, ausgenommen sich selbst, als Inputs bekommen. Anschließend wird die Fitness jedes der Individuen der Population über den Root Mean Squared Error aus Formel 4.3 ermittelt.
Auf Basis dieser Fitness kann die aktuelle Population der neuronalen Netze absteigend nach ihrer Fitness sortiert werden, sodass im folgenden Schritt λ Individuen selektiert und ebenso viele neue Individuen durch Mutation und Rekombination der verwendeten Eingangsgrößen erstellt werden können. Die hier verwendeten Varianten der Selektion, Mutation und Rekombination wurden jeweils in den Kapiteln 3.2.1, 3.2.2 und 3.2.3 erläutert. Diese neu erzeugten Individuen werden trainiert, gepruned und der bestehenden Population hinzugefügt. Bei dem erneuten Training werden die Gewichte allerdings nicht komplett neu initialisiert, sondern, falls möglich, von ihren Eltern übernommen. Diese Form der Weitergabe von während der „Lebenszeit“

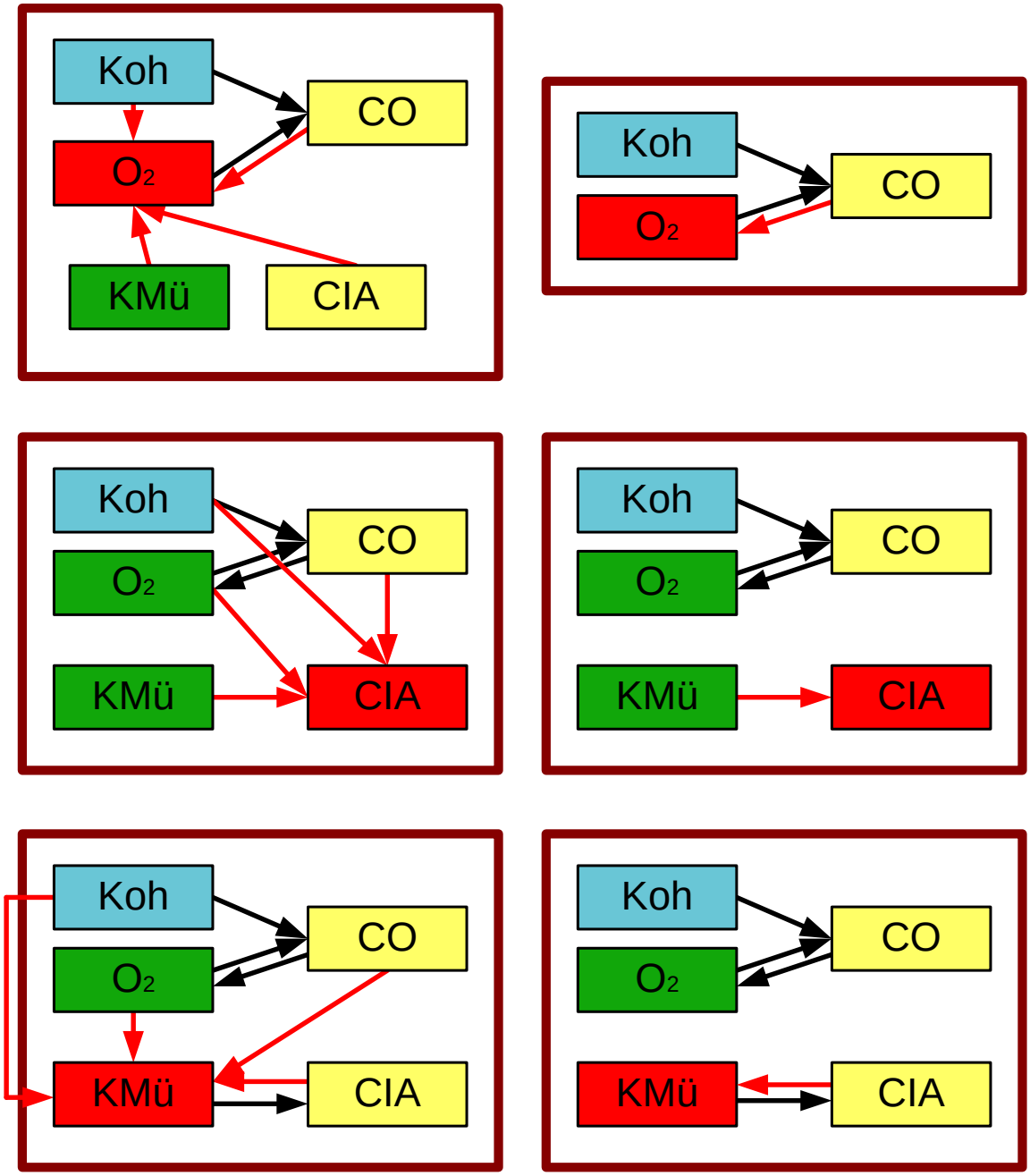

Abbildung 4.12: Illustration der Problemzerlegung durch Input Pruning für das Beispielproblem. Die linke Spalte zeigt die jeweils aktuelle Problemgröße (rot) mit den zu Beginn der Iteration angenommenen Abhängigkeiten zu allen anderen Problemgrößen (rote Pfeile). Durch Input Pruning lassen sich die Abhängigkeiten auf die tatsächlich vorhandenen reduzieren (rechte Spalte).

erlernten Eigenschaften nennt sich Lamarckismus und wird u.a. in [Ku und Mak, 1997] beschrieben. Sie verringert hier zum Einen die Trainingszeit und ermöglicht es zum Anderen, das Wissen der Eltern zu nutzen und beim Training von einem guten Punkt aus zu starten. Nachdem die Fitness der neuen Individuen berechnet wurde, kann die Population erneut nach ihrer Fitness sortiert und, um die Größe der Population konstant zu halten, die schlechtesten λ Individuen gelöscht werden. Dies wird bis zu einem vorher bestimmten Abbruchkriterium, beispielsweise einer festen Anzahl an Iterationen, wiederholt und für alle Populationen und damit für alle Mess- und Regelgrößen durchgeführt. Mit Hilfe der so ermittelten Abhängigkeiten der Problemgrößen kann ein Abhängigkeitsgraph entsprechend der Abbildung 4.12 unten rechts erstellt und die Zusammenhangskomponenten durch wiederholte Anwendung von Tiefensuche ermittelt werden. Die so erhaltenen Zusammenhangskomponenten entsprechen den gesuchten Teilproblemen.

Ein Vorteil der Problemzerlegung durch Input Pruning ist, dass nicht nur die Teilprobleme selbst, sondern auch gleich die Abhängigkeiten innerhalb dieser ermittelt werden, was zum grundlegenden Verständnis des Prozesses beitragen kann und es ermöglichen könnte, die Partitionen noch weiter zu zerlegen. Auch werden für alle Problemgrößen neuronale Netze trainiert, welche beispielsweise für Regelzwecke eingesetzt werden können. Ein weiterer Vorteil ist, dass kein Wissen über die Anzahl der vorhandenen Teilprobleme benötigt wird, wie dies bei den folgenden Verfahren zur Problemzerlegung der Fall ist. Die Anzahl wird bei der Problemzerlegung durch Input Pruning mit Hilfe des Zusammenhangstests automatisch ermittelt.
Der größte Nachteil ist seine Fehleranfälligkeit, welche durch das Input Pruning hervorgerufen wird. Neuronale Netze, welche für die Lösung ihres Problems zu wenige Inputs besitzen, wenn also wichtige Abhängigkeiten fehlen, können nicht hoffen, einen guten Approximationsfehler zu erreichen. Sind jedoch zu viele Abhängigkeiten vorhanden, besitzt also das neuronale Netz nicht nur die für die zur Lösung des Problems relevanten Inputs, sondern auch noch zusätzliche, irrelevante, erschwert dies durch die zusätzliche Komplexität und die höhere Anzahl Parameter zwar das Training durch das Lernverfahren, das Problem bleibt jedoch prinzipiell lösbar, da grundsätzlich

Algorithmus 4.3 Problemzerlegung durch Input Pruning

Input μ, λ ▷ μ: Größe der Population ,λ: Anzahl Nachkommen
Input $x[1..N_x]$ ▷ x: Wert der vergangenen Problemgrößen
Input $y[1..N_y]$ ▷ y: Werte der aktuellen Mess- und Regelgrößen
Input $nmax$ ▷ $nmax$: maximale Anzahl Iterationen der Evolution
for $i = 1 \rightarrow N_y$ **do** ▷ für jede Mess- und Regelgröße
  $n \leftarrow 0$ ▷ n: aktuelle Anzahl Iterationen der Evolution
  $P_i[1..\mu] \leftarrow$ INITIALISIEREPOPULATION() ▷ zufällige Initialisierung der Netze
  for $j = 1 \rightarrow \mu$ **do** ▷ für alle Individuen der Population
    $P_i(j) \leftarrow$ TRAIN$(x, y(P_i(j)))$ ▷ trainiere neuronales Netz j mit Alg. 4.2
    $Fitness_i(j) \leftarrow RMSE_i(j)$ ▷ berechne Fitness über Formel 4.3
  end for
  $P_i \leftarrow$ SORT(P_i) ▷ sortiere Individuen der Population absteigend nach Fitness
  repeat
    $n \leftarrow n + 1$
    for $j = 1 \rightarrow \lambda$ **do**
      $k \leftarrow$ SELEKTION() ▷ selektiere Individuum k mit Wkt. aus Formel 3.1
      $P_i(\mu + j) \leftarrow$ REPRODUKTION$(P_i(k))$ ▷ Mutation/Rekombination der
      ▷ Inputs nach Kapitel 3.2.2 und 3.2.3
      $P_i(\mu + j) \leftarrow$ TRAIN$(x, y(P_i(\mu + j)))$ ▷ trainiere neuronales Netz mit
      ▷ Algorithmus 4.2
      $Fitness_i(\mu + j) \leftarrow RMSE_i(\mu + j)$ ▷ berechne Fitness über Formel 4.3
    end for
    $P_i \leftarrow$ SORT(P_i) ▷ sortiere Individuen der Population absteigend nach Fitness
    $P_i[\mu + 1, .., \mu + \lambda] \leftarrow \{\}$ ▷ lösche Individuen $\mu + 1$ bis $\mu + \lambda$
  until $n \geq nmax$ ▷ bis Abbruchkriterium erreicht
end for
$G_{abh} \leftarrow$ ERSTELLEABHÄNGIGKEITSGRAPH$(P_1(1), .., P_{N_y}(1))$
$(C_1, \ldots, C_m) \leftarrow$ DFS(G_{abh}) ▷ ermittle Komponenten von G_{abh} über Tiefensuche
return $C_1, \ldots, C_m$

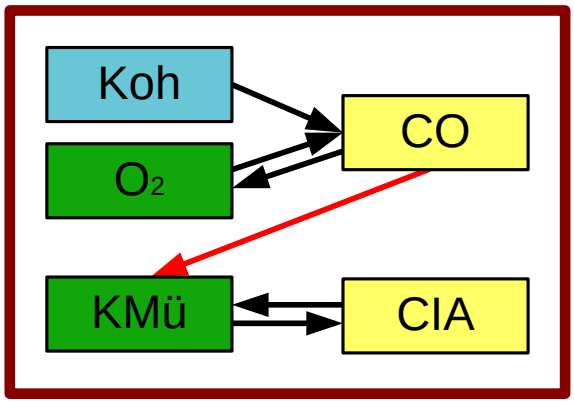

Abbildung 4.13: Illustration der Nachteile der Problemzerlegung durch Input Pruning am Beispielproblem. Eine einzige falsche Abhängigkeit (roter Pfeil) genügt, um das Problem nicht mehr in zwei Teilprobleme zerlegen zu können.

keine für die Lösung des Problems wichtigen Informationen fehlen. Auch verringert sich der negative Einfluss der unnötigen Inputs mit deren Zahl. Der Unterschied in der Fehlerrate bei einem neuronalen Netz mit nur den relevanten Inputs und einem mit zusätzlich einem oder einigen wenigen irrelevanten Inputs ist zwar im Mittel vorhanden, jedoch, abhängig von der Komplexität des Problems, oftmals eher gering. Wenn man zusätzlich den Zufallsfaktor beim Training neuronaler Netze in Betracht zieht, kann es daher nicht verwundern, dass das oben vorgestellte evolutionäre Input Pruning Verfahren gelegentlich einige nicht vorhandene Abhängigkeiten ermittelt. Zusätzliche, nicht vorhandene Abhängigkeiten sind jedoch für die Zerlegung des Problems in Teilprobleme kritisch, da sie oftmals über die Teilproblemgrenzen hinweg bestehen und damit eine mögliche Partitionierung stören. Dies wird in Abbildung 4.13 anhand des Beispielproblems illustriert. Auch wenn alle anderen Abhängigkeiten (schwarze Pfeile) richtig klassifiziert wurden, so genügt doch eine einzige falsche Abhängigkeit (roter Pfeil) und die Zerlegung in zwei Teilprobleme ist nicht mehr möglich.

So zeigen auch die Tests in Kapitel 5.2, dass das Input Pruning zwar alle für die Lösung des Problems wichtigen Größen erkennt, jedoch oftmals einige Abhängigkeiten zuviel ermittelt und damit die Problemzerlegung behindert oder gar zunichte macht.

4.3.2 Problemzerlegung durch Partitionierung Variante 1

Die Problemzerlegung durch Partitionierung wurde entworfen, um auf die Nachteile der Problemzerlegung durch Input Pruning einzugehen. Aufgrund der Fehleranfälligkeit des Input Prunings, welches oftmals zu viele, nicht vorhandene Abhängigkeiten erkennt, kann ein Abhängigkeitsgraph mit weniger Komponenten und damit weniger ermittelten Teilproblemen als tatsächlich vorhanden entstehen.
Einer der Hauptgründe dafür ist, dass die Problemzerlegung durch Input Pruning keine Rücksicht auf mögliche Problemzerlegungen nimmt. Würde in Abbildung 4.13 die rote Abhängigkeit entfernt werden, könnte man das Problem wieder zerlegen, und dies ohne, dass der Approximationsfehler darunter leiden würde. Das Verfahren sieht dies jedoch nicht vor, da die Abhängigkeiten der Problemgrößen unabhängig voneinander ermittelt werden, d.h. ohne dass Rücksicht darauf genommen wird, welche Abhängigkeiten die anderen Problemgrößen besitzen und ob sich bereits Partitionen herausgebildet haben, selbst wenn dadurch am Ende im Extremfall ein komplett zusammenhängender Abhängigkeitsgraph entsteht, die Problemzerlegung also gescheitert wäre.

Die Idee der Problemzerlegung durch Partitionierung ist deshalb, eine Problemzerlegung zu erzwingen. Um dies zu erreichen, werden die Probleme in mehrere Partitionen zerlegt, welche den Teilproblemen entsprechen. Zwei mögliche Partitionierungen für das Beispielproblem sind einmal in Abbildung 4.14 dargestellt. Das linke Bild zeigt eine inkorrekte Partitionierung, da Abhängigkeiten über die Teilproblemgrenzen hinweg bestehen (rote Pfeile). Im rechten Bild ist die korrekte Partitionierung zu sehen. Zu beachten ist, dass in der Abbildung zur Illustration der Sachverhalte die Abhängigkeiten zwar eingezeichnet sind, diese aber, anders als bei der Problemzerlegung durch Input Pruning, durch dieses Verfahren nicht ermittelt werden. Da die tatsächlichen Teilprobleme und damit die korrekten Partitionen natürlich anfangs noch nicht bekannt sind, müssen diese evolviert werden. Dies wird wie gewohnt durch Rekombination und Mutation bewerkstelligt, wobei die Mutation der Partitionierung dadurch geschieht, dass jede Problemgröße mit einer Mutationsrate ausgewählt und in eine zufällige andere

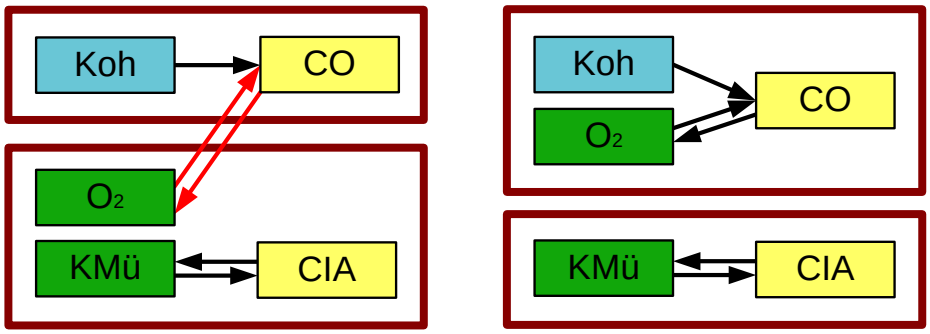

Abbildung 4.14: Illustration der Problemzerlegung durch Partitionierung für das Beispielproblem. Gezeigt sind zwei unterschiedliche Partitionierungen. In der linken Abbildung ist eine Partitionierung zu sehen, bei der Abhängigkeiten über die Partitionsgrenzen hinweg bestehen (rot), was sich in einem schlechten Approximationsfehler niederschlagen würde. Rechts ist die korrekte Partitionierung zu sehen.

Partition verschoben wird. Die Mutationsrate kann dabei in das Chromosom aufgenommen und über die lokale Varianzanpassung aus Kapitel 3.2.2 mit evolviert werden. Dafür werden die Problemgrößen der Partitionen durch neuronale Netze approximiert. Jeder Partition wird ein neuronales Netz zugeteilt, welches alle dort enthaltenen Mess- und Regelgrößen für den aktuellen Zeitpunkt t approximiert, mit den Werten aller Problemgrößen aus der Vergangenheit als Inputs. Die ist in Abbildung 4.15 einmal für die in Abbildung 4.14 links gezeigte (inkorrekte) Partitionierung dargestellt. Ist die Partitionierung dabei inkorrekt, bestehen also Abhängigkeiten über die Partitionsgrenzen hinweg, stehen für die Approximierung der Mess- und Regelgrößen nicht genügend Informationen zur Verfügung und der Approximationsfehler fällt entsprechend hoch aus. So kann beispielsweise bei dem Beispiel aus Abbildung 4.15 das Kohlenmonoxid (CO) nicht ohne das Wissen über die Luftzufuhr (O_2) approximiert werden, und umgekehrt.

Es ist klar, dass je mehr Problemgrößen einer falschen Partition zugeordnet werden und je mehr Abhängigkeiten die Teilproblemgrenzen überschreiten, desto schlechter der Approximationsfehler, welcher, da er hier als Fitnesswert verwendet wird, zum Aussterben solcher ungeeigneter Partitionierungen führt, während die korrekte Partitionierung, sobald sie im Laufe der Evolution gefunden wurde, den besten

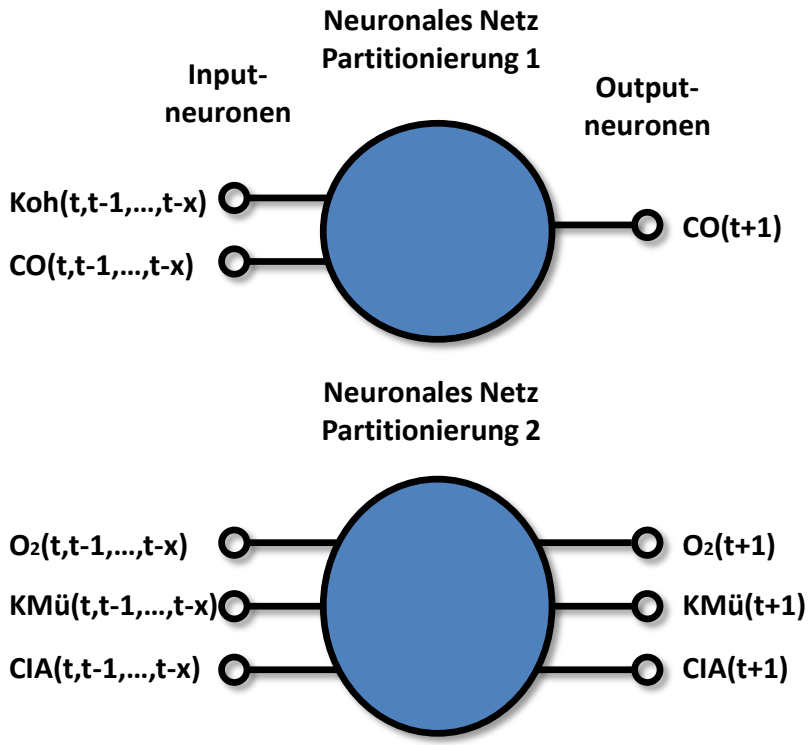

Abbildung 4.15: Veranschaulichung des Trainings der neuronalen Netze bei der Problemzerlegung durch Partitionierung. Für jede Partition wird ein eigenes neuronales Netz trainiert, jeweils auf alle Problemgrößen gleichzeitig.

Approximationsfehler und damit die beste Fitness besitzt, da alle Abhängigkeiten und alle zur Approximation der Problemgrößen wichtigen Informationen vorhanden sind.

Der komplette Algorithmus zur Problemzerlegung durch Partitionierung ist in Algorithmus 4.4 als Pseudocode dargestellt. Zu Beginn werden μ Individuen, welche zufällig ausgewürfelte Partitionen repräsentieren, erstellt, mit Hilfe des Levenberg-Marquardt Lernverfahrens trainiert und das Optimal Brain Surgeon Verfahren angewendet, um ein Network Pruning durchzuführen. Schließlich wird ihre Fitness auf Basis des Root Mean Squared Errors aus Formel 4.4 berechnet.
Mit Hilfe dieses Fitnesswertes kann die Population der Individuen absteigend nach ihrer Fitness sortiert werden, um sie anschließend zu evolvieren. Dafür werden

Algorithmus 4.4 Problemzerlegung durch Partitionierung Variante 1

Input μ, λ ▷ μ: Größe der Population ,λ: Anzahl Nachkommen
Input $x[1..N_x]$ ▷ x: Wert der vergangenen Problemgrößen
Input $y[1..N_y]$ ▷ y: Werte der aktuellen Mess- und Regelgrößen
Input $nmax$, N_P ▷ $nmax$: max. Anzahl Iterationen, N_P: Anzahl Teilprobleme
for $i = 1 \rightarrow \mu$ **do** ▷ für alle Individuen der Population
  $P_i[1..N_P] \leftarrow$ INITIALISIEREPOPULATION(x, y) ▷ zufällige Initialisierung
  ▷ $P_i(j)$: Liste mit Problemgrößen in Partition j und Individuum i
  for $j = 1 \rightarrow N_P$ **do** ▷ für alle Partitionen
    $N_i(j) \leftarrow$ TRAIN$(x(P_i(j)), y(P_i(j)))$ ▷ trainiere Netz mit Alg. 4.2 auf
    ▷ alle Größen in $P_i(j)$
  end for
  $Fitness_i \leftarrow RMSE_i$ ▷ berechne Fitness über Formel 4.4
end for
$(N, P) \leftarrow$ SORT$(Fitness)$ ▷ sortiere Individuen der Pop. absteigend nach Fitness
$n \leftarrow 0$ ▷ n: aktuelle Anzahl Iterationen der Evolution
repeat
  $n \leftarrow n + 1$
  for $i = 1 \rightarrow \lambda$ **do**
    $k \leftarrow$ SELEKTION() ▷ selektiere Individuum k mit Wkt. aus Formel 3.1
    $P_{\mu+i} \leftarrow$ REPRODUKTION(P_k) ▷ Mutation/Rekombination der
    ▷ Partitionen nach Kapitel 3.2.2 und 3.2.3
    for $j = 1 \rightarrow N_P$ **do** ▷ für alle Partitionen
      $N_{\mu+i}(j) \leftarrow$ TRAIN$(x(P_{\mu+i}(j)), y(P_{\mu+i}(j)))$
    end for
    $Fitness_{\mu+i} \leftarrow RMSE_{\mu+i}$ ▷ berechne Fitness über Formel 4.4
  end for
  $(N, P) \leftarrow$ SORT$(Fitness)$ ▷ sortiere Ind. der Pop. absteigend nach Fitness
  $P_i[\mu + 1, .., \mu + \lambda] \leftarrow \{\}$ ▷ lösche Individuen $\mu + 1$ bis $\mu + \lambda$
until $n \geq nmax$ ▷ bis Abbruchkriterium erreicht
return P_1

zunächst λ Individuen nach dem in Kapitel 3.2.1 beschriebenen Verfahren selektiert und durch Mutation und Rekombination der Partitionierung (Kapitel 3.2.2 und 3.2.3) aus diesen selektierten Individuen ebenso viele Nachkommen erzeugt, welche trainiert und gepruned werden, wobei auch hier wieder Lamarckismus zum Einsatz kommt, die Individuen also die Gewichte der Eltern, falls möglich, vor dem Training übernehmen. Nachdem dann die Fitness der neuen Individuen berechnet wurde, kann die Population erneut absteigend nach ihrer Fitness sortiert und, um die Größe der Population konstant zu halten, die schlechtesten λ Individuen gelöscht werden. Dies wird bis zu einem vorher bestimmten Abbruchkriterium, beispielsweise einer festen Anzahl an Iterationen, wiederholt. Die Partitionen des Individuums mit der höchsten Fitness werden schlussendlich als Ergebnis der Problemzerlegung ausgegeben.

Der Vorteil der Problemzerlegung durch Partitionierung ist vor allem, dass es auf jeden Fall zu einer Problemzerlegung kommt. So ist es beispielsweise nicht möglich, dass am Ende des Verfahrens nur eine einzige Partition ausgegeben wird, welche alle Problemgrößen enthält, wie dies bei der Partitionierung durch Input Pruning vorkommen kann. Die Zahl der entstehenden Partitionen und damit der Teilprobleme muss bei diesem Verfahren allerdings dem Algorithmus vorgegeben werden. Dies erfordert entweder Vorwissen oder eine automatische Abschätzung (Kapitel 4.4.3). Ist die Anzahl der vorgegebenen Teilprobleme dabei höher als die der tatsächlich vorhandenen, wird das Problem allerdings auch in zu viele Teilprobleme zerlegt. Der Algorithmus sucht dabei eine Art „minimalen Schnitt“, also das Problem so zu zerlegen, dass möglichst wenige und möglichst unwichtige Abhängigkeiten über die Teilproblemgrenzen hinweg reichen. Ist die Anzahl der Teilprobleme dagegen geringer als die der tatsächlich Vorhandenen, geht Potenzial verloren. Da allerdings auch eine Zerlegung in weniger Teilprobleme bereits schon merkliche Vorteile bringt, empfiehlt es sich im Zweifelsfall, dem Verfahren eher etwas weniger Partitionen vorzugeben.

Ein kleiner Nachteil dagegen ist, dass in der oben vorgestellten Grundvariante, anders als beim Input Pruning Ansatz, die Abhängigkeiten selbst nicht ermittelt werden, sondern nur die Zerlegung in Teilprobleme. Aber auch die Kenntnis der Abhängigkeiten

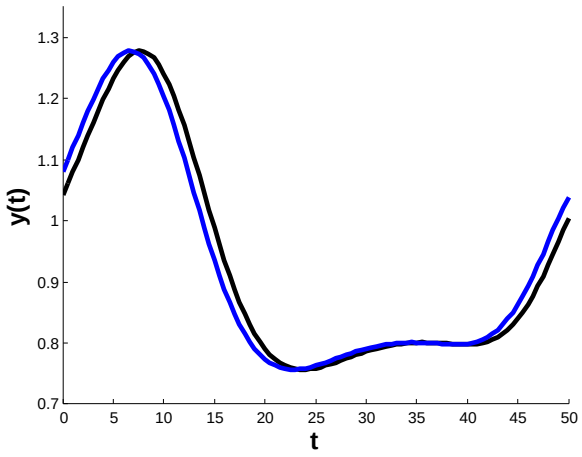

Abbildung 4.16: Approximation einer Problemgröße (schwarze Kurve) durch vergangene Werte von sich selbst (blaue Kurve). Die Approximation $\hat{f}(t)$ kann hier einfach gleich dem vergangenen Wert $f(t-1)$ gesetzt werden.

kann hilfreich sein. Dem kann jedoch mit etwas Zusatzaufwand, beispielsweise durch den nachfolgenden Einsatz des Input Pruning Verfahrens, abgeholfen werden.

Ein größeres Problem ist allerdings, dass, dadurch, dass die Problemgrößen einer Partition alle durch ein einziges neuronales Netz gemeinsam approximiert werden, mit den gleichen Problemgrößen - allerdings den Werten aus der Vergangenheit - als Inputs, Problemgrößen sich selbst als Inputs erhalten. Auf der einen Seite ist es auch in der Praxis oftmals so, dass die Werte der aktuellen Größen von ihren Werten aus der Vergangenheit abhängig sind. Ein Beispiel hierfür ist die Temperatur. Durch die Zufuhr von Wärmeenergie steigt die Temperatur, der absolute Wert hängt jedoch davon ab, wie hoch die Temperatur vor der Zufuhr der Energie war. Auf der anderen Seite macht die Kenntnis der Werte der Größe aus der Vergangenheit die Approximation für das neuronale Netz zu leicht. Dies ist in Abbildung 4.16 einmal veranschaulicht. Dargestellt ist eine beliebige Zeitreihe (schwarze Kurve). Im einfachsten Fall ließe sich die Approximation zum Zeitpunkt t auf den Wert zum Zeitpunkt $t-1$ setzen (blaue Kurve). Es wird also keine Änderung während des Zeitraums angenommen. Wie in der Abbildung zu sehen ist der Fehler, besonders in Bereichen mit eher flachem Verlauf, relativ gering, vor allem wenn, wie in der Praxis oftmals der Fall, zusätzlich ein Rauschen auf

den Datenpunkten liegt. Auch wenn dies sicherlich keine optimale Approximation ist, so bietet es doch eine einfach zu findende Lösung für das Verfahren, werden doch auf diese Weise die anderen Abhängigkeiten nicht mehr so dringend benötigt und es wird, wenn auch kein optimaler, so doch ein relativ guter Approximationsfehler erreicht.
Auch die Tests in Kapitel 5.2.1 zeigen, dass diese Tatsache das Verfahren von der Notwendigkeit befreit, alle voneinander abhängigen Problemgrößen in eine Partition zu verlegen. Oftmals besteht die Lösung aus einigen Partitionen mit nur jeweils einer einzigen Problemgröße, meist Größen mit eher flachem Verlauf, welche sich gut durch sich selbst approximieren lassen, während die restlichen Problemgrößen auf die übrigen Partitionen verteilt werden und damit den Großteil der Größen enthalten. Durch diese relativ großen Partitionen, welche dann meist mehr als ein real vorhandenes Teilproblem enthalten, wird sichergestellt, dass alle Abhängigkeiten innerhalb der Partitionen bleiben, während die Partitionen mit jeweils nur einer einzigen Problemgröße lediglich sicherstellen sollen, dass die vorgegebene Anzahl Teilprobleme erreicht wird. Dieses Vorgehen erleichtert es also dem Verfahren, eine Lösung mit einer relativ hohen Fitness zu finden, verringert aber, da die Partitionen mit jeweils einer einzelnen Problemgröße nicht als Teilproblem angesehen werden können und eigentlich falsch zugeordnet wurden, die Anzahl der effektiv gefundenen Teilprobleme.

4.3.3 Problemzerlegung durch Partitionierung Variante 2

Wegen der Probleme bei der Problemzerlegung durch Partitionierung wurde das Verfahren zusätzlich zu einer zweiten Variante abgewandelt. Die Probleme entstanden aufgrund der Tatsache, dass bei der Approximation der Problemgrößen das neuronale Netz die vergangenen Werte dieser Größen selbst als Input bekam. Dies machte die Approximation für das Netz zu einfach, da theoretisch nur der aktuelle Wert gleich dem letzten bekannten Wert der Größe gesetzt werden müsste, um eine relativ gute Approximation zu erhalten.

Die zweite Variante der Problemzerlegung durch Partitionierung zielt deshalb darauf ab, diese Tatsache zu vermeiden und sicherzustellen, dass eine Problemgröße aus-

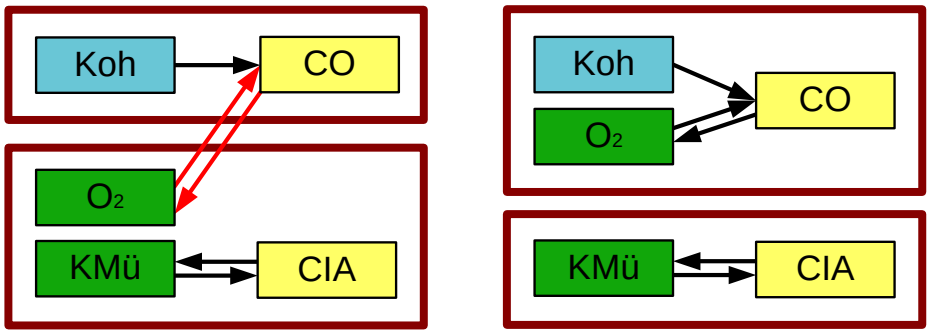

Abbildung 4.17: Illustration der Problemzerlegung durch Partitionierung Variante 2 für das Beispielproblem. Gezeigt sind zwei unterschiedliche Partitionierungen. In der linken Abbildung ist eine Partitionierung zu sehen, bei der Abhängigkeiten über die Partitionsgrenzen hinweg gehen (rot), was sich in einem schlechten Approximationsfehler niederschlagen würde. Rechts ist die korrekte Partitionierung zu sehen.

schließlich Werte anderer Problemgrößen als Input bekommt. Sie kann als eine Art Mischung zwischen der ersten Variante der Problemzerlegung durch Partitionierung und der Problemzerlegung durch Input Pruning angesehen werden, bei welcher jede Problemgröße durch ein eigenes neuronales Netz approximiert wurde, welches die Werte dieser Problemgröße selbst außerdem nicht als Eingangsgröße bekam.
Entsprechend werden, ähnlich wie bei der Problemzerlegung durch Partitionierung Variante 1, die Problemgrößen in mehrere Partitionen zerlegt, um eine Problemzerlegung zu erzwingen. Die Partitionen entsprechen dabei wie gewohnt den Teilproblemen. In Abbildung 4.17 werden zwei mögliche Partitionierungen für das Beispielproblem gezeigt. Links ist dabei eine inkorrekte Partitionierung zu sehen, da Abhängigkeiten über Teilproblemgrenzen hinweg reichen, während rechts die korrekte Partitionierung abgebildet ist, welche bei diesem Verfahren durch die Evolution ermittelt werden soll. Zur Bewertung der Fitness der Partitionierung werden auch hier die Problemgrößen innerhalb der Partitionierung durch neuronale Netze approximiert. Dafür wird allerdings jede Mess- und Regelgröße für den Zeitpunkt t durch ein eigenes neuronales Netz approximiert, welches die Werte aller anderen Problemgrößen der Partition, ausgenommen der Werte der aktuellen Mess- oder Regelgröße selbst, als

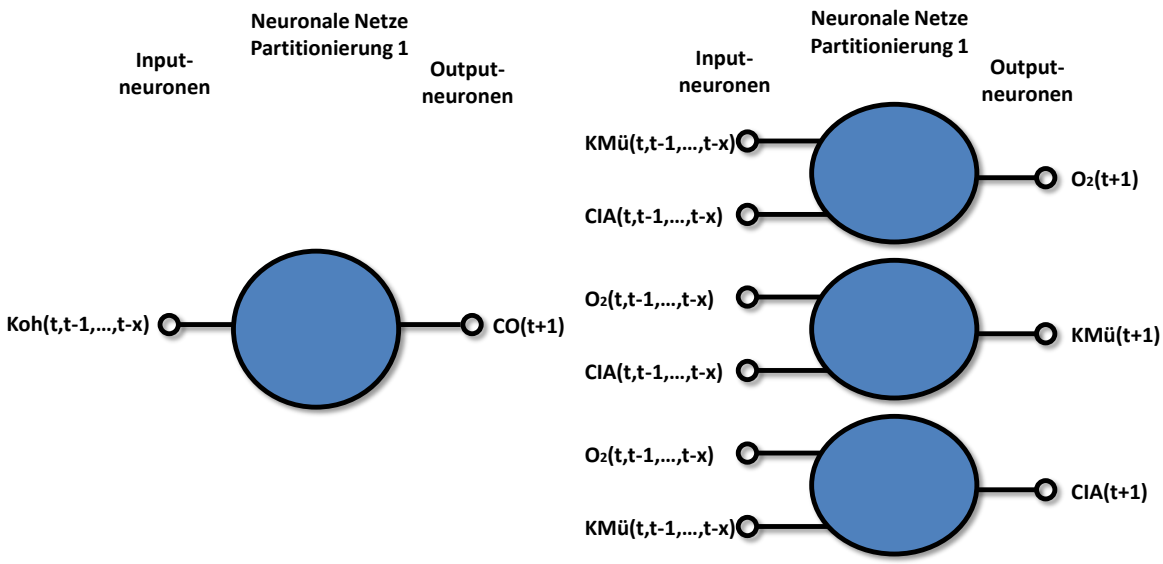

Abbildung 4.18: Veranschaulichung des Trainings der neuronalen Netze bei der Problemzerlegung durch Partitionierung Variante 2. Auf jede Mess- und Regelgröße wird ein eigenes neuronales Netz trainiert, mit jeweils allen anderen Problemgrößen der Partition als Eingangsgrößen.

Inputs bekommt. Auf diese Weise wird sichergestellt, dass zur Approximation einer Problemgröße nicht die Werte dieser Problemgröße selbst herangezogen, sondern ausschließlich Abhängigkeiten zu anderen Problemgrößen berücksichtigt werden. So wird die oben beschriebene triviale Lösung vermieden.

In Abbildung 4.18 wird die Struktur der neuronalen Netze für die Partitionierung aus Abbilding 4.17 links veranschaulicht. Ist die Partitionierung dabei inkorrekt, fehlen zur Approximation benötigte Abhängigkeiten und der Approximationsfehler wird schlechter. So fehlen im Beispiel zur Approximation des Kohlenmonoxidwertes (CO) die Werte für die Luftzufuhr (O_2) und umgekehrt. Je mehr der Abhängigkeiten die Teilproblemgrenzen überschreiten, desto mehr Informationen fehlen den neuronalen Netzen zur Approximation der Problemgrößen und desto schlechter ist der Gesamtapproximationsfehler. Da dieser in der Evolution als Fitnesswert verwendet wird, steht zu erwarten, dass schlechte Partitionierungen aussterben und im Gegenzug gute

Partitionierungen, also Partitionierungen mit weniger Abhängigkeiten, welche über Teilproblemgrenzen hinausreichen, sich durchsetzen, bis letztendlich die korrekte, optimale Partitionierung gefunden wurde.

Das gesamte Verfahren ist in Algorithmus 4.5 als Pseudocode dargestellt. Es unterscheidet sich dabei nur im Detail des Trainings der neuronalen Netze auf die Problemgrößen von der Variante 1 der Problemzerlegung durch Partitionierung. Anfangs werden μ zufällig ausgewürfelte Partitionierungen erstellt, welche die Startpopulation bilden. Bei allen Individuen trainiert man anschließend für jede Partition und jede Mess- und Regelgröße innerhalb der Partitionen ein neuronales Netz mit dem Levenberg-Marquardt Lernverfahren, welches alle anderen Problemgrößen aus der jeweiligen Partition als Inputs bekommt. Nach der Anwendung des Network Prunings durch das Optimal Brain Surgeon Verfahren kann dann die Fitness auf Basis des Root Mean Squared Errors berechnet werden. Auch wenn die neuronalen Netze dieses Verfahrens nur eine Outputgröße besitzen, sollte Formel 4.4 verwendet werden, um die Outputs der Netze der Individuen jeweils zu einem Gesamt-RMSE zusammenzufassen. Mit diesem Root Mean Squared Error, welcher als Fitnesswert fungiert, können die Individuen absteigend nach ihrer Fitness sortiert werden. Anschließend werden sie evolviert, indem zuerst λ Individuen selektiert werden (Kapitel 3.2.1), und danach ebenso viele Nachkommen aus diesen Individuen durch Mutation und Rekombination erzeugt werden (Kapitel 3.2.2 und 3.2.3). Nach dem erneuten Training nach oben beschriebenem Schema, Pruning und Fitnessberechnung kann die Population anschließend wieder absteigend nach ihrer Fitness sortiert und, um die Größe der Population konstant zu halten, die Individuen mit der geringsten Fitness gelöscht werden. Dieser Vorgang wird bis zum Abbruchkriterium, beispielsweise einer vorher festgelegten Anzahl an Generationen, wiederholt und letztendlich die Partitionierung des Individuums mit der höchsten Fitness als Ergebnis ausgegeben.

Die Vorteile dieser zweiten Variante der Problemzerlegung durch Partitionierung sind zum Einen, dass es, wie auch bei der ersten Variante, auf jeden Fall zu einer Problemzerlegung kommt, wobei das Problem in genau die vorher festgelegte Anzahl an

Algorithmus 4.5 Problemzerlegung durch Partitionierung Variante 2

Input μ, λ ▷ μ: Größe der Population ,λ: Anzahl Nachkommen
Input $x[1..N_x]$ ▷ x: Wert der vergangenen Problemgrößen
Input $y[1..N_y]$ ▷ y: Werte der aktuellen Mess- und Regelgrößen
Input $nmax$, N_P ▷ $nmax$: max. Anzahl Iterationen, N_P: Anzahl Teilprobleme
for $i = 1 \rightarrow \mu$ **do** ▷ für alle Individuen der Population
  $P_i[1..N_P] \leftarrow$ INITIALISIEREPOPULATION(x, y) ▷ zufällige Initialisierung
  ▷ $P_i(j)$: Liste mit Problemgrößen in Partition j und Individuum i
  for $j = 1 \rightarrow N_P$ **do** ▷ für alle Partitionen
    for all $k \in P_i(j)$ **do** ▷ für alle Problemgrößen k in der Partition
      $N_i(j, k) \leftarrow$ TRAIN$(x(P_i(j)), y(k))$ ▷ train. Netz mit Alg. 4.2 auf Größe k
    end for
  end for
  $Fitness_i \leftarrow RMSE_i$ ▷ berechne Fitness über Formel 4.4
end for
$(N, P) \leftarrow$ SORT$(Fitness)$ ▷ sortiere Individuen der Pop. absteigend nach Fitness
$n \leftarrow 0$ ▷ n: aktuelle Anzahl Iterationen der Evolution
repeat
  $n \leftarrow n + 1$
  for $i = 1 \rightarrow \lambda$ **do**
    $k \leftarrow$ SELEKTION() ▷ selektiere Individuum k mit Wkt. aus Formel 3.1
    $P_{\mu+i} \leftarrow$ REPRODUKTION(P_k) ▷ Mutation/Rekombination der
    ▷ Partitionen nach Kapitel 3.2.2 und 3.2.3
    for $j = 1 \rightarrow N_P$ **do** ▷ für alle Partitionen
      for all $k \in P_{\mu+i}(j)$ **do** ▷ für alle Problemgrößen k in der Partition
        $N_{\mu+i}(j, k) \leftarrow$ TRAIN$(x(P_{\mu+i}(j)), y(k))$ ▷ trainiere Netz auf Größe k
      end for
    end for
    $Fitness_{\mu+i} \leftarrow RMSE_{\mu+i}$ ▷ berechne Fitness über Formel 4.4
  end for

$(N, P) \leftarrow \text{SORT}(Fitness)$	▷ sortiere Ind. der Pop. absteigend nach Fitness
$P_i[\mu+1, .., \mu+\lambda] \leftarrow \{\}$	▷ lösche Individuen $\mu+1$ bis $\mu+\lambda$
until $n \geq nmax$	▷ bis Abbruchkriterium erreicht
return P_1	

Teilproblemen zerlegt wird. Allerdings ist auch hier eine gute Abschätzung der tatsächlich vorhandenen Teilprobleme vonnöten. Eine Möglichkeit dafür wird in Kapitel 4.4.3 vorgestellt. Zum Anderen wird hier der Fall vermieden, dass, durch die Verwendung mehrerer neuronaler Netze innerhalb jeder Partition, zur Approximation einer Größe die Werte dieser Größe selbst herangezogen werden, sodass triviale Lösungen entstehen, bei welchen beispielsweise die Approximation zum Zeitpunkt t gleich dem Wert der Größe zum Zeitpunkt $t-1$ gesetzt wird, was zu fehlerhaften Partitionierungen und damit zu einer fehlerhaften Problemzerlegung führen kann. Auch werden bei diesem Verfahren in der Grundvariante zwar die Abhängigkeiten innerhalb der Teilprobleme nicht automatisch mit ermittelt, jedoch ist dies durch eine kleine Modifikation relativ leicht zu bewerkstelligen, da auf jede Größe bereits ein eigenes neuronales Netz trainiert wird. Durch ein einfaches Input Pruning jedes dieser neuronalen Netze lassen sich für die Approximation dieser Größe unwichtige Abhängigkeiten ausschließen, sodass nur die relevanten übrig bleiben.

Ein Nachteil ist allerdings, dass zur hinreichend guten Approximation einiger Problemgrößen die vergangenen Werte dieser Größe vonnöten sein können. Auch wenn die Hinzunahme dieser bei der Problemzerlegung und der Partitionierung zu Problemen führen kann, können sie dennoch in einigen Fällen zur optimalen Abschätzung notwendig sein. Die Lösung dieses Dilemmas besteht entweder darin, die Variante 1 der Problemzerlegung durch Partitionierung zu verwenden, trotz der eventuell damit verbundenen Probleme, oder die Problemzerlegung und die Abschätzung so gut wie möglich durch die restlichen Abhängigkeiten vorzunehmen. Denn auch wenn die Abschätzung der Größe auf diese Weise nicht optimal ist, kann sie eventuell dennoch zur Problemzerlegung genutzt werden, da alle Individuen der Evolution unter dem gleichen Mangel an Informationen für die Approximation leiden. Die Abschätzung wird in diesem Fall „so gut wie möglich“ durchgeführt. Leider ist es dann jedoch nicht mehr

empfehlenswert die trainierten neuronalen Netze zur Regelung zu verwenden, da zu erwarten steht, dass das Training ohne die benötigten vergangenen Werte der Problemgröße suboptimale Ergebnisse hervorbrachte.

4.4 Erweiterungen der Verfahren zur Problemzerlegung

Bisher wurden im Kapitel 4.3 nur die grundlegenden Verfahren zur Problemzerlegung vorgestellt, welche erst einmal die dahinterliegende Grundidee illustrieren sollten. Für eine Anwendung in der Praxis müssen jedoch noch einige weitere Aspekte beachtet werden, welche bisher vernachlässigt wurden. Dafür werden im Folgenden Möglichkeiten für die Time Delay Estimation, also das Ermitteln der Totzeit zwischen der zu approximierenden und den davon abhängigen Größen vorgestellt und es wird ein Verfahren beschrieben, welches die Anzahl der Teilprobleme, welche bei den beiden Algorithmen zur Problemzerlegung durch Partitionierung noch vorgegeben werden musste, automatisch ermittelt. Schließlich wird erläutert, wie eine nicht exklusive Problemzerlegung vorgenommen werden kann.

4.4.1 Time Delay Estimation

Ein wichtiger Aspekt bei der Problemzerlegung ist die Time Delay Estimation (TDE). Dabei wird die Totzeit, also die Zeitverzögerung ermittelt, mit welcher die abhängigen Größen auf die aktuelle Problemgröße einwirken. Dies ist in Abbildung 4.19 am Beispielproblem demonstriert. Abgebildet ist der Verlauf des Kohlenmonoxidausstoßes des Kraftwerkes (CO) und der beiden darauf Einluss nehmenden Größen der Kohlezufuhr (Koh) und der Luftzufuhr (O_2). Die Kohlezufuhr wird während des betrachteten Zeitraums konstant gehalten, während gleich zu Beginn die Luftzufuhr schlagartig ansteigt. Da die Verbrennungsprozesse innerhalb des Kraftwerkes aber erst einige Zeit benötigen, bleibt der Kohlenmonoxidausstoß vorerst gleich und steigt erst nach einiger Zeit T ebenfalls an. Diese Zeitverzägerung T wird auch Totzeit genannt.

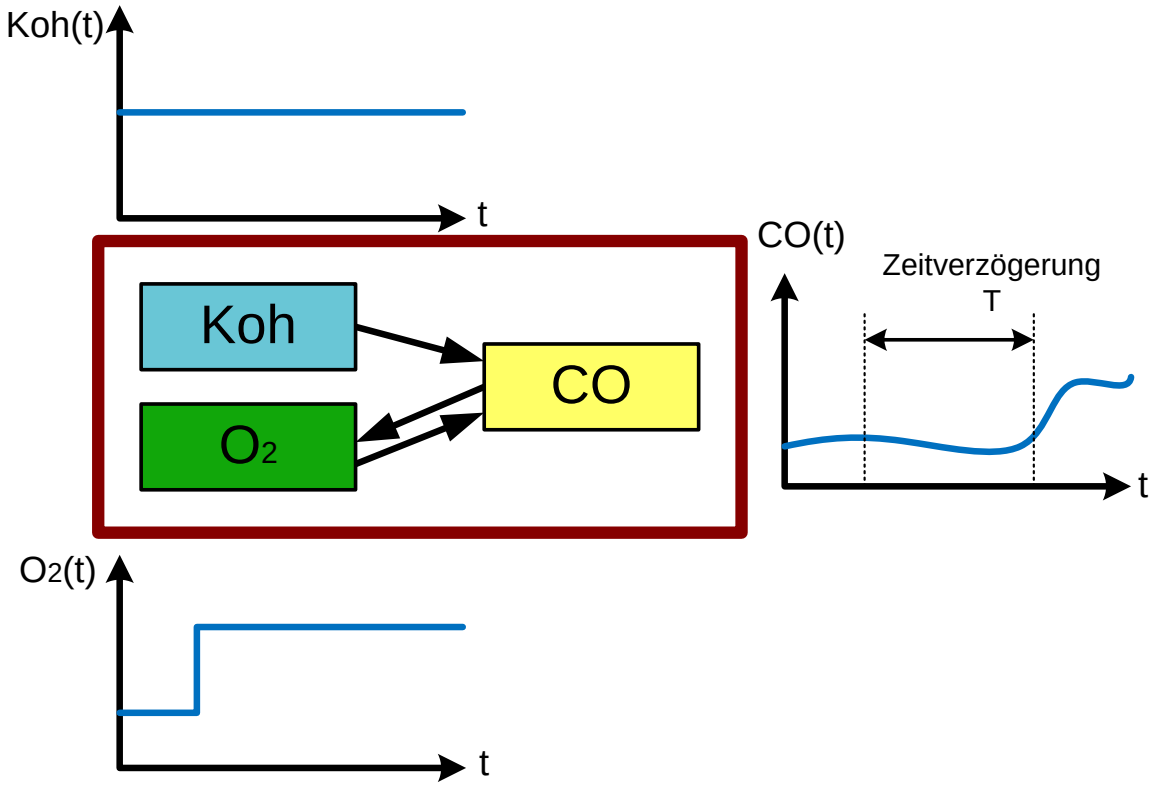

Abbildung 4.19: Illustration der Notwendigkeit der Time Delay Estimation am Beispielproblem. Die zeitlichen Abläufe der Werte der Problemgrößen sind als Kurve (blau) dargestellt. Während die Kohlezufuhr (Koh) konstant gehalten wird, steigt die Luftzufuhr (O_2) nach kurzer Zeit sprungartig an. Auch der Kohlenmonoxidausstoß (CO) verändert sich, allerdings erst einige Zeit später. Die Zeitverzögerung zwischen Änderung der Eingangsgröße und der Wirkung auf die Ausgangsgröße wird Totzeit T genannt.

Ein gewöhnliches Multilayerperzeptron dagegen benötigt die Reaktion auf eine Änderung idealerweise sofort, da dort keine zeitliche Vorgeschichte berücksichtigt wird und nur die aktuellen Wertepaare als kausal zusammengehörig betrachtet werden. Bisher wurden in dieser Arbeit deshalb die Eingangsgrößen zu sämtlichen vergangenen Zeitpunkten t, $t-1$, ..., $t-x$ verwendet. In der Praxis ist es aber natürlich nicht möglich, die Werte zu allen vergangenen Zeitpunkten zu nutzen. Theoretisch wäre es denkbar, die Auswahl auf ein vorher festgelegtes, sinnvolles Zeitfenster zu begrenzen, und diese entweder z.B. mit einem Time Delay Neural Network zu verwenden, oder durch Input Pruning die Größen zu nicht benötigten Zeitpunkten wieder zu entfernen. Jedoch ist die damit verbundene Vervielfachung der Eingangsgrößen für das neuronale Netz mit einer sehr hohen Trainingszeit und ebenfalls meist mit einer eher schlechteren Fehlerrate verbunden und die damit einhergehende Vervielfachung der freien Parameter (Gewichte) und Erhöhung der Komplexität für den Lernvorgang sind dem Anliegen dieser Arbeit entgegengesetzt, weshalb hier auf eine andere Alternative zurückgegriffen wird. Dafür wird die Totzeit zunächst über ein Verfahren abgeschätzt, und dann aus dem Input des neuronalen Netzes herausgerechnet. In Abbildung 4.20 ist dies grafisch veranschaulicht. Für die fünf Eingangsgrößen wurde hier die Totzeit berechnet und das neuronale Netz bekommt die Daten mit dieser Totzeit zeitverzögert als Inputs. Die restlichen Daten werden dem Netz vorenthalten, da sie keine weiteren relevanten Informationen enthalten.

Es gibt mehrere Verfahren, welche die Totzeit berechnen können. Als besonders Laufzeiteffizient und einfach in die problemzerlegenden Verfahren zu integrieren hat sich dabei die Time Delay Estimation durch Mutual Information erwiesen, welche deshalb im Folgenden beschrieben wird und bei den Tests in Kapitel 5 Verwendung gefunden hat.

4.4.2 Time Delay Estimation durch Mutual Information

Die Mutual Information, welche in Kapitel 3.4 vorgestellt wurde, ist, wie dort bereits beschrieben, ein Maß für die Abhängigkeit von zwei Zufallsgrößen. Je höher die Mutual Information zwischen zwei Größen, desto größer die Abhängigkeit und desto mehr

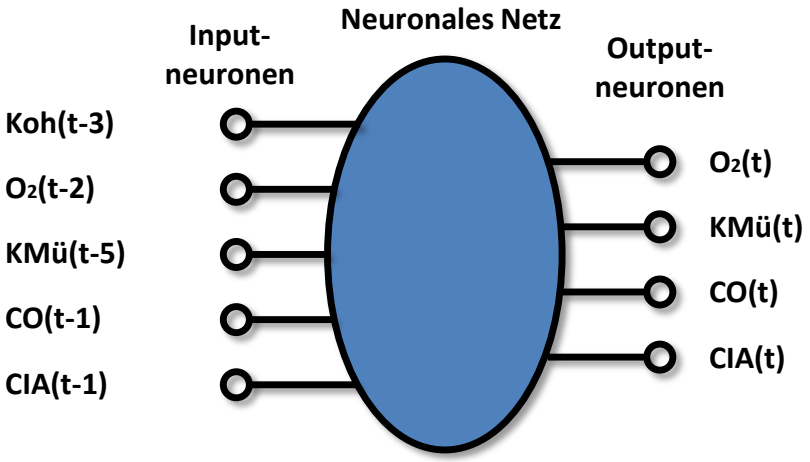

Abbildung 4.20: Veranschaulichung des Herausrechnens der Zeitverzögerung für das neuronale Netz am Beispielproblem.

Informationen über die eine Größe steckt in der anderen. Damit bietet sie sich auch zur Berechnung der Totzeiten der Problemgrößen bei der Problemzerlegung an, wie dies in [Wen und Wan, 2009] beschrieben ist. Diese kann ermittelt werden, indem die Mutual Information zwischen verschiedenen zeitverzögerten Inputs und den Teachwerten des neuronalen Netzes berechnet wird. Die Mutual Information wird dabei maximal, wenn die Zeitverzögerung der Totzeit entspricht. Oder mathematisch ausgedrückt:

$$\hat{T}_i = \max_T \left(I(x_i(t-T); y(t)) \right) \tag{4.38}$$

Dabei stellt I die Mutual Information dar, T ist die Zeitverzögerung und $\hat{T}_i$ die abgeschätzte Totzeit für Eingangsgröße i.
Auf diese Weise wird die Totzeit für die Problemgröße bestimmt, bei welcher der maximale Zusammenhang zur Outputgröße besteht. Da sich die Totzeit außerdem von Größe zu Größe ändern kann, wird dieses Verfahren für alle Kombinationen von denkbaren Input- und Outputgrößen wiederholt.

Algorithmus 4.6 beschreibt das gesamte Verfahren als Pseudocode. Zu Beginn wird eine Matrix $Delay$ mit den Dimensionen $AnzahlInputs \times AnzahlOutputs$ initialisiert,

Algorithmus 4.6 Time Delay Estimation durch Mutual Information

Input $x[1..N_x]$ ▷ x: Wert der vergangenen Problemgrößen
Input $y[1..N_y]$ ▷ y: Werte der aktuellen Mess- und Regelgrößen
Input T_{max} ▷ T_{max}: Schranke für maximal zulässige Zeitverzögerung
$Delay[1..N_x, 1..N_y] = (T_{max} + 1)_{N_x,N_y}$ ▷ Matrix mit Delays für Input/Output-Paare
for $i = 1 \rightarrow N_x$ **do** ▷ für alle Inputs
  for $j = 1 \rightarrow N_y$ **do** ▷ für alle Outputs
    for $T = 1 \rightarrow T_{max}$ **do** ▷ für alle zulässigen Zeitverzögerungen
      $MI_{x_i,y_j}(T) = I(x_i(t-T); y_j(t))$ ▷ Mutual Information Berechnung
      ▷ mit Algorithmus 3.2
    end for
    $Delay[i,j] = \max_T \left(MI_{x_i,y_j}(T)\right)$
  end for
end for
return $Delay$

welche jedem Input-Output-Paar einen Wert für die Zeitverzögerung zuweist. Anschließend wird für jeden Input, jeden Output und jeder, von vorher festgelegten Schranken begrenzten, Zeitverzögerung die Mutual Information mit Algorithmus 3.2 berechnet. Jedem Input-Output-Paar wird dann die Zeitverzögerung zugewiesen, bei welchem die Mutual Information über alle möglichen Zeitverzögerungen maximal wird. Als Ergebnis wird schließlich die Matrix *Delay*, welche die Totzeiten enthält, ausgegeben. Mithilfe dieser Totzeiten können die Inputs der neuronalen Netze entsprechend Abbildung 4.20 auf die wesentlichen Inputs, mit der korrekten Totzeit, beschränkt werden.

Anzumerken ist, dass mit dieser Methode der maximierten Mutual Information nicht nur die Totzeiten, sondern auch beispielsweise eventuell vorhandene Tiefpässe berechnet werden können, wenn also der Output nicht nur von einem zeitverzögerten Input, sondern beispielsweise vom Mittelwert mehrerer vergangener Inputs abhängig ist. In diesem Fall muss der Algorithmus nur noch dahin gehend angepasst werden, dass die Mutual Information bei jedem Input-Output-Paar zusätzlich zu jeder denkbaren Zeit-

verzögerung noch zu jeder denkbaren Mittelung über die letzten n Werte berechnet und auch zusätzlich über diese maximiert werden.

4.4.3 Berechnung der Anzahl der Teilprobleme

Den Verfahren zur Problemzerlegung in Kapitel 4.3 musste bisher die Anzahl der Teilprobleme vorgegeben werden, mit der Ausnahme der Problemzerlegung durch Input Pruning, welche dies bereits von Grund auf automatisch bewerkstelligte. In der Praxis jedoch ist die Anzahl der Teilprobleme, genau wie die Problemzerlegung selbst, aufgrund mangelnden Wissens über den zugrundeliegenden Prozess, meist nicht bekannt.
Oftmals genügt eine gute Schätzung dieser Anzahl an Teilproblemen, denn selbst wenn die tatsächlich vorhandene Anzahl dabei nicht getroffen wird, kann das Ergebnis oftmals dennoch zufriedenstellend sein. Ist die geschätzte Anzahl dabei zu gering, wird das Problem in eine niedrigere Anzahl Teilprobleme zerlegt. In diesem Fall würden sich dadurch stellenweise mehrere reale Teilprobleme eine Partition teilen. Die Zerlegung bleibt also unvollständig. Aber auch ein unvollständig zerlegtes Problem bringt bereits die in Kapitel 2.4 genannten Vorteile mit sich, wenn auch nicht im größtmöglichen Maße. Kritischer ist es, wenn die Schätzung der Anzahl der Teilprobleme zu hoch angesetzt wird. In diesem Fall wird das Problem in zu viele Partitionen zerlegt und ein oder mehrere real vorhandene Teilprobleme werden auseinandergerissen. Die Evolution ist jedoch so gestaltet, dass die Abhängigkeiten zwischen den auseinandergerissenen Teilproblemen möglichst gering sind, sodass die Approximation auch ohne diese Abhängigkeiten eine einigermaßen gute Fehlerrate aufweisen kann.

Um jedoch die optimale Lösung zu erreichen, ist es notwendig, die genaue Anzahl real vorhandener Teilprobleme zu kennen. Um diese zu ermitteln, bietet sich erneut ein evolutionärer Algorithmus an, durch welchen bereits die Partitionierung ermittelt wird. In Abbildung 4.21 ist die Mutation am Beispielproblem mit drei Partitionen gezeigt, so, wie sie bisher bei den Verfahren zur Problemzerlegung durch Partitio-

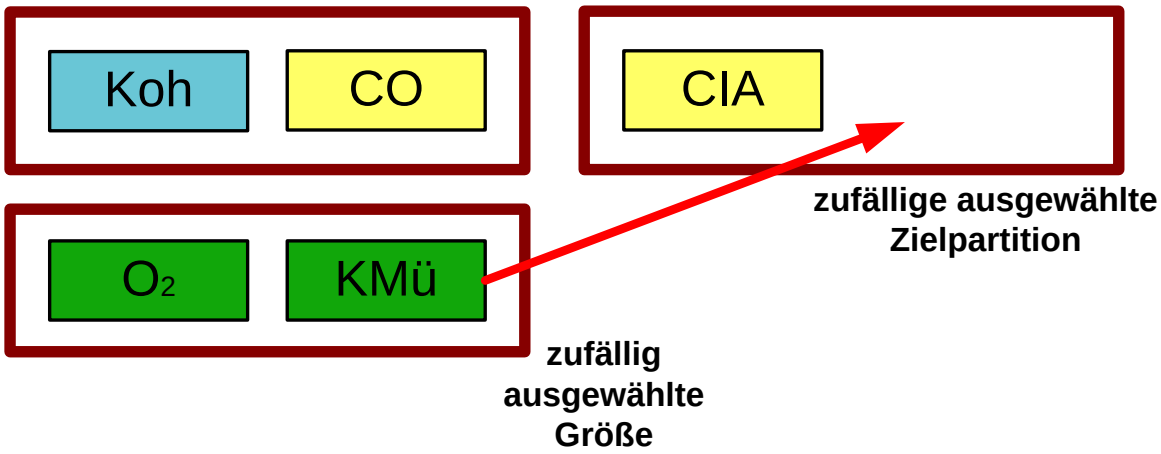

Abbildung 4.21: Darstellung der Mutation der Partitionen zur Problemzerlegung. Aus der gesamten Menge der Problemgrößen wird eine ausgewählt und in eine zufällige andere Partition verlegt.

nierung in beiden Varianten vorgenommen wird. Dafür werden eine oder mehrere Problemgrößen zufällig ausgewählt und in eine zufällige andere Partition verlegt. Dabei bleibt bisher die vorgegebene Anzahl der Teilprobleme konstant, d.h. es wird sichergestellt, dass durch eine solche Mutation keine leere Partition entsteht und es kann, da eine Mutation in eine leere, noch nicht vorhandene Partition nicht möglich ist, keine neue Partition entstehen.

Für ein automatisches Ermitteln der Anzahl der Teilprobleme können diese Beschränkungen theoretisch weggelassen werden, d.h. es soll einerseits möglich gemacht werden, dass durch Mutation eine leere Partition entsteht, welche anschließend gelöscht wird, wodurch sich also die Anzahl der Partitionen verringert, und andererseits, dass Mutation nicht nur in bereits vorhandene Partitionen, sondern ebenfalls in eine neue Partition möglich ist. Dies wird in Abbildung 4.22 grafisch anhand des Beispielproblems verdeutlicht. Zusätzlich zu den vorhandenen Partitionen wird dabei während der Evolution eine leere Partition bereitgehalten. Durch Mutation (rote Pfeile) ist es möglich, Problemgrößen nicht nur in die vorhandenen, bereits belegten Partitionen, sondern auch in diese leere Partition zu verlegen und so die Anzahl

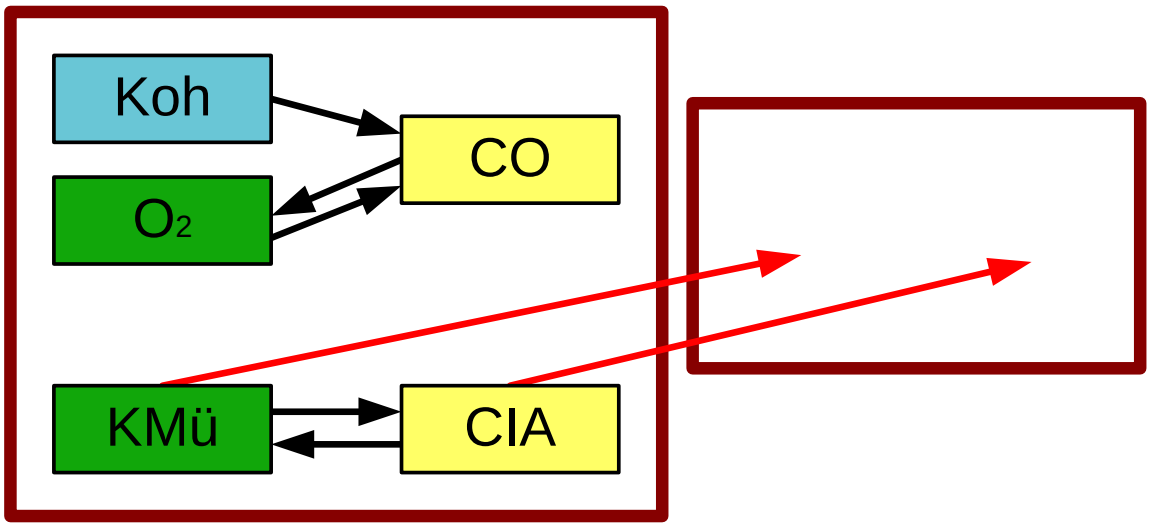

Abbildung 4.22: Darstellung der Mutation in eine neue Partition am Beispielproblem. Die Problemgrößen können durch Mutation (rote Pfeile) in eine neue, noch nicht vorhandene Partition verlegt werden. Die Wahrscheinlichkeit für eine erfolgreiche Mutation ist dabei jedoch eher gering.

Partitionen zu erhöhen.

In der Praxis ist Letzteres jedoch problematisch, da die Wahrscheinlichkeit, durch eine solche Mutation eine sinnvolle neue Lösung zu erlangen, relativ gering ist. In Abbildung 4.22 liegen vor Durchführung der Mutation alle Problemgrößen in einer Partition. Also auch, wenn das Problem noch nicht zerlegt ist, so befinden sich doch alle Abhängigkeiten innerhalb der Partition. Damit ist eine relativ gute Approximation der Problemgrößen möglich, was zu einer guten Fitness in der Evolution führt. Soll nun im Rahmen der Problemzerlegung ein Teilproblem durch die zufällige Mutation in eine neue Partition verlegt werden, ist es notwendig, dies für alle Problemgrößen eines der Teilprobleme zugleich, also mit einer einzigen Mutation zu tun. Wird dies nicht getan und nur ein Teil der Problemgrößen in einem Schritt verlegt, zerreißt dies die Abhängigkeiten. Als Folge davon ist die Approximation der Problemgrößen, wegen der mangelnden Informationen, nur mit einer eher schlechten Fehlerrate möglich. Dies führt zu einer niedrigen Fitness der Partitionierung und damit höchstwahrscheinlich zu einem raschen Aussterben dieser Lösung im Rahmen der Evolution.

Aufgrund dieser Einschränkung soll hier auf die Mutation in neue Partitionen und allgemein die Erhöhung der Partitionsanzahl verzichtet werden und die automatische Ermittlung der Anzahl der Teilprobleme wird ausschließlich über die Verringerung der Anzahl der Partitionen geschehen. Dazu ist es notwendig, das Verfahren mit einer deutlich höheren Anzahl Teilprobleme zu starten, als tatsächlich vorhanden sind, beispielsweise mit etwa doppelt so vielen. Durch Mutation werden dann Problemgrößen in andere Partitionen verschoben und es entstehen gelegentlich leere Partitionen, welche gelöscht werden. Da die Anzahl der Partitionen anfangs weit höher ist als die der tatsächlich vorhandenen Teilprobleme, ist es nicht zu vermeiden, dass Abhängigkeiten auch zwischen Problemgrößen verschiedener Partitionen bestehen, was sich negativ auf deren Fitness auswirkt. Verringert sich durch die Mutation aber die Anzahl der Partitionen, verringert sich damit gewöhnlich auch die Menge dieser fehlenden Abhängigkeiten, die Fitness dieser neuen Partition steigt und sie setzen sich in der Evolution gegenüber den anderen Individuen durch. Verringert sich die Anzahl der Partitionen dagegen zu weit, entstehen beim Training der neuronalen Netze zu große neuronale Netze mit zu vielen Eingangsgrößen, zu vielen freien Parametern und zu hoher Komplexität, was sich ebenfalls negativ auf die Fitness dieser Partitionierung auswirkt, sodass man annehmen kann, dass sich die Partitionierung der Problemgrößen in die Anzahl der Partitionen, welcher der Anzahl der tatsächlichen Teilprobleme entspricht, letztendlich durchsetzt. Schematisch ist dies in Abbildung 4.23 dargestellt.

Die gesamte Problemzerlegung mit automatischer Ermittlung der Anzahl der Teilprobleme ist in Algorithmus 4.7 als Pseudocode zusammengefasst. Der Algorithmus lässt sich auf beide Varianten der Problemzerlegung durch Partitionierung übertragen, welche sich im Wesentlichen nur im Training der neuronalen Netze unterscheiden. Es wurde hier eine Variable hinzugefügt, welche die Anzahl der Partitionen der einzelnen Individuen speichert. Sie wird mit einer vorgegebenen initialen Anzahl Partitionen initialisiert und im Laufe der Evolution immer wieder aktualisiert. Außerdem wird nach einer Reproduktion geprüft, ob sich durch Mutation die Anzahl der Partitionen ver-

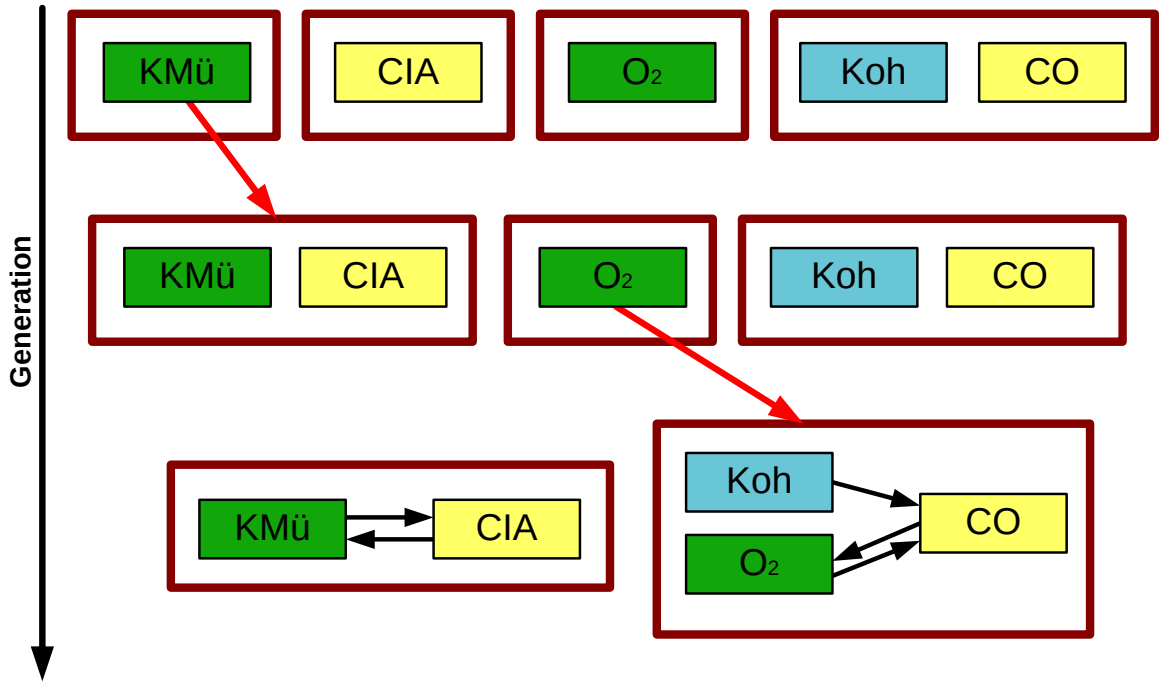

Abbildung 4.23: Illustration der automatischen Berechnung der Anzahl der Teilprobleme durch Evolution am Beispielproblem. Zu Beginn (oben) wird mit deutlich mehreren Partitionen gestartet als Teilprobleme vorhanden sind. Im Laufe der Evolution verschwinden Teilprobleme, weil ihre Problemgrößen in andere Partitionen mutiert wurden (rote Pfeile), bis die optimale Partitionsanzahl und Partitionierung gefunden wurde (unten).

ringert hat. Falls ja, werden die neuronalen Netze der nun leeren Partition(en) gelöscht.

Der Nachteil dieses Verfahrens ist allerdings, dass ähnlich der Problemzerlegung durch Input Pruning die Problemzerlegung nicht mehr erzwungen wird. Bei dieser konnte es beispielsweise vorkommen, dass nach der Evolution am Ende ein einziges Teilproblem entstand, also effektiv keine Problemzerlegung mehr stattfand. Um dies zu umgehen, wurden die beiden Verfahren der Problemzerlegung durch Partitionierung entworfen, welche durch die Partitionierung eine Zerlegung in genau die vorgegebene Anzahl Teilprobleme erzwang. Indem dem Algorithmus allerdings die Wahl der Anzahl der Teilprobleme wieder selbst überlassen wurde, besteht hier die Gefahr eines

Rückschrittes. Denn auch wenn theoretisch eine Zerlegung in genau die Anzahl Partitionen, welche der Anzahl der realen Teilprobleme entspricht, den geringsten Approximationsfehler der neuronalen Netze und damit die beste Fitness verspricht, so ist eine solche Partitionierung für den Algorithmus doch recht schwierig zu finden. Es besteht hier die Gefahr, dass im Laufe der Evolution die Anzahl der Teilprobleme zu schnell verringert wird, dabei die optimale Lösung nicht gefunden wurde und auch später, da eine Erhöhung der Anzahl der Partitionen nicht vorgesehen ist, keine Möglichkeit mehr dazu besteht.

Algorithmus 4.7 Ermittlung der Anzahl Teilprobleme bei der Problemzerlegung durch Partitionierung

Input μ, λ ▷ μ: Größe der Population ,λ: Anzahl Nachkommen
Input $x[1..N_x]$ ▷ x: Wert der vergangenen Problemgrößen
Input $y[1..N_y]$ ▷ y: Werte der aktuellen Mess- und Regelgrößen
Input $nmax$, $N_{P_{init}}$ ▷ $nmax$: max. Anzahl Iterationen, $N_{P_{init}}$: initiale Anz. Teilpr.
for $i = 1 \rightarrow \mu$ **do** ▷ für alle Individuen der Population
  $N_P(i) \leftarrow N_{P_{init}}$ ▷ Anzahl Teilprobleme für Individuum i
  $P_i[1..N_P] \leftarrow$ INITIALISIEREPOPULATION(x, y) ▷ zufällige Initialisierung
  ▷ $P_i(j)$: Liste mit Problemgrößen in Partition j und Individuum i
  for $j = 1 \rightarrow N_P(i)$ **do** ▷ für alle Partitionen des Individuums i
    $N_i(j) \leftarrow$ TRAIN() ▷ trainiere Netz entsprechend Variante 1 oder 2
  end for
  $Fitness_i \leftarrow RMSE_i$ ▷ berechne Fitness über Formel 4.4
end for
$(N, P) \leftarrow$ SORT$(Fitness)$ ▷ sortiere Individuen der Pop. absteigend nach Fitness
$n \leftarrow 0$ ▷ n: aktuelle Anzahl Iterationen der Evolution
repeat
  $n \leftarrow n + 1$
  for $i = 1 \rightarrow \lambda$ **do**
    $k \leftarrow$ SELEKTION() ▷ selektiere Individuum k mit Wkt. aus Formel 3.1

```
        P_{μ+i} ← REPRODUKTION(P_k)                ▷ Mutation/Rekombination der
                                                   ▷ Partitionen nach Kapitel 3.2.2 und 3.2.3
        delPart ← ERMITTLEGELOESCHTEPARTITIONEN(P_{μ+i})
                                                   ▷ delPart: Liste mit gelöschten Partitionen
        for part ∈ delPart do                      ▷ für alle gelöschten Partitionen
            N_i(part) ← {}                         ▷ lösche neuronale Netze der Partition
        end for
        N_P(i) ← N_P(i) − length(delPart)          ▷ neue Anzahl Partitionen
        for j = 1 → N_P(i) do                      ▷ für alle Partitionen des Individuums i
            N_{μ+i}(j) ← TRAIN()                   ▷ trainiere Netze entspr. Variante 1 oder 2
        end for
        Fitness_{μ+i} ← RMSE_{μ+i}                 ▷ berechne Fitness über Formel 4.4
    end for
    (N, P) ← SORT(Fitness)                         ▷ sortiere Ind. der Pop. absteigend nach Fitness
    P_i[μ + 1, .., μ + λ] ← {}                     ▷ lösche Individuen μ + 1 bis μ + λ
  until n ≥ nmax                                   ▷ bis Abbruchkriterium erreicht
  return P_1
```

4.4.4 Nicht exklusive Problemzerlegung

Während die bisher gezeigten Probleme, insbesondere das Beispielproblem aus dem Kraftwerk, immer exklusiv trennbar waren, also die Problemgrößen sich nach der in dieser Arbeit aufgestellten Definition der Problemzerlegung in mehrere Partitionen trennen ließen, wobei keine Abhängigkeiten mehr zwischen Problemgrößen verschiedener Partitionen vorhanden sind, ist dies jedoch eher eine idealisierte Form der Problemzerlegung. In der Praxis ist es aber oftmals der Fall, dass eine oder mehrere Problemgrößen auf mehrere oder gar alle Teilprobleme Einfluss besitzen.

Ein Beispiel hierfür ist in Abbildung 4.24 zu sehen. Hier wurde zu dem Beispielproblem aus dem Kraftwerk eine zusätzliche Kenngröße K hinzugefügt, welche auf beide Teilprobleme Einfluss besitzt. Dieses Problem ist nach bisheriger Definition

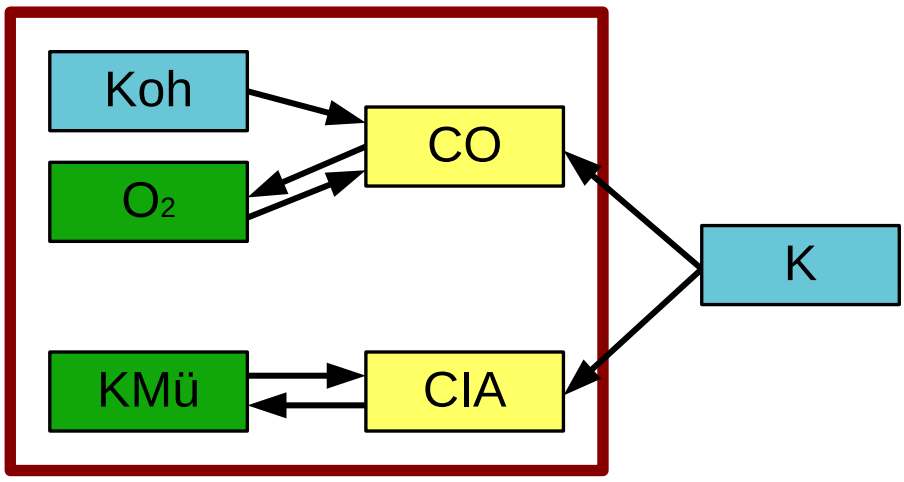

Abbildung 4.24: Beispiel für ein nicht exklusiv trennbares Problem. Dargestellt ist das Beispielproblem mit einer zusätzlichen Kenngröße (K), welche auf beide Teilprobleme Einfluss besitzt.

nicht mehr zerlegbar. Die Anwendung der Problemzerlegung durch Partitionierung mit automatischer Ermittlung der Anzahl der Teilprobleme, welche in Kapitel 4.4.3 vorgestellt wurde, würde höchstwahrscheinlich nur ein einziges Teilproblem liefern, welches alle Problemgrößen enthält. Die Festlegung der Anzahl der Teilprobleme auf eine Anzahl von zwei wiederum führt in diesem Fall zu einer Art „minimalen Schnitt“, bei welchem die Problemgröße K dem Teilproblem zugeordnet werden wird, auf welches sie den größten Einfluss besitzt. Die Abhängigkeit zu dem anderen Teilproblem dagegen geht dabei verloren.

Dennoch könnte das Problem von einer Problemzerlegung profitieren. Denn auch wenn die Problemgröße K auf beide Teilprobleme Einfluss besitzt, ist das bei den restlichen Problemgrößen nicht der Fall, sodass die Approximation der Größen der Teilprobleme ohne die Problemgrößen der jeweils exklusiv zum anderen Teilproblem gehörigen Problemgrößen vonstatten gehen könnte, mitsamt den damit verbundenen Vorteilen, wie z.B. die geringere Komplexität für die neuronalen Netze, die schnellere Trainingszeit und die besseren Lösungen. Zumindest theoretisch ist eine nicht exklusive Problemzerlegung also so lange sinnvoll, wie mindestens eine Problemgröße exklusiv zu einem Teilproblem zugehörig ist, ohne eine Abhängigkeit zu einem der anderen Teilprobleme.

Aufgrund der Natur der Problemzerlegung durch Input Pruning, welche die Abhängigkeiten für jeweils jede Problemgröße einzeln ermittelt, beherrscht diese bereits von vornherein die nicht exklusive Problemzerlegung. Um jedoch mit Hilfe der Problemzerlegung durch Partitionierung in beiden Varianten eine solche nicht exklusive Problemzerlegung durchzuführen, ist eine Erweiterung notwendig. Dafür wird eine Evolution in zwei Stufen verwendet: Zu Beginn wird die Problemzerlegung durch Partitionierung in einer der beiden Varianten in der Grundversion, wie in Kapitel 4.3.2 und 4.3.3 beschrieben, durchgeführt. Durch die Vorgabe der Anzahl der Teilprobleme entstehen daraus, wie oben beschrieben, mehrere Teilprobleme, wobei allerdings noch Abhängigkeiten in einer Art „minimalem Schnitt“ zwischen Problemgrößen mehrerer Partitionen bestehen. In Abbildung 4.25 wird ein denkbares Ergebnis für obiges Beispielproblem gezeigt. Problemgröße K wurde hier dem unteren Teilproblem zugeordnet, zu welchem es die stärkere Abhängigkeit besitzt.

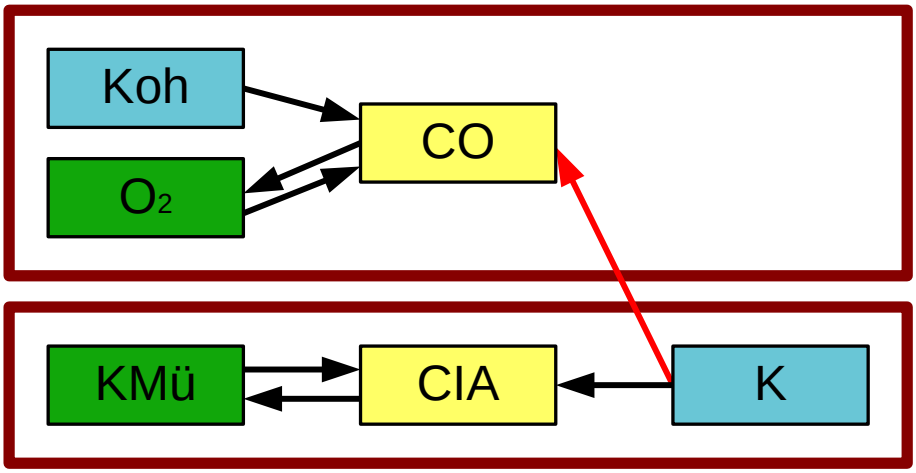

Abbildung 4.25: Darstellung eines „minimalen Schnittes“ am Beispielproblem. Problemgröße K wurde einem der beiden Teilprobleme zugeordnet. Die Abhängigkeit zum anderen Teilproblem (rot) überspringt dabei die Grenzen der beiden Partitionen, ist jedoch eher gering.

Ausgehend davon folgt anschließend ein evolutionäres Input Pruning für die neuronalen Netze der entstandenen Partitionen, ähnlich dem Input Pruning Verfahren aus

Kapitel 4.3.1. Dafür werden den Netzen potenziell auch wieder die Problemgrößen aus den anderen Partitionen zur Verfügung gestellt, sodass sie schließlich wieder Zugang zu den fehlenden Abhängigkeiten erhalten. Ziel ist hier also eher kein Input Pruning im eigentlichen Sinne, also das Entfernen irrelevanter Inputs, sondern die Erweiterung der neuronalen Netze mit den noch fehlenden Inputs, welche durch die erzwungene Partitionierung entstanden sind. Im Folgenden soll der Begriff „Input Pruning“ somit für eine evolutionäre Auswahl der Inputs stehen.

Algorithmus 4.8 Nicht exklusive Problemzerlegung durch Partitionierung

Input μ, λ ▷ μ: Größe der Population ,λ: Anzahl Nachkommen
Input $x[1..N_x]$ ▷ x: Wert der vergangenen Problemgrößen
Input $y[1..N_y]$ ▷ y: Werte der aktuellen Mess- und Regelgrößen
Input $nmax$, N_P ▷ $nmax$: max. Anzahl Iterationen, N_P: Anzahl Teilprobleme
$P \leftarrow$ ProblemZerlPartitionierung$(\mu, \lambda, x, y, nmax, N_P)$ ▷ Alg. 4.4 oder 4.4
▷ P: Partitionierung der Problemgrößen
for $i = 1 \rightarrow N_P$ **do** ▷ für alle Partitionen
  $N_i \leftarrow$ evolInputPruning$(\mu, \lambda, x, y, nmax, P)$ ▷ evolutionäres Input Pruning
  ▷ analog Alg. 4.3
  ▷ N_i: neuronales Netz zur Approximation von Partition i
  return N
end for

Das Vorgehen bei der nicht exklusiven Problemzerlegung durch Partitionierung ist in Algorithmus 4.8 als Pseudocode dargestellt. Zu Beginn wird wie gewohnt eine Problemzerlegung durch Partitionierung in Variante 1 oder Variante 2 durchgeführt. Auf Basis der daraus resultierenden Partitionierung, erfolgt anschließend ein evolutionäres Input Pruning, ähnlich dem, welches in Algorithmus 4.3 verwendet wurde. Gepruned werden dabei die neuronalen Netze der einzelnen Partitionen, welche zu Beginn nur die Problemgrößen der eigenen Partition, später, im Verlauf der Evolution, aber potenziell auch die Problemgrößen der anderen Partitionen als Inputs erhalten können. Die resultierenden neuronalen Netze werden am Ende als Ergebnis

ausgegeben.
Der Pseudocode in Algorithmus 4.8 wurde zur Verwendung mit der Problemzerlegung durch Partitionierung in Variante 1 erstellt, bei welcher die Problemgrößen einer Partition auch nur durch ein einzelnes neuronales Netz approximiert werden. Für Variante 2, bei welcher jede Problemgröße jeweils ein eigenes Netz besitzt, muss der Algorithmus entsprechend noch mit einer weiteren Schleife erweitert werden, sodass alle Netze einer Partition gepruned werden.

Der Vorteil dieses kombinierten Einsatzes der Partitionierung der Problemgrößen und dem anschließendem Input Pruning gegenüber der einfachen Problemzerlegung durch Input Pruning liegt, zumindest wenn man annimmt, dass nur relativ wenige Problemgrößen auf mehrere Partitionen Einfluss nehmen, vor allem in einem guten „Startpunkt“ für das Input Pruning. Durch die vorherige Partitionierung sollten die meisten irrelevanten Inputs der neuronalen Netze bereits ausgeschlossen worden sein, sodass nur noch einige wenige Abhängigkeiten ergänzt werden müssen, was eine schneller Laufzeit und bessere Ergebnisse verspricht. Auch ist es für das evolutionäre Input Pruning gewöhnlich leichter, fehlende Inputs zu ergänzen, als Überflüssige zu entfernen, da diese einen relativ geringen Einfluss auf die Fitness besitzen.
Der Nachteil dieses Verfahrens liegt jedoch darin, dass der kombinierte Einsatz mit dem Verfahren zur automatischen Ermittlung der Teilprobleme aus Kapitel 4.4.3 nur schwer möglich ist. Bei diesem würde für die Partitionierung höchstwahrscheinlich kein „minimaler Schnitt“ herauskommen, sondern gleich alle voneinander abhängigen Problemgrößen zu einem Teilproblem zusammengefasst werden. Folglich muss die Anzahl der Teilprobleme abgeschätzt und vorgegeben werden. Das nachfolgende Input Pruning kann jedoch ungenaue Abschätzungen kompensieren. Auch ist es denkbar, das Verfahren mehrere Male, mit verschiedenen Abschätzungen der Anzahl der Teilprobleme zu starten und die Partitionierung mit dem besten Ergebnis zu verwenden, was allerdings die Laufzeit vervielfachen würde.

4.5 Zusammenfassung Methoden zur Problemzerlegung

Zu Beginn des Kapitels wurden mehrere Methoden zur Fitnessberechnung für die evolutionären Verfahren, also zur Bewertung von möglichen Problemzerlegungen, vorgestellt. Zum Einen ist dies die *Fitnessberechnung über die Zugehörigkeit der Problemgrößen*, bei welcher der Prozentsatz der zum korrekten Teilproblem zugeordneten Problemgrößen berechnet wurde:

$$F_{PG} = \max \left(\frac{\sum_{i=1}^{N_P} |P_{Ind_\sigma(i)} \cap P_{true_i}|}{|M|} \;\middle|\; \sigma : \{1, .., N_P\} \to \{1, ..., N_P\} \text{Permutationsfunktion} \right)$$

Dabei stellt P_{Ind_i} die i. Partition der zu prüfenden Partitionierung und entsprechend P_{true_i} die i. Partition der Partitionierung, welche den realen Teilproblemen entspricht, dar. Außerdem gibt M die Menge der Problemgrößen und N_p die Anzahl der Partitionen an.

Darüber hinaus gab es die *Fitnessberechnung über die Abhängigkeiten*, wo der Prozentsatz der korrekten Abhängigkeiten als Fitnesswert verwendet wird. Eine Abhängigkeit gilt dabei als korrekt, wenn sie nicht über Teilproblemgrenzen hinweg, sondern nur zwischen zwei Problemgrößen einer Partition bzw. eines Teilproblems besteht:

$$F_{Abh} = \frac{C}{|A|}$$

Mit:

$$C = \{(a_1, a_2) \in A \mid \exists j \in \{1, .., N_P\} : \exists (p_1, p_2) \in P_j \times P_j : p_1 = a_1, p_2 = a_2\}$$

Dabei ist A die Menge der Abhängigkeiten, welche aus den Tupeln zweier Problemgrößen (a_1, a_2) bestehen, bei welchen eine Abhängigkeit von Problemgröße a_1 zu a_2 vorhanden ist. Weiterhin stellt P_j die Partition j dar und enthält alle Problemgrößen, welche dieser Partition zugeordnet wurden. Schließlich gibt N_P die Anzahl der Parti-

tionen an.
Beide Varianten besaßen den Nachteil, dass die korrekte Zerlegung in Teilprobleme für ihre Verwendung bekannt sein musste, weshalb noch eine dritte Methode zur Fitnessberechnung vorgestellt wurde, die *Fitnessberechnung über die Fehlerrate der neuronalen Netze*, welche die Mess- und Regelgrößen approximieren. Verwendet wurde in dieser Arbeit dafür der Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y(x_i) - t_i)^2}$$

Dabei ist N die Anzahl der Trainingspattern, $y(x_i)$ die Ausgabe des neuronalen Netzes für den aktuellen Vektor der Eingangsgrößen x_i und t_i das Trainingspattern i.
Wird ein neuronales Netz auf mehr als eine Problemgröße trainiert, muss die Berechnung des Root Mean Sqared Errors entsprechend angepasst werden:

$$RMSE = \sqrt{\frac{1}{N \cdot N_y}\sum_{i=1}^{N}\sum_{j=1}^{N_y}(y_j(x_i) - t_{ij})^2}$$

Dabei stellt N_y hier die Anzahl der Outputneuronen dar. Außerdem ist $y_j(x_i)$ die Ausgabe des j. Outputneurons und t_{ij} das i. Trainingspattern für Outputneuron j.

Anschließend wurde eine Methode für das Network Pruning vorgestellt, also die Anpassung der Struktur und Größe des neuronalen Netzes an das Problem. Dafür wurde zuerst eine *Bayes'sche Regularisierung* durchgeführt, bei welcher nicht nur die Summe der quadratischen Fehler (SSE) sondern auch die Summe der quadrierten Gewichte (SSW) minimiert wurden:

$$E = \beta \cdot SSE + \alpha \cdot SSW$$

Dabei ist E die zu minimierende Fehlerfunktion und α und β lassen sich wie folgt berechnen:

$$\alpha = \frac{\gamma}{2 \cdot SSW}$$

$$\beta = \frac{N - \gamma}{2 \cdot SSE}$$

$$\gamma = N_w - 2\alpha \cdot trace\,(H)^{-1}$$

N ist dabei die Anzahl der Trainingspattern, N_w die Anzahl der Gewichte und H die beim Levenberg-Marquardt Algorithmus abfallende Hessematrix. Die neue Updateformel für die Gewichte bei diesem Levenberg-Marquardt Trainingsalgorithmus mit Bayes'scher Regularisierung lautet:

$$w_{new} = w_{old} - \left(\beta \cdot J^T J + (\alpha + \lambda) \cdot I\right)^{-1} \left(\beta \cdot J^T \epsilon\,(w_{old}) + \alpha \cdot w_{old}\right)$$

Dabei stellt J die Jacobi-Matrix dar, welche durch Backpropagation of Error ermittelt werden kann, I ist die Einheitsmatrix und ϵ ist der einfache Fehler des Netzoutputs $y - t$. Außerdem stellt λ die Lernrate dar.

Nach einem Training durch dieses Verfahren bis zu einem zumindest lokalen Minimum w_{Min} können für die Approximation irrelevante Gewichte des neuronalen Netzes gelöscht werden. Die Berechnung der Relevanz erfolgt bei dem in dieser Arbeit verwendeten *Optimal Brain Surgeon* Verfahren über die sogenannte Saliency S. Die Saliency für Gewicht q ergibt sich durch:

$$S_q = \frac{1}{2} \cdot \frac{w_{Min,q}^2}{\left[H^{-1}\right]_{q,q}} \cdot e_q$$

Dabei bezeichnet $\left[H^{-1}\right]_{q,q}$ das q. Diagonalelement der invertierten Hessematrix und e_q ist die Einheitsmatrix mit dem q. Element = 1. In jeder Iteration wird dabei das

Gewicht im Netz mit der kleinsten Saliency gelöscht. Die restlichen Gewichte können dann über folgende Formel aktualisiert werden:

$$\Delta w = -\frac{w_{Min,q}}{[H^{-1}]_{q,q}} H^{-1} \cdot e_q$$

Dies wird bis zu einem beliebigen Abbruchkriterium wiederholt, wofür sich beispielsweise der oben definierte Wert γ, welcher eine Abschätzung der effektiv vom neuronalen Netz verwendeten freien Parameter (Gewichte) darstellt, erhöht um einen Pufferwert von etwa 20%.

Zur Problemzerlegung selbst wurden drei Verfahren vorgestellt. Zum Einen die *Problemzerlegung durch Input Pruning*, bei welcher auf jede Mess- und Regelgröße ein neuronales Netz trainiert wurde. Diese Netze erhalten jeweils alle anderen Problemgrößen als Inputs und die Abhängigkeiten werden durch ein evolutionäres Input Pruning ermittelt. Die Zusammenhangskomponenten im daraus gebildeten Graphen ergeben die Teilprobleme.
Das zweite Verfahren ist die *Problemzerlegung durch Partitionierung in der ersten Variante*, bei welcher die Problemzerlegung durch eine Zerlegung in eine vorgegebene Anzahl Partitionen erzwungen werden soll. Dabei werden alle Problemgrößen in mehrere Partitionen zerlegt, welche der Anzahl der vorgegebenen Teilprobleme entsprechen und durch einen evolutionären Algorithmus evolviert werden. Zur Fitnessberechnung trainiert man dabei jeweils in jeder Partition ein neuronales Netz auf alle Mess- und Regelgrößen dieser Partition, mit den Problemgrößen dieser Partition als Eingangsgrößen.
Das letzte Verfahren ist die *Problemzerlegung durch Partitionierung Variante zwei.* Bei dieser werden die Problemgrößen ebenfalls partitioniert und evolviert. Beim Training der neuronalen Netze zur Fitnessberechnung wird jedoch auf jede Mess-und Regelgröße ein eigenes Netz trainiert, mit allen anderen Problemgrößen der Partition als Inputs. Dies soll sicherstellen, dass ein neuronales Netz zur Approximation einer Größe diese Größe selbst nicht als Input erhält.

Schließlich wurden einige Erweiterungen dieser Verfahren zur Problemzerlegung vorgestellt. Darunter die *Time Delay Estimation durch Mutual Information*. Durch diese sollen die Totzeiten der Eingangsgrößen der neuronalen Netze berechnet werden, um nur die Problemgrößen zu den Zeitpunkten als Inputs zu verwenden, welche auf den aktuellen Wert der zu approximierenden Größe den größten Einfluss besitzen. Dazu wird die Mutual Information paarweise für jede Output- und jede um T zeitverzögerte Inputgröße berechnet, wobei T durch ein vorher definiertes Zeitfenster begrenzt ist. Die Totzeit entspricht dann der Zeitverzögerung T, bei welcher die Mutual Information maximal wird.

Anschließend wurde eine Methode vorgestellt, durch welche die *Anzahl der Teilprobleme* bei der Problemzerlegung durch Partitionierung nicht vorgegeben werden muss, sondern *automatisch ermittelt* wird. Um dies zu erreichen, wurde die Evolution mit einer weitaus höheren Anzahl Partitionen gestartet als reale Teilprobleme vorhanden sind. Entsteht dann bei einer Mutation eine leere Partition, wird diese gelöscht. Auf diese Weise verringert sich im Laufe der Evolution die Anzahl der Partitionen immer weiter. Die Partitionierung, welche den tatsächlichen Teilproblemen entspricht, kann dabei am besten durch neuronale Netze approximiert werden, besitzt damit die höchste Fitness und setzt sich in der Evolution durch.

Gegen Ende des Kapitels wurde die Problemzerlegung durch Partitionierung für die Anwendung auf *nicht exklusiv trennbare Probleme* angepasst, bei welchen eine oder einige Problemgrößen auf mehrere Teilprobleme Einfluss besitzen. Dafür wurde zunächst eine gewöhnliche Problemzerlegung durch Partitionierung gemäß den Verfahren aus Kapitel 4.3.2 und 4.3.3 durchgeführt, und anschließend die Inputs der neuronalen Netze der entstandenen Partitionen evolutionär erweitert, sodass diese auch Zugriff auf Problemgrößen anderer Partitionen erhalten können.

Kapitel 5

Experimentelle Untersuchungen

In diesem Kapitel sollen die in Kapitel 4 vorgestellten Verfahren zur Problemzerlegung mitsamt ihrer Erweiterungen getestet und gegenübergestellt werden. Dafür werden zunächst die Testdaten vorgestellt. Anschließend sollen die Verfahren zunächst anhand dieser simulierten Testdaten und später auch anhand einiger Realweltdaten untersucht und verglichen werden. Am Ende des Kapitels erfolgt schließlich eine Gegenüberstellung der Verfahren mit denen einiger anderer aus dem APD-Projekt.

5.1 Verwendete Testdaten

Um die in dieser Arbeit verwendeten Verfahren auf ihre Tauglichkeit zu testen, wurden zum Einen mehrere künstlich erzeugte, simulierte Testdatensätze, und zum Anderen auch ein Realweltdatensatz verwendet, welche jeweils im Folgenden beschrieben werden sollen.

5.1.1 Simulierte Testdaten

Die hier verwendeten künstlichen Testdaten wurden im Rahmen des APD-Projektes in Kooperation mit der Firma Powitec mit Hilfe eines Simulators erzeugt. Der Vorteil von solchen simulierten Daten ist, dass alle Einfluss nehmenden Problemgrößen, Abhängigkeiten und mögliche Problemzerlegungen bereits bekannt sind und mit den Ergebnissen der Verfahren verglichen werden können und weiterhin eine möglichst breit

gefächerte Auswahl an Testfällen generiert werden kann.
Allen diesen simulierten Testdatensätzen ist gemein, dass sie aus einer Mischung aus Kenn-, Mess- und Regelgrößen im Sinne von Kapitel 2.1 bestehen und die Abhängigkeiten verschiedene Totzeiten und Tiefpässe besitzen, wobei letzteres sich darin niederschlägt, dass der Mittelwert der Problemgröße über mehrere Zeitschritte Einfluss besitzt.
Im Folgenden soll jeder der Datensätze kurz vorgestellt werden, jeweils eine Grafik mit den Abhängigkeiten und Teilproblemen und eine Tabelle mit den wichtigsten Eigenschaften werden gezeigt. Die Totzeiten und Tiefpässe der einzelnen Testdatensätze sind im Anhang A zu finden.

Testdatensatz $P1$

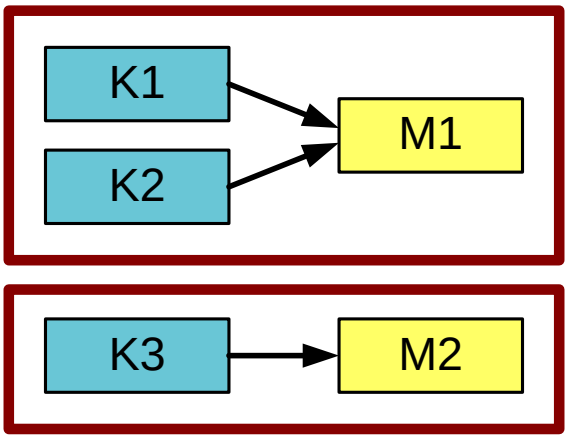

Abbildung 5.1: Abhängigkeiten des Testdatensatzes P1. Er ist ungeregelt und lässt sich in zwei Teilprobleme zerlegen.

Der Testdatensatz $P1$ besteht aus insgesamt fünf Problemgrößen, welche in Abbildung 5.1 mitsamt ihren Abhängigkeiten, dargestellt über gerichtete Pfeile, zu sehen sind. Kenngrößen (K) sind dabei blau, Messgrößen (M) gelb eingefärbt. Regelgrößen sind in diesem Problem nicht vorhanden. Es ist daher ungeregelt. Aufgrund der Abhängigkeiten bzw. deren nicht Vorhandensein zwischen bestimmten Größen lässt sich das Problem exklusiv in zwei Teilprobleme zerlegen.

Die Merkmale dieses Datensatzes sind in Tabelle 5.1 tabellarisch zusammengefasst. Tabelle A.1 im Anhang zeigt die Totzeiten und Tiefpässe für die Abhängigkeiten des Problems $P1$.

Datensatz	P1
Anzahl Problemgrößen	5
Anzahl Abhängigkeiten	3
Anzahl Teilprobleme	2
Eigenschaften	ungeregelt, exklusiv trennbar
Anzahl Datenpunkte	1000

Tabelle 5.1: Merkmale des Datensatzes $P1$

Testdatensatz $P1g$

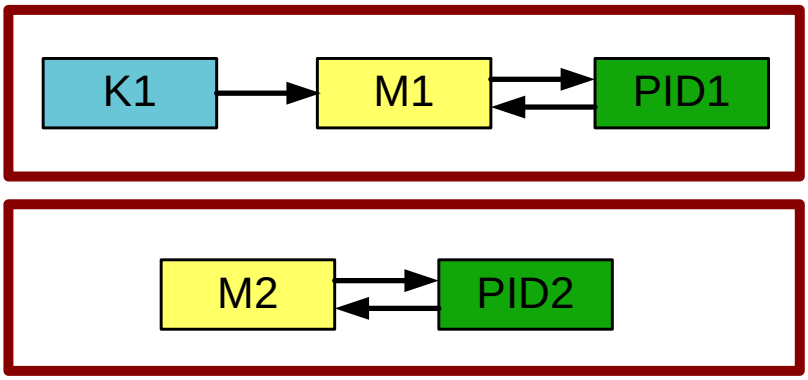

Abbildung 5.2: Abhängigkeiten des Testdatensatzes P1g. Er ist geregelt und lässt sich in zwei Teilprobleme zerlegen.

Testdatensatz $P1g$ besteht, wie auch Datensatz $P1$, aus fünf Problemgrößen und lässt sich ebenfalls exklusiv in zwei Teilprobleme zerlegen, ist aber im Gegensatz zu diesem geregelt. Die Problemgrößen und Abhängigkeiten sind in Abbildung 5.2 dargestellt. Kenngrößen (K) sind dabei blau, Messgrößen (M) gelb und Regelgrößen (PID) grün eingefärbt.
Die Merkmale dieses Datensatzes sind in Tabelle 5.2 und die Totzeiten und Tiefpässe der Abhängigkeiten in Tabelle A.2 zusammengefasst.

Datensatz	P1g
Anzahl Problemgrößen	5
Anzahl Abhängigkeiten	5
Anzahl Teilprobleme	2
Eigenschaften	geregelt, exklusiv trennbar
Anzahl Datenpunkte	1000

Tabelle 5.2: Merkmale des Datensatzes $P1g$

Testdatensatz $P2$

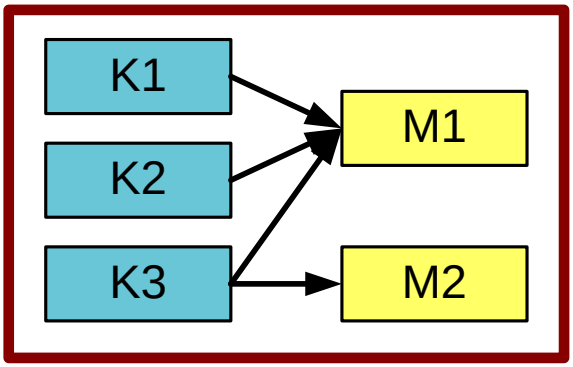

Abbildung 5.3: Abhängigkeiten des Testdatensatzes P2. Er ist ungeregelt und lässt sich nicht exklusiv in Teilprobleme zerlegen.

Testdatensatz $P2$ besteht wieder aus insgesamt fünf Problemgrößen und kann, aufgrund der fehlenden Regelgrößen, als ungeregelt klassifiziert werden. Wegen der Größen und ihrer Abhängigkeiten, welche in Abbildung 5.3 zu sehen sind, lässt sich das Problem nicht exklusiv in Teilprobleme zerlegen. Da hier aber nur die Abhängigkeiten der Problemgröße $K3$ auf beide Teilprobleme Einfluss besitzen, wird der Datensatz als partiell trennbar bezeichnet. Partiell trennbare Datensätze sollen später beim Test der nicht exklusiven Problemzerlegung Verwendung finden.
In Tabelle 5.3 sind die Merkmale des Datensatzes $P2$ zu sehen, während Tabelle A.3 die Totzeiten und Tiefpasse der Abhängigkeiten zeigt.

Datensatz	P2
Anzahl Problemgrößen	5
Anzahl Abhängigkeiten	4
Anzahl Teilprobleme	1
Eigenschaften	ungeregelt, partiell trennbar
Anzahl Datenpunkte	1000

Tabelle 5.3: Merkmale des Datensatzes $P2$

Testdatensatz $P2g$

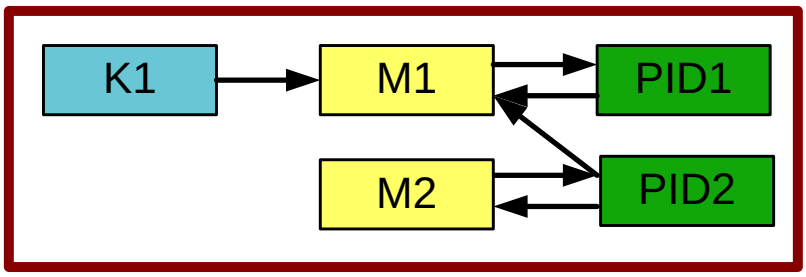

Abbildung 5.4: Abhängigkeiten des Testdatensatzes P2g. Er ist geregelt und lässt sich nicht exklusiv in Teilprobleme zerlegen.

Testdatensatz $P2g$ besitzt insgesamt fünf Problemgrößen und wird geregelt. Aufgrund der Tatsache, dass Problemgröße $PID2$ auf beide Teilprobleme Einfluss nimmt, was in Abbildung 5.4 zu sehen ist, ist das Problem nur partiell trennbar.
Die Merkmale des Testdatensatzes $P2g$ sind in Tabelle 5.4 dargestellt, während die Totzeiten und Tiefpässe in Tabelle A.4 zu sehen sind.

Datensatz	P2g
Anzahl Problemgrößen	5
Anzahl Abhängigkeiten	6
Anzahl Teilprobleme	1
Eigenschaften	geregelt, partiell trennbar
Anzahl Datenpunkte	1000

Tabelle 5.4: Merkmale des Datensatzes $P2g$

Testdatensatz $P3$

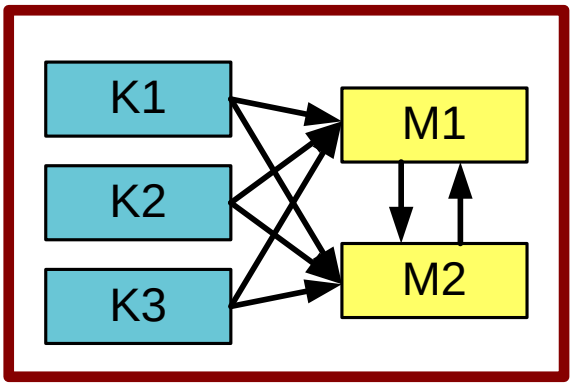

Abbildung 5.5: Abhängigkeiten des Testdatensatzes P3. Er ist ungeregelt und lässt sich nicht in Teilprobleme zerlegen.

Der Testdatensatz $P3$ besteht wieder aus fünf Problemgrößen und ist außerdem, aufgrund der fehlenden Regelgrößen, ungeregelt. Wie in Abbildung 5.5 zu sehen, welche die Problemgrößen mitsamt ihren Abhängigkeiten zeigt, besitzt hier jede Größe Einfluss auf jede andere Größe, weshalb das Problem als nicht trennbar klassifiziert werden muss.
Die Merkmale des Datensatzes $P3$ sind in Tabelle 5.5 zu sehen. Tabelle A.5 zeigt die Totzeiten und Tiefpässe der Abhängigkeiten des Problems.

Datensatz	P3
Anzahl Problemgrößen	5
Anzahl Abhängigkeiten	8
Anzahl Teilprobleme	1
Eigenschaften	ungeregelt, nicht trennbar
Anzahl Datenpunkte	1000

Tabelle 5.5: Merkmale des Datensatzes $P3$

Testdatensatz *P3g*

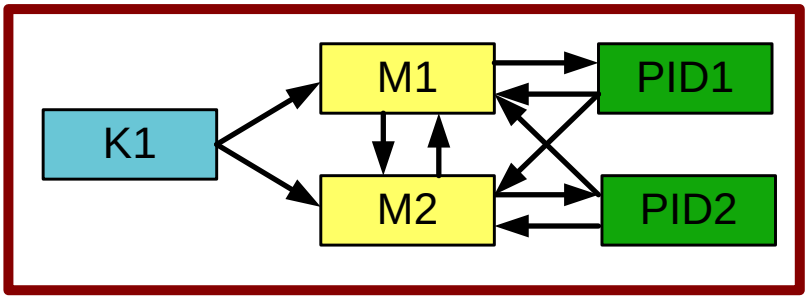

Abbildung 5.6: Abhängigkeiten des Testdatensatzes P3g. Er ist geregelt und lässt sich nicht in Teilprobleme zerlegen.

Testdatensatz *P3g* besteht aus insgesamt fünf Problemgrößen und ist geregelt. Die Problemgrößen und ihre Abhängigkeiten sind in Abbildung 5.6 grafisch dargestellt. Wie dort zu sehen, lässt sich das Problem nicht in Teilprobleme zerlegen, ist also nicht trennbar.
Die Merkmale dieses Datensatzes sind in Tabelle 5.6 und die Totzeiten in Tabelle A.6 zusammengefasst.

Datensatz	P3g
Anzahl Problemgrößen	5
Anzahl Abhängigkeiten	10
Anzahl Teilprobleme	1
Eigenschaften	geregelt, nicht trennbar
Anzahl Datenpunkte	1000

Tabelle 5.6: Merkmale des Datensatzes *P3g*

Testdatensatz *P4gS*

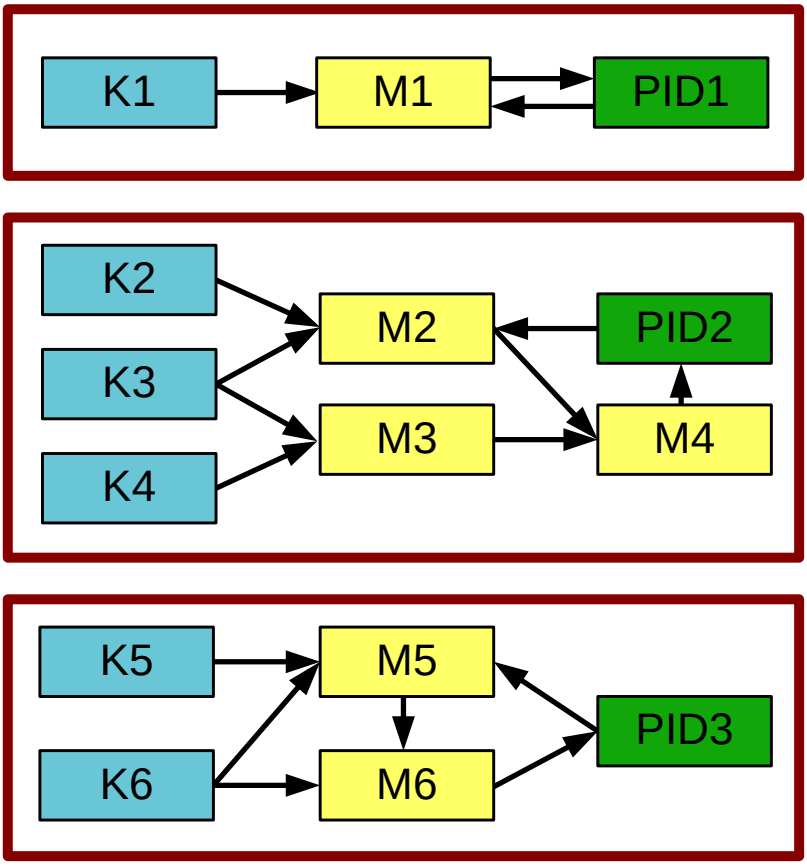

Abbildung 5.7: Abhängigkeiten des Testdatensatzes P4gS. Er ist geregelt und lässt sich exklusiv in drei Teilprobleme zerlegen.

Der Testdatensatz *P4gS* wurde entworfen, um ein etwas komplexeres Problem zum Test zur Verfügung zu stellen, im Gegensatz zu den bisherigen Problemen, welche mit nur fünf Problemgrößen und maximal zwei Teilproblemen recht einfach ausfielen. Während diese eher dazu dienen sollen nachzuweisen, dass die Verfahren prinzipiell funktionieren, soll am Datensatz *P4gS* gezeigt werden, dass sie auch in komplexeren Situationen anwendbar sind. Demzufolge enthält der Datensatz insgesamt 15 Problemgrößen, welche sich exklusiv in drei geregelte Teilprobleme zerlegen lassen. Dargestellt sind diese Größen und ihre Zusammenhänge in Abbildung 5.7.
Die Eigenschaften des Datensatzes sind in Tabelle 5.7 zusammengefasst. Um den

Schwierigkeitsgrad weiter zu erhöhen, wurde ebenfalls eine größere Bandbreite an Totzeiten und Tiefpässen für die Abhängigkeiten verwendet, welche in Tabelle A.7 zu sehen sind.

Datensatz	P4gS
Anzahl Problemgrößen	15
Anzahl Abhängigkeiten	17
Anzahl Teilprobleme	3
Eigenschaften	geregelt, exklusiv trennbar
Anzahl Datenpunkte	5000

Tabelle 5.7: Merkmale des Datensatzes $P4gS$

Testdatensatz *P9g*

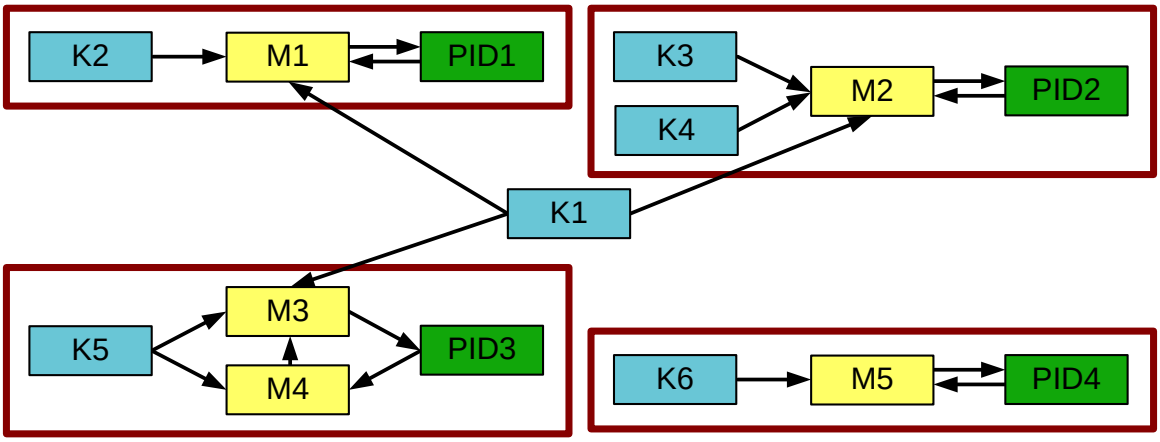

Abbildung 5.8: Abhängigkeiten des Testdatensatzes P9g. Er ist geregelt und lässt sich nicht exklusiv in Teilprobleme zerlegen.

Der Testdatensatz *P9g* ist in Abbildung 5.8 zu sehen. Er ist ein etwas komplexerer Datensatz mit insgesamt 15 Problemgrößen und geregelt. Es lassen sich etwa vier Teilprobleme erkennen, jedoch besitzt die Kenngröße K1 Einfluss auf drei der Teilprobleme. Der Datensatz ist somit nicht exklusiv trennbar (zumindest nicht in die vier Teilprobleme) und soll deshalb vor allem dazu genutzt werden, die Verfahren zur nicht exklusiven Problemzerlegung zu testen.
Die Eigenschaften von Datensatz *P9g* sind in Tabelle 5.8 zu sehen, während die Tiefpässe und Totzeiten in Tabelle A.8 aufgelistet sind.

Datensatz	P9g
Anzahl Problemgrößen	15
Anzahl Abhängigkeiten	18
Anzahl Teilprobleme	1
Eigenschaften	geregelt, partiell trennbar
Anzahl Datenpunkte	5000

Tabelle 5.8: Merkmale des Datensatzes *P9g*

5.1.2 Realweltdaten

Um zu demonstrieren, dass die Verfahren zur Problemzerlegung nicht nur bei simulierten Daten, sondern auch bei realen Problemen einsetzbar sind, wurden sie auch an Realweltdaten getestet.

Testdatensatz $P5g$

Dieser Datensatz ist von der Firma „Powitec“ zur Verfügung gestellt worden und stammt aus einer Müllverbrennungsanlage. Dazu wurde, im Hinblick auf die Ziele dieser Arbeit, ein Problem ausgewählt, welches durch menschliche Experten nicht oder zumindest nur schwer durchschaubar und zerlegbar ist. Aufgrund dieser Tatsache sind die Anzahl der Teilprobleme, die Abhängigkeiten und auch die vorhandenen Totzeiten und Tiefpässe leider nicht bekannt. Das Problem besteht jedoch aus insgesamt 24 Problemgrößen, darunter 2 Kenngrößen, 14 Messgrößen und 8 Regelgrößen. Es ist außerdem geregelt und nur partiell trennbar.
Die bekannten Merkmale zu diesem Datensatz sind in Tabelle 5.9 tabellarisch zusammengefasst.

Datensatz	P5g
Anzahl Problemgrößen	24
Anzahl Abhängigkeiten	*unbekannt*
Anzahl Teilprobleme	*unbekannt*
Eigenschaften	geregelt, partiell trennbar
Anzahl Datenpunkte	15000

Tabelle 5.9: Merkmale des Datensatzes $P5g$

5.2 Test der Algorithmen

Im Folgenden werden die drei in Kapitel 4.3 vorgestellten Verfahren, nämlich die Problemzerlegung durch Input Pruning, die Problemzerlegung durch Partitionierung Variante 1 und die Problemzerlegung durch Partitionierung Variante 2 an den Testdatensätzen getestet. Sie kommen dabei jeweils in fünf Varianten zur Anwendung:

1. Die Verfahren werden in der *Standardvariante*, wie sie in Kapitel 4.3 für jedes der Verfahren beschrieben sind, getestet. Die Totzeiten, Tiefpässe und die Anzahl der Teilprobleme werden dabei mit Hilfe des Vorwissens über die generierten Testdatensätze den Verfahren vorgegeben.

2. Die Totzeiten und Tiefpässe werden durch die *Time Delay Estimation* aus Kapitel 4.4.1 automatisch durch die Verfahren selbst ermittelt und anschließend zur Problemzerlegung verwendet. Nur die Anzahl der Teilprobleme wird den Verfahren vorgegeben.

3. Die Verfahren ermitteln die *Anzahl der Teilprobleme* durch den in Kapitel 4.4.3 vorgestellten Algorithmus selbst. Die Totzeiten und Tiefpässe werden den Verfahren vorgegeben.

4. Die Verfahren werden zur *nicht exklusiven Problemzerlegung* nach dem Vorgehen aus Kapitel 4.4.4 verwendet, wobei die Anzahl der nicht exklusiv trennbaren Teilprobleme und die Totzeiten und Tiefpässe vorgegeben werden.

5. Die Verfahren werden an den Realweltdaten getestet, wobei sie *mitsamt aller ihrer Erweiterungen* zum Einsatz kommen. Das Problem wird dabei nicht exklusiv zerlegt und die Totzeiten und Tiefpässe werden durch die Verfahren selbst ermittelt. Lediglich auf die automatische Ermittlung der Anzahl der Teilprobleme muss verzichtet werden, da sie mit der nicht exklusiven Problemzerlegung inkompatibel ist.

Jedes der drei Verfahren zur Problemzerlegung soll, mit Ausnahme der nicht kompatiblen und nicht sinnvollen Kombinationen, in jeweils jeder der fünf Varianten getestet

werden.

Die verwendeten Algorithmen besitzen eine große Anzahl an Parametern, welche den Verfahren vorgegeben werden müssen. Die Werte der zum Test verwendeten Parameter sind in Tabelle 5.10 zusammengefasst. Sie basieren dabei entweder auf den in den zugehörigen Literaturquellen empfohlenen Werten oder auf Erfahrungswerten.

Aufgrund der hohen Laufzeit der Verfahren, bedingt durch die Evolution und die damit verbundene hohe Anzahl zu trainierender neuronaler Netze, konnte im Zeitrahmen der Masterarbeit jede Kombination von Verfahren und Testfall nur einmal getestet werden, weshalb, durch statistische Schwankungen, die Qualität der Ergebnisse zum Teil vom Zufall beeinflusst ist. Die Struktur der gefundenen Lösungen lässt jedoch gewöhnlich Rückschlüsse auf die Brauchbarkeit der verwendeten Verfahren zu.
Im Folgenden werden die Ergebnisse der Tests grafisch und tabellarisch gezeigt und darauf basierend die Verfahren bewertet und untereinander verglichen. Aufgrund der hohen Anzahl der Grafiken und Tabellen sollen hier nicht alle einzeln dargestellt, sondern nur eine Auswahl herausgenommen werden. Es finden sich jedoch alle im Anhang B wieder.

Leider standen bis zum Abgabetermin dieser Arbeit die Ergebnisse des Input Pruning Verfahrens für den Testdatensatz $P4gS$ in allen Varianten noch nicht zur Verfügung. Die fehlenden Werte sind entsprechend mit xxx gekennzeichnet, Sie werden jedoch, sobald die Berechnungen beendet sind, nachgeliefert. Die Ergebnisse des Input Pruning Verfahrens mit den kleineren Testdatensätzen lassen aber bereits Schlussfolgerungen auf die vermutliche Tendenz dieser Ergebnisse zu, auf welche an entsprechender Stelle eingegangen wird.

Bayes'sche Regularisierung		
Parameter	Beschreibung	Wert
α	Wichtung für SSW bei der Berechnung von E	0
β	Wichtung für SSE bei der Berechnung von E	1
Evolution		
Parameter	Beschreibung	Wert
λ	Anzahl der Nachkommen	10
μ	Größe der Population	20
n	Anzahl Generationen	500
$p_{crossover}$	Wahrscheinlichkeit für ein Auftreten von Crossover	$0,6$
$p_{crossover_swap}$	Wahrscheinlichkeit für einen Austausch der Gene beim Auftreten von Crossover	$0,5$
p_{mut_init}	initiale Mutationswahrscheinlichkeit (wird durch lokale Varianzanpassung angepasst)	$0,2$
lokale Varianzanpassung		
Parameter	Beschreibung	Wert
r_1	Parameter für lokale Varianzanpassung	$0,1$
r_2	Parameter für lokale Varianzanpassung	$0,2$
$\sigma_{i_e_init}$	initiale Standardabweichung von e	$0,1$
$\sigma_{i_pmut_init}$	initiale Standardabweichung von p_{mut}	$0,05$
Neuronale Netze		
Parameter	Beschreibung	Wert
n_{max}	maximale Anzahl Trainingsdurchläufe	200
N_{Hidden_init}	initale Anzahl Hiddenneuronen (wird durch OBS angepasst)	15
λ_{init}	initale Lernrate	$0,005$

Tabelle 5.10: Parameter der zum Test verwendeten Verfahren

Optimal Brain Surgeon		
Parameter	Beschreibung	Wert
e_{init}	initaler Pufferwert für die Anzahl der benötigten Gewichte (wird durch lokale Varianzanpassung angepasst)	$0,2$
Time Delay Estimation		
Parameter	Beschreibung	Wert
C_{max}	maximal zulässige Breite des Tiefpasses	10
T_{max}	maximal zulässige Totzeit	10

Tabelle 5.11: Parameter der zum Test verwendeten Verfahren

5.2.1 Test der Standardverfahren

In diesem Abschnitt werden die Verfahren in der Standardvariante, ohne Erweiterungen getestet. Die Totzeiten, Tiefpässe und die Anzahl der Teilprobleme wurden dabei mit Hilfe des Vorwissens über die generierten Testdatensätze den Verfahren vorgegeben.
Da hier die exklusive Problemzerlegung getestet werden soll, wurden ausschließlich die Testdatensätze verwendet, bei denen eine solche Zerlegung möglich ist. Das sind die Datensätze 1, $1g$ und $4gS$. Die Ergebnisse wurden in Tabelle 5.12 für jeweils die drei Verfahren, die drei Datensätze und die drei Arten der Fitnessberechnung nach Kapitel 4.1 dargestellt.

Datensatz	**Fitness-berechn.**	**Input Pruning**	**Partitionierung Variante 1**	**Partitionierung Variante 2**
P1	F_{PG}	–	100%	100%
	F_{Abh}	–	100%	100%
	F_{RMSE}	$2,48 \cdot 10^{-5}\%$	$2,24 \cdot 10^{-4}\%$	$8,74 \cdot 10^{-4}\%$
P1g	F_{PG}	–	100%	100%
	F_{Abh}	–	100%	100%
	F_{RMSE}	$2,82\%$	$1,56\%$	$3,02\%$
P4gS	F_{PG}	–	$86,7\%$	60%
	F_{Abh}	–	$88,2\%$	$70,6\%$
	F_{RMSE}	$xxx\%$	$2,02\%$	$5,25\%$

Tabelle 5.12: Ergebnisse der Standardverfahren

Mit Blick auf die Fitnessberechnung über die Fehlerrate F_{RMSE} scheint die Problemzerlegung durch Input Pruning die besten Ergebnisse zu liefern. An dieser Stelle sei noch einmal darauf hingewiesen, dass die Fitnessberechnungen über die Zugehörigkeit der Problemgrößen F_{PG} und über die Abhängigkeiten F_{Abh} sich nicht zur Bewertung des Input Pruning Verfahrens eignen und deshalb die entsprechenden Einträge freige-

lassen wurden. Der gute Root Mean Squared Error verwundert jedoch nicht, wenn man bedenkt, dass das Input Pruning Verfahren das Einzige ist, welches für jede Problemgröße ein komplettes Input Pruning durchführt, sodass es ausschließlich die Größen als Inputs erhält, welche es benötigt. Bei der Problemzerlegung durch Partitionierung in beiden Varianten hingegen bekommt jede Problemgröße alle anderen Größen der selben Partition als Eingangsgrößen, selbst wenn ein Teil davon genügen würde. Die Reduzierung der Dimensionalität und der Komplexität für die neuronalen Netze ist beim Input Pruning Verfahren also am größten.

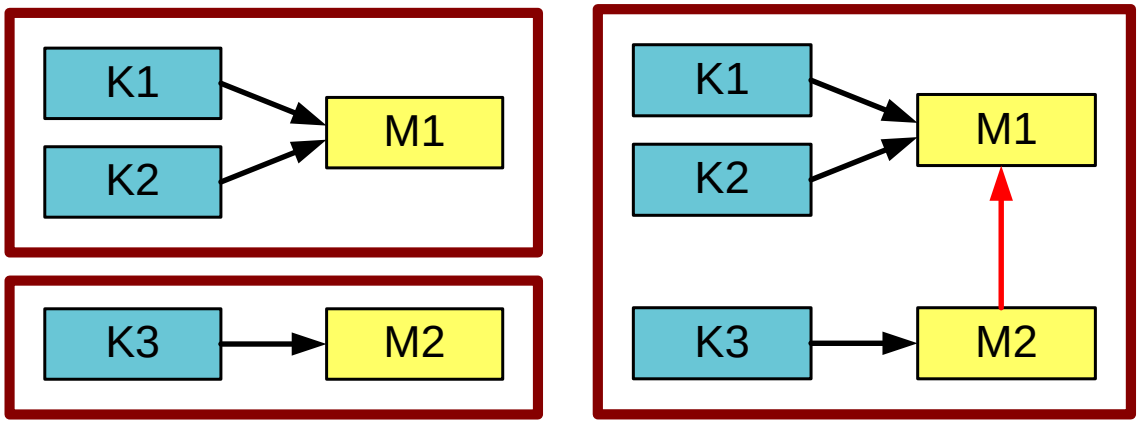

Abbildung 5.9: Problemzerlegung des Testdatensatzes P1 durch Input Pruning in der Standardvariante. Links sind die korrekten Abhängigkeiten (schwarze Pfeile) und die korrekte Problemzerlegung (rote Umrandung) zu sehen, während rechts das Ergebnis des Input Pruning Ansatzes abgebildet ist. Rote Pfeile markieren dabei die Abhängigkeiten, welche in der Realität nicht vorhanden sind.

Bei der Betrachtung der Grafiken mit den Abhängigkeiten, beispielsweise die Grafik für Datensatz $P1$, welche in Abbildung 5.9 zu sehen ist, wird jedoch der Nachteil des Verfahrens deutlich. Links ist die korrekte Problemzerlegung mit den korrekten Abhängigkeiten dargestellt, während rechts die vom Input Pruning Verfahren ermittelten Abhängigkeiten zu sehen sind, wobei korrekte Abhängigkeiten schwarz, nicht vorhandene Abhängigkeiten dagegen rot gefärbt sind. Obwohl das Input Pruning die

meisten nicht vorhandenen Abhängigkeiten gelöscht hat, blieb eine einzige Abhängigkeit von M2 nach M1 übrig. Daraus resultiert ein Abhängigkeitsgraph, welcher sich nicht mehr in Teilprobleme zerlegen lässt. Da dieses Problem selbst bei den hier verwendeten relativ kleinen Testfällen auftritt, steht zu erwarten, dass es bei einer höheren Anzahl an Problemgrößen, beispielsweise bei Datensatz $P4gS$ und mehr noch bei der Anwendung in der Praxis, verstärkt auftritt. Offenbar fallen, auch wenn bei neuronalen Netzen eine verringerte Anzahl an nicht benötigten Eingangsgrößen im Allgemeinen zu besseren Ergebnissen führt, eine oder einige wenige zusätzliche Inputgrößen nicht mehr so stark ins Gewicht, sodass oftmals, verbunden mit dem Zufallsfaktor bei der Evolution, gelegentlich zu viele, nicht vorhandene Abhängigkeiten ermittelt werden.
Positiv hervorzuheben ist dagegen, dass keine in Wirklichkeit vorhandenen Abhängigkeiten fälschlicherweise gelöscht wurden, sodass, im Gegensatz zu Problemzerlegungen durch Partitionierung, nicht befürchtet werden muss, dass den Problemgrößen zur Approximation und zur Regelung benötigte Daten vorenthalten werden.

Zu den beiden Verfahren der Problemzerlegung durch Partitionierung lässt sich sagen, dass diese die kleineren Testdatensätze $P1$ und $P1g$ jeweils zu 100% korrekt zerlegt haben. Offensichtlich funktioniert die hinter diesen Verfahren steckende Idee zumindest grundsätzlich. Der Root Mean Squared Error F_{RMSE} der ersten Variante ist dabei durchgängig etwas besser, was jedoch zu erwarten stand, da dieses Verfahren zur Approximation der Problemgrößen auch die vergangenen Werte der Problemgröße selbst als Inputs erhält, was sich natürlich positiv auf die Fehlerrate auswirkt. Ein direkter Vergleich der beiden Verfahren auf diese Weise sollte deshalb jedoch fairerweise vermieden werden.
Bei einer Gegenüberstellung der Fitness über die Zuordnung der Problemgrößen und der Abhängigkeiten bei Datensatz $P4gS$ besitzt jedoch ebenfalls die erste Variante der Problemzerlegung durch Partitionierung die besseren Ergebnisse. Ein Blick auf die Ergebnisgrafik 5.10 offenbart aber auch hier die Schwäche des Verfahrens. Links ist hier die korrekte, rechts die vom Verfahren ermittelte Partitionierung zu sehen. Die Abhängigkeiten in der rechten Grafik wurden dabei nachträglich, mit

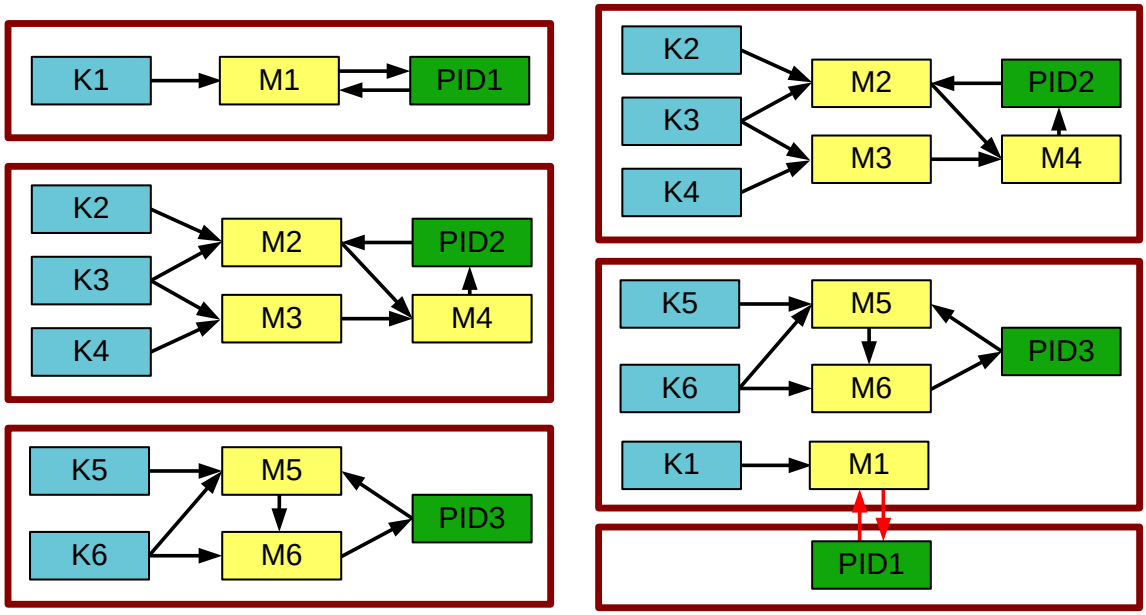

Abbildung 5.10: Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 1. Links ist die korrekte, rechts die vom Verfahren ermittelte Partitionierung zu sehen. Korrekte Abhängigkeiten sind durch schwarze und nicht korrekte Abhängigkeiten durch rote Pfeile dargestellt.

Hilfe des Vorwissens über das Problem, ergänzt. Korrekte Abhängigkeiten, welche zwischen Problemgrößen einer Partition bestehen, sind dabei schwarz, nicht korrekte Abhängigkeiten, welche zwischen Problemgrößen verschiedener Partitionen bestehen, rot eingefärbt.

Hierbei ist der in Kapitel 4.3.2 beschriebene Fall eingetreten, in dem das Verfahren das Problem eigentlich in weniger als die vorgegebene Anzahl an Teilprobleme zerlegen wollte. Während das mittlere der links gezeigten Teilprobleme noch korrekt erkannt wurde, sind die restlichen beiden größtenteils zu einem einzigen Teilproblem zusammengefasst worden. Die dem Verfahren vorgeschriebene Anzahl an drei Teilproblemen wurde erreicht, indem die Größe PID1 einer eigenen Partition zugeteilt wurde. Dadurch, dass bei diesem Verfahren die neuronalen Netze zur Approximation der Problemgrößen die vergangenen Werte der Problemgrößen selbst als Inputs erhalten, lässt sich der Approximationsfehler für PID1 trotz dieser offensichtlichen

Fehlentscheidung relativ gering halten (siehe dazu auch Kapitel 4.3.2 gegen Ende). Auch wenn die Fitnesswerte für das Verfahren also relativ gut scheinen, ist die Lösung im Sinne einer Problemzerlegung eher ungewollt.

Die Problemzerlegung durch Partitionierung Variante 2 schafft es ebenfalls nicht, das Problem $P4gS$ korrekt zu zerlegen. Abbildung 5.11 zeigt das zugehörige Ergebnis. Auch hier hat das Verfahren mehrere Teilprobleme, das obere und das untere der linken Seite, in einer Partition zusammengefasst. Da für dieses Verfahren, aufgrund der Architektur der neuronalen Netze, allerdings nicht die Möglichkeit bestand, eine Problemgröße durch sich selbst zu approximieren, wurde das dritte Teilproblem in zwei Teile zerlegt.

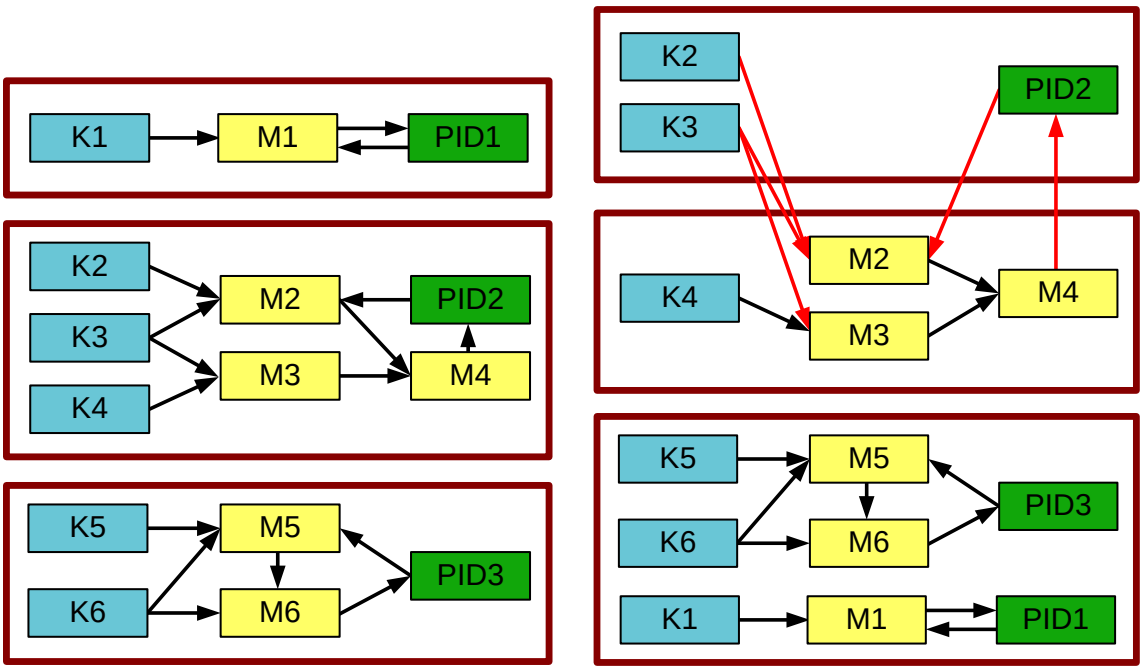

Abbildung 5.11: Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 2. Links ist die korrekte, rechts die vom Verfahren ermittelte Partitionierung zu sehen. Korrekte Abhängigkeiten sind durch schwarze und nicht korrekte Abhängigkeiten durch rote Pfeile dargestellt.

Allgemein lässt sich sagen, dass die Problemzerlegung durch Input Pruning für eine Problemzerlegung im Sinne dieser Arbeit zu fehleranfällig ist. Die Problemzerlegung durch Partitionierung funktioniert bei kleineren Problemen sehr gut. Mit zunehmender Anzahl an Problemgrößen und Abhängigkeiten bleibt sie zwar grundsätzlich anwendbar und kann brauchbare Lösungen liefern, scheint aber trotz der Evolution zunehmend Schwierigkeiten zu haben, die optimale Lösung zu finden und kann in lokalen Optima mit zum Teil zusammengefassten und zum Teil zerrissenen Teilproblemen hängen bleiben.

5.2.2 Test der Time Delay Estimation

Im Folgenden soll die Time Delay Estimation zur Ermittlung der Totzeiten und Tiefpässe der Abhängigkeiten getestet werden. Dafür wurde das in Kapitel 4.4.2 beschriebene Verfahren verwendet, welches für jedes Input-Output-Paar durch Maximierung der Mutual Information eine entsprechende Totzeit bzw. einen entsprechenden Tiefpass berechnet. Nachfolgend werden für die tatsächlich vorhandenen Abhängigkeiten diese ermittelten Werte mit den wahren verglichen. Die berechneten Totzeiten und Tiefpässe für die nicht existenten Abhängigkeiten sind der Vollständigkeit halber im Anhang C zu finden.

Abhängigkeit	wahre Totzeit	ermittelte Totzeit	wahrer Tiefpass	ermittelter Tiefpass
K1 → M1	1	1	1	1
K2 → M1	1	1	1	1
K3 → M2	1	1	1	1

Tabelle 5.13: Totzeiten und Tiefpässe des Datensatzes $P1$

Tabelle 5.13 zeigt die Ergebnisse für Testdatensatz $P1$. Die Abhängigkeiten sind zeilenweise abgetragen. In den Spalten finden sich paarweise links die wahren und rechts die berechneten Werte für die Totzeiten bzw. Tiefpässe wieder. Im Folgenden

sind falsch berechnete Werte rot markiert.
Für alle drei Abhängigkeiten wurden sowohl die Totzeiten als auch die Tiefpässe korrekt ermittelt. Für dieses relativ einfache Problem scheint das Verfahren somit völlig ausreichend.

Abhängigkeit	wahre Totzeit	ermittelte Totzeit	wahrer Tiefpass	ermittelter Tiefpass
K1 → M1	1	7	1	2
M1 → PID1	1	1	1	7
M2 → PID2	1	10	1	3
PID1 → M1	1	1	1	1
PID2 → M2	1	7	1	10

Tabelle 5.14: Totzeiten und Tiefpässe des Datensatzes $P1g$

Anders sieht es bei den Datensätzen $P1g$ und $P4gS$ aus, deren Ergebnisse in den Tabellen 5.14 und 5.15 zu finden sind. Zwischen einem Viertel und einem Drittel der Werte wurden korrekt ermittelt, während die restlichen teilweise massiv überschätzt wurden.
Die Gründe dafür sind vermutlich beim Rauschen auf den Datenwerten zu finden, welches bei diesen Datensätzen, im Gegensatz zu Testdatensatz $P1$, wo nur sehr geringes Rauschen vorhanden war, merklich höher ist. Die Approximierung der Mutual Information kann somit nur relativ ungenau vorgenommen werden, was sich auch auf die Berechnung der Totzeiten und Tiefpässe auswirkt.

Da das Hauptanliegen hier aber weniger in der Time Delay Estimation, sondern hauptsächlich in der Problemzerlegung liegt, werden nachfolgend die Ergebnisse einer Problemdekomposition durch die entsprechenden Verfahren gezeigt, wobei die durch die Time Delay Estimation berechneten Totzeiten und Tiefpässe verwendet wurden. Dadurch soll die Frage geklärt werden, ob und wie stark sich die teilweise fehlerhaften Werte für die Totzeiten und Tiefpässe auf die Problemzerlegung auswirken. Die

Abhängigkeit	wahre Totzeit	ermittelte Totzeit	wahrer Tiefpass	ermittelter Tiefpass
K1 → M1	2	2	3	3
K2 → M2	5	4	3	3
K3 → M2	1	1	2	9
K3 → M3	2	2	5	5
K4 → M3	5	1	2	10
K5 → M5	2	10	3	7
K6 → M5	5	2	2	5
K6 → M6	2	1	3	10
M1→ PID1	1	10	1	10
M2 → M4	2	2	3	3
M3 → M4	1	6	5	10
M4 → PID2	1	9	1	10
M5 → M6	5	5	2	2
M6 → PID3	1	10	1	10
PID1 → M1	5	1	2	10
PID2 → M2	5	8	3	9
PID3 → M5	5	10	2	8

Tabelle 5.15: Totzeiten und Tiefpässe des Datensatzes $P4gS$

Anzahl der Teilprobleme wurde dabei den Verfahren vorgegeben. Für eine exklusive Problemdekomposition sind entsprechend die Testdatensätze $P1$, $P1g$ und $P4gS$ verwendet worden. Die ermittelten Fitnesswerte sind in Tabelle 5.16 zu finden.

Die Ergebnisse des Input Pruning Verfahrens, welches bereits mit vorgegebenen Werten für die Totzeiten und Tiefpässe oftmals zu viele Abhängigkeiten ermittelte, verschlechtern sich noch einmal ein wenig und es zeigt sich, dass eine Problemzerlegung selbst bei kleinen Problemen somit nahezu unmöglich gemacht wird. Die Grafiken dazu sind im Anhang B.2.1 zu finden.

Datensatz	Fitness-berechn.	Input Pruning	Partitionierung Variante 1	Partitionierung Variante 2
P1	F_{PG}	–	100%	100%
	F_{Abh}	–	100%	100%
	F_{RMSE}	$1,79 \cdot 10^{-5}\%$	$4,2 \cdot 10^{-4}\%$	$1,37 \cdot 10^{-3}\%$
P1g	F_{PG}	–	80%	100%
	F_{Abh}	–	60%	100%
	F_{RMSE}	$4,20\%$	$2,95\%$	$4,86\%$
P4gS	F_{PG}	–	$66,7\%$	$46,7\%$
	F_{Abh}	–	$70,6\%$	$64,7\%$
	F_{RMSE}	$xxx\%$	$2,37\%$	$4,39\%$

Tabelle 5.16: Ergebnisse der Problemzerlegung mit Time Delay Estimation

Wenig überraschend dagegen wurde auch hier das Problem $P1$ von den beiden Verfahren zur Problemzerlegung durch Partitionierung entsprechend den realen Teilproblemen zerlegt, da hier die Werte für die Totzeiten und Tiefpässe korrekt durch die Time Delay Estimation berechnet wurden. Bei Problem $P1g$ jedoch sind bereits die Auswirkungen der zum Teil fehlerhaft berechneten Werte spürbar. Der Root Mean Squared Error steigt für alle Verfahren um etwa 50%, und während die Problemzerlegung durch Partitionierung Variante 2 das Problem korrekt zerlegt, liefert Variante 1 hier eine fehlerhafte Partitionierung. Diese ist in Abbildung 5.12 zu sehen. Dabei tritt das für dieses Verfahren bereits bekannte Problem auf, dass eine Problemgröße in einer eigenen Partition landet und durch sich selbst approximiert wird.

Auch die Ergebnisse für Datensatz $P4gS$ haben sich etwas verschlechtert. Beispielhaft hierfür ist in Abbildung 5.13 die Partitionierung durch Variante 1 zu sehen. Im Gegensatz zur Problemzerlegung durch Partitionierung ohne Time Delay Estimation, wo meist die Abhängigkeiten von zumindest zwei Teilproblemen korrekt sind, wurden hier gleich zwei Teilprobleme zerrissen. Lediglich die Problemgrößen des oberen

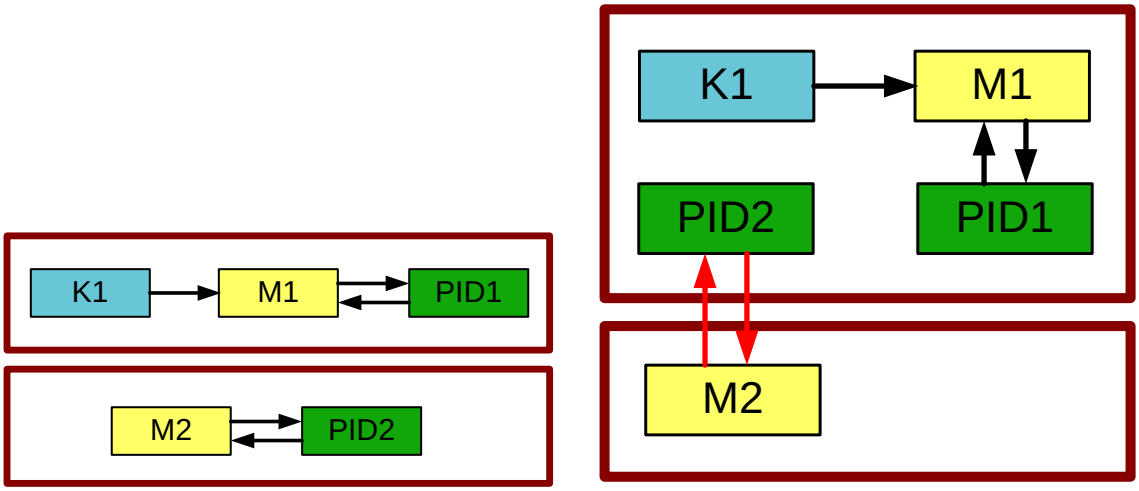

Abbildung 5.12: Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 1. Das linke Bild zeigt die korrekte, das rechte die vom Verfahren ermittelte Partitionierung. Korrekte Abhängigkeiten sind schwarz, nicht korrekte Abhängigkeiten rot markiert.

Teilproblems liegen komplett in einer Partition. Auch die verschiedenen Fitnesswerte liegen leicht hinter denen aus Kapitel 5.2.1, wobei auffällt, dass der Abfall beim Root Mean Squared Error hier eher gering ausfällt.
Die Ergebnisse der Problemzerlegung durch Partitionierung Variante 2 für Datensatz $P4gS$ sind vergleichbar.

Allgemein lässt sich sagen, dass die Time Delay Estimation durch Mutual Information vor allem bei verrauschten Daten oftmals fehlerhafte Werte für die Totzeiten und Tiefpässe liefert und sich dies auch auf die Güte der Problemzerlegung niederschlägt. Die lediglich leicht verschlechterten Werte beim Root Mean Squared Error zeigen jedoch, dass auch die hierdurch erlangten Lösungen noch durchaus brauchbar sind.

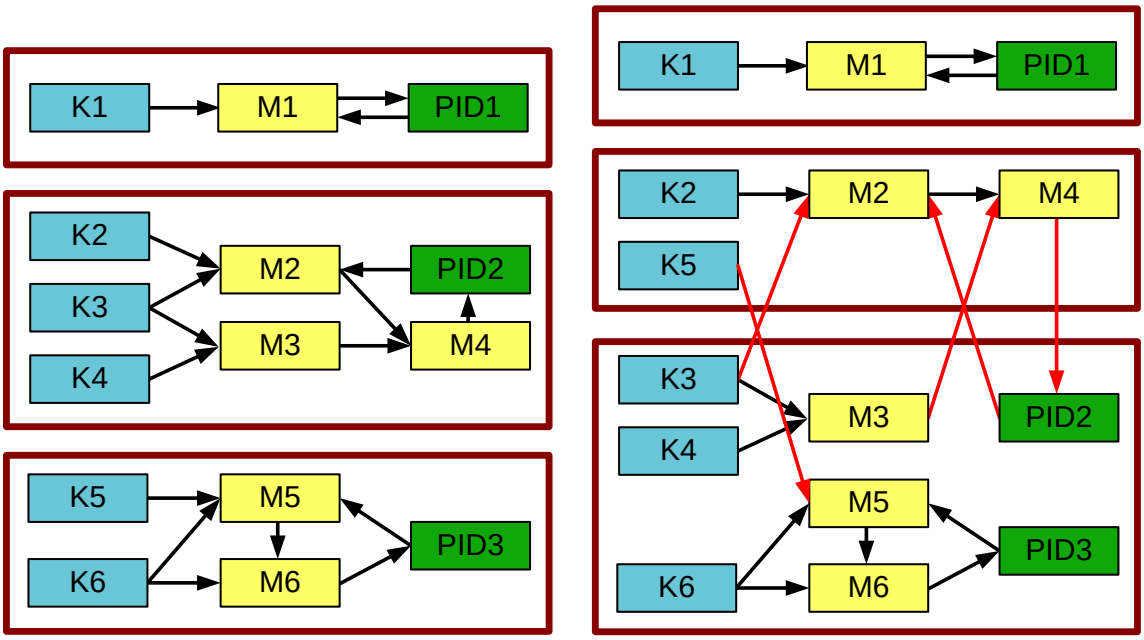

Abbildung 5.13: Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 1. Links ist die korrekte Zerlegung zu sehen, während rechts die vom Verfahren ermittelte Partitionierung abgebildet ist. Korrekte Abhängigkeiten werden durch schwarze, nicht korrekte durch rote Pfeile dargestellt.

5.2.3 Test der Ermittlung der Anzahl der Teilprobleme

Im folgenden Abschnitt werden die Verfahren getestet, wobei sie die Anzahl der Teilprobleme automatisch evolvieren und eine Problemzerlegung in die ermittelte Anzahl Teilprobleme durchführen sollen. Dazu wurde ihnen als Startwert die doppelte Anzahl der real vorhandenen Teilprobleme als Input gegeben, welche diese dann, mit Hilfe der Evolution, auf den korrekten Wert reduzieren sollten. Lediglich die Totzeiten und Tiefpässe werden den Verfahren vorgegeben.

Auch hier soll eine exklusive Problemzerlegung durchgeführt werden, weshalb erneut die Datensätze 1, $1g$ und $4gS$ Anwendung finden. Die Ergebnisse sind in Tabelle 5.17 zu finden. Da die Fitnessberechnungen über die Abhängigkeiten und über die Zugehörigkeit der Problemgrößen hier nicht anwendbar sind, ist hier lediglich der Root Mean

Squared Error der verwendeten neuronalen Netze und, als zusätzliches Gütekriterium für die Lösungen, die ermittelte Anzahl der Teilprobleme eingetragen. Allerdings sei hier noch einmal angemerkt, dass die Fehlerrate der neuronalen Netze und damit auch der Root Mean Squared Error, aufgrund der konzeptionellen Unterschiede, der Anordnung der neuronalen Netze und der Architektur der neuronalen Netze, zwar als Fitnesswert für die Evolution taugt, sich allerdings nur schlecht zum Vergleich der Lösungen der unterschiedlichen Verfahren untereinander eignet.

Datensatz	**Fitness-berechn.**	**Input Pruning**	**Partitionierung Variante 1**	**Partitionierung Variante 2**
P1	Teilprobleme	1	2	2
	F_{RMSE}	$2,48 \cdot 10^{-5}\%$	$5,06 \cdot 10^{-4}\%$	$7,15 \cdot 10^{-4}\%$
P1g	Teilprobleme	1	2	2
	F_{RMSE}	$xxx\%$	$1,56\%$	$3,07\%$
P4gS	Teilprobleme	xxx	3	3
	F_{RMSE}	$xxx\%$	$2,37\%$	$4,62\%$

Tabelle 5.17: Ergebnisse der Standardverfahren

Das Input Pruning Verfahren zerlegt das Problem, zumindest theoretisch, bereits automatisch in Teilprobleme, ohne, dass diese dem Verfahren vorgegeben oder separat durch die Evolution ermittelt werden müssen. Deshalb sind in der Tabelle, zum Vergleich mit den anderen Verfahren, nur noch einmal die Ergebnisse aus Kapitel 5.2.1 eingetragen. Wie auch dort schon beschrieben, eignet sich das Input Pruning Verfahren eher nicht zur Problemzerlegung, da oftmals zu viele, nicht vorhandene Abhängigkeiten ermittelt werden, welche eine Zerlegung in Teilprobleme stark behindern und meist sogar unmöglich machen, wie dies auch für die Testdatensätze der Fall war.

Die beiden Verfahren der Problemzerlegung durch Partitionierung haben die kleineren Testprobleme $P1$ und $P1g$ jeweils zu 100% korrekt zerlegt, wobei sowohl zu der Anzahl

der Teilprobleme als auch der Partitionierung selbstständig Ergebnisse ermittelt wurden, welche den realen Werten entsprechen. Sogar das Testproblem *P4gS* wurde durch die Problemzerlegung durch Partitionierung Variante 1 hier erstmals komplett richtig zerlegt, was in Abbildung 5.14 noch einmal grafisch zu sehen ist. Bemerkenswert ist hier allerdings, dass der Root Mean Squared Error trotz dieser optimalen Zerlegung sogar leicht hinter dem des Ergebnisses aus Kapitel 5.2.1 zurückbleibt. Offenbar ist der Unterschied zu suboptimalen Problemzerlegungen nur sehr gering und auch diese sind durchaus brauchbar.

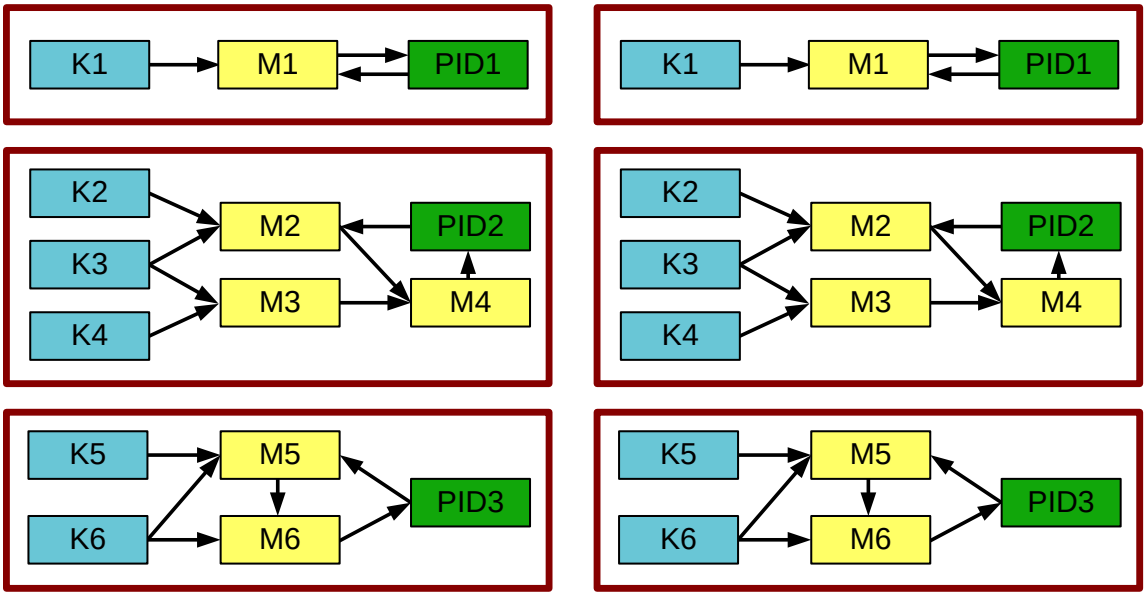

Abbildung 5.14: Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 1. Die linke Grafik zeigt die reale, die rechte die vom Verfahren ermittelte Partitionierung. Korrekte Abhängigkeiten sind schwarz, nicht korrekte Abhängigkeiten rot dargestellt.

Dies stellt für den Datensatz *P4gS* das beste in den Tests erzielte Ergebnis dar. Da objektiv allerdings kein Grund vorhanden ist, dass das Verfahren mit automatischer Ermittlung der Anzahl der Teilprobleme bessere Ergebnisse bringt, als wenn dem Verfahren diese Anzahl vorgegeben wird und somit eigentlich mehr Informationen für die Lösung zur Verfügung stehen, kann dies noch einmal als Bestätigung gesehen

werden, dass die Problemzerlegung durch Partitionierung allgemein auch geeignet ist, größere Probleme korrekt zu zerlegen.

Das Ergebnis der Problemzerlegung durch Partitionierung in der zweiten Variante für den Testdatensatz *P4gS* ist in Abbildung 5.15 zu sehen. Es entspricht qualitativ etwa dem Ergebnis des Verfahrens beim Test mit vorgegebener Anzahl Teilprobleme aus Kapitel 5.2.1, wo ebenfalls das obere und untere der linken Teilprobleme in einer Partitionierung zusammengefasst wurden, während das verbliebene Teilproblem in zwei Partitionen zersplittert ist. Die Anzahl an Partitionen entspricht allerdings korrekt der der tatsächlich vorhandenen Teilprobleme.

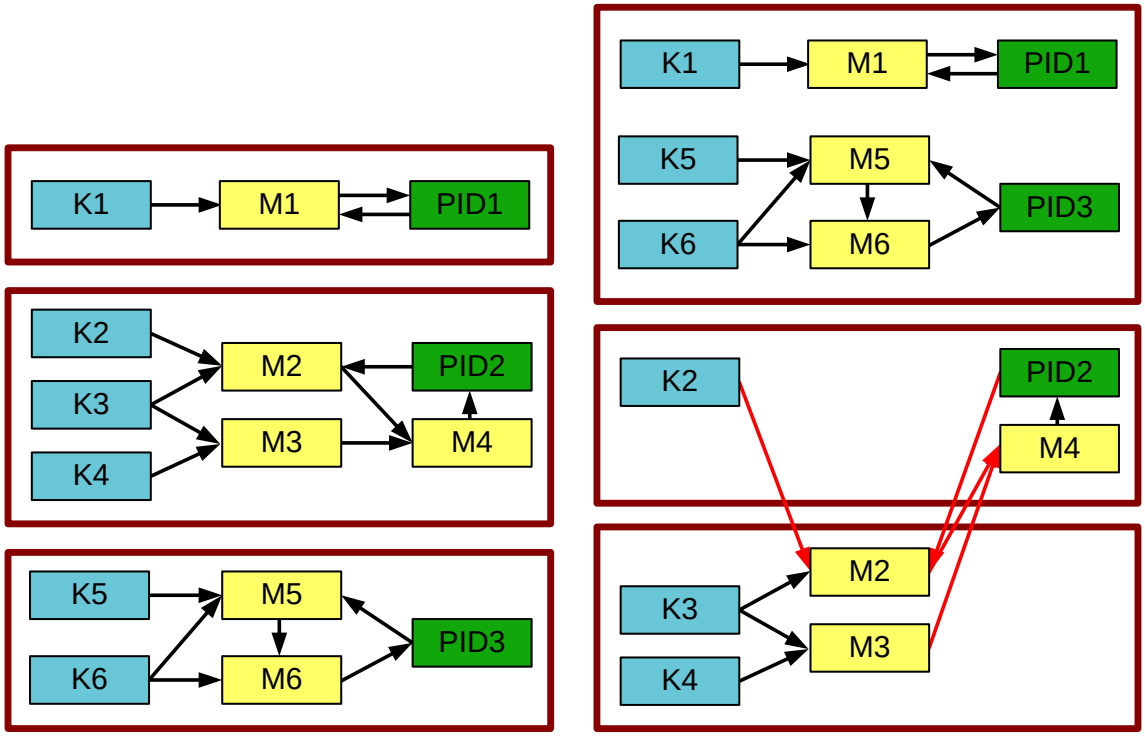

Abbildung 5.15: Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 2. Links ist die korrekte, rechts die vom Verfahren ermittelte Partitionierung zu sehen. Korrekte Abhängigkeiten sind durch schwarze und nicht korrekte Abhängigkeiten durch rote Pfeile dargestellt.

Zusammengefasst lässt sich sagen, dass die automatische Ermittlung der Anzahl der Teilprobleme, mit Ausnahme des Input Pruning Verfahrens, sehr gut funktioniert. Die Anzahl wurde in allen Testfällen korrekt ermittelt und der von der Evolution erst zusätzlich zu ermittelnde Parameter scheint sich in keiner Weise negativ auf die Güte der gefundenen Lösung auszuwirken.

5.2.4 Test der nicht exklusiven Problemzerlegung

Zum Zeitpunkt der Abgabe dieser Masterarbeit standen die Testergebnisse der nicht exklusiven Problemzerlegung und auch die Ergebnisse der Experimente zu den Realweltdaten noch nicht zur Verfügung. Sie werden allerdings, sobald die Berechnungen beendet sind, nachgeliefert werden.

5.3 Vergleich mit anderen Verfahren

Im Folgenden sollen die Ergebnisse der in dieser Arbeit verwendeten Verfahren mit den Ergebnissen aus anderen Verfahren zur Problemzerlegung aus dem APD-Projekt verglichen werden.
Bei der Fertigstellung dieser Arbeit lagen lediglich die Resultate aus [Torke, 2012] vor, wo die Problemdekomposition mit Hilfe von Methoden aus der Informationstheorie, im Einzelnen durch die Mutual Information und die Granger Kausalität, vorgenommen wurde, weshalb hier lediglich ein Vergleich mit diesen angestellt werden kann.

Zum Test der Algorithmen wurden in [Torke, 2012] ebenfalls die bisher hier getesteten Datensätze $P1$ und $P1g$ verwendet. Testdatensatz $P4gS$ findet sich dort jedoch nur in einer erweiterten Variante mit insgesamt 5 Teilproblemen wieder, weshalb die dort beschriebenen Algorithmen zum Vergleich mit den Ergebnissen dieser Arbeit noch einmal mit dem reduzierten Datensatz $P4gS$ getestet wurden. Verwendet worden sind dafür wie in [Torke, 2012] empfohlen ein $Var(12)$-Modell, 2000 Iterationsschritte mit dem Spring Embedder und die Clusterung mit dem

Average Linkage Verfahren. Die Ergebnisse sind weiter unten in Abbildung 5.16 und 5.17 zu sehen.

Da in [Torke, 2012] die Problemgrößen selbst nicht approximiert, sondern nur Problemzerlegungen berechnet wurden, steht der Root Mean Squared Error zur Bewertung der Lösung nicht zur Verfügung und aufgrund der Tatsache, dass dort die Anzahl der Teilprobleme selbstständig berechnet werden und damit stark schwanken können, ist auch eine Fitnessberechnung über die Abhängigkeiten oder über die Zugehörigkeiten der Problemgrößen nicht sinnvoll, weshalb hier ausschließlich ein Vergleich auf intuitiver Basis stattfinden kann. Dazu wurden die in [Torke, 2012] zur Auswertung benutzten Dendrogramme in die in dieser Arbeit verwendeten Grafiken umgewandelt, welche die Problemgrößen mit ihren Abhängigkeiten zeigen.

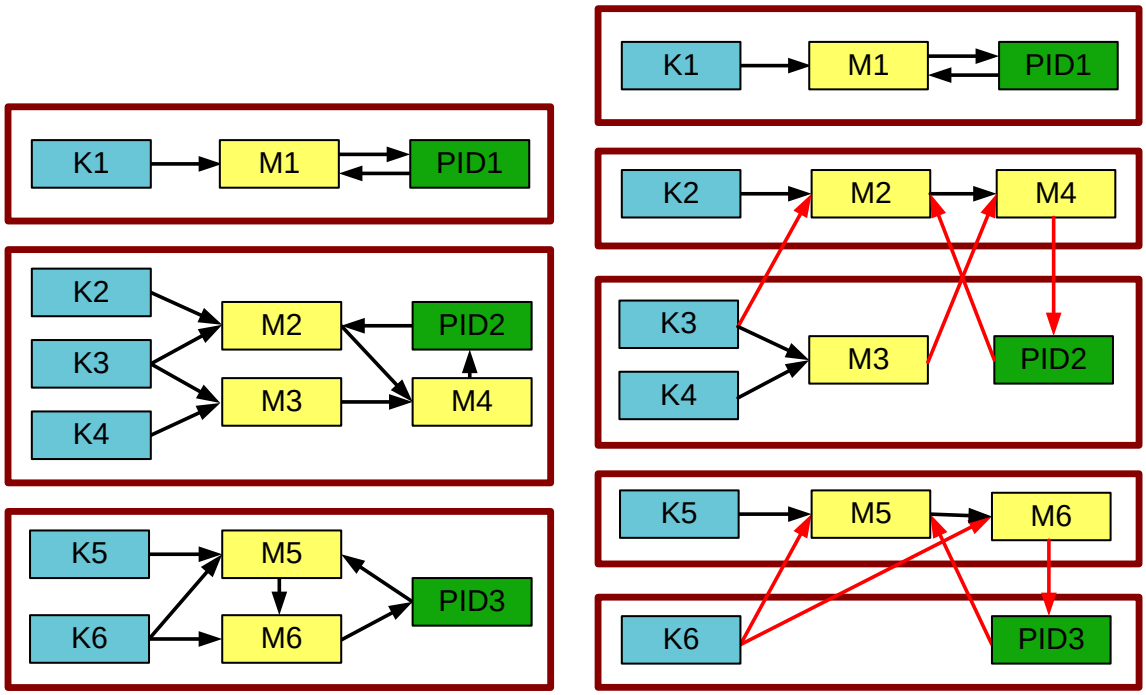

Abbildung 5.16: Problemzerlegung des Testdatensatzes P4gS durch das Mutual Information Verfahren. Auf der linken Seite ist die korrekte, auf der rechten Seite die vom Verfahren ermittelte Partitionierung abgebildet. Korrekte Abhängigkeiten sind dabei schwarz, nicht korrekte Abhängigkeiten rot markiert.

Die kleineren Testdatensätze $P1$ und $P1g$ wurden in [Torke, 2012] von beiden dort verwendeten Verfahren jeweils komplett korrekt zerlegt, was mit der Leistung der Verfahren dieser Arbeit vergleichbar ist und deshalb hier nicht grafisch dargestellt ist. Das Ergebnis für das Mutual Information Verfahren aus [Torke, 2012] für Datensatz $P4gS$ ist in Abbildung 5.16 zu sehen. Auffällig ist hier die Zerlegung in relativ viele Teilprobleme von insgesamt fünf, anstelle der tatsächlich vorhandenen drei Teilprobleme. Dementsprechend werden auch gleich zwei der drei Teilprobleme zerrissen und in jeweils zwei Partitionen verteilt und 8 von 17 Abhängigkeiten bestehen zwischen Problemgrößen verschiedener Teilprobleme, was, im Vergleich zu den in dieser Arbeit verwendeten Verfahren, relativ viele sind.

Bessere Ergebnisse wurden mit dem Granger Kausalität-Verfahren erzielt, mit dessen Hilfe das Problem komplett korrekt zerlegt werden konnte. In Abbildung 5.17 ist dies grafisch dargestellt. Ein solches Ergebnis wurde beim Testdatensatz $P4gS$ mit den Verfahren dieser Arbeit nur ein einziges Mal erzielt, allerdings ohne Time Delay Estimation. Die Totzeiten und Tiefpässe mussten dem Algorithmus dabei erst vorgegeben werden.

Ein weiterer Vorteil der Verfahren aus [Torke, 2012] ist die geringe Ausführungszeit von wenigen Sekunden oder Minuten. Auch müssen diesen bereits in der Standardvariante weder die Werte für die Totzeiten und Tiefpässe noch die Anzahl der Teilprobleme vorgegeben werden.

Der Vorteil der Verfahren dieser Arbeit ist hingegen, dass die Problemgrößen automatisch approximiert und zu diesem Zwecke passende neuronale Netze mit der optimalen Anzahl an Neuronen und Gewichten evolviert werden, was der Konstruktion entsprechender Regler dienlich sein kann. Auch steht auf diese Weise mit der Fehlerrate der Netze ein Qualitätskriterium zur Bewertung und zum Vergleich der ermittelten Problemzerlegung zur Verfügung, was den Verfahren aus [Torke, 2012] fehlt.

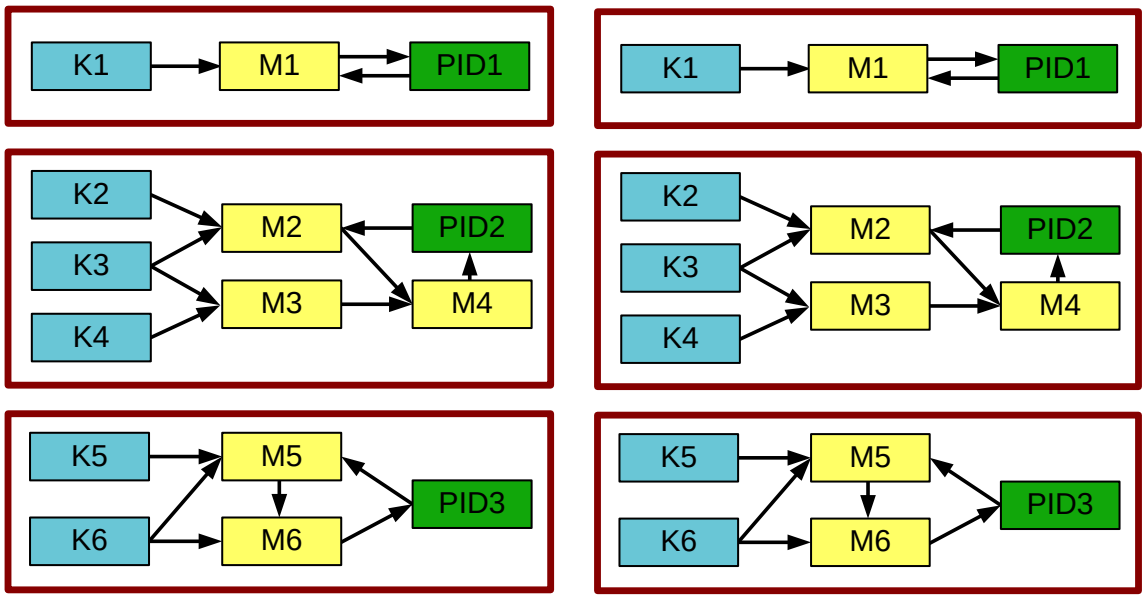

Abbildung 5.17: Problemzerlegung des Testdatensatzes P4gS durch das Granger Kausalität-Verfahren. Links ist die korrekte, rechts die vom Verfahren ermittelte Partitionierung zu sehen. Korrekte Abhängigkeiten sind durch schwarze und nicht korrekte Abhängigkeiten durch rote Pfeile dargestellt.

5.4 Zusammenfassung experimentelle Untersuchungen

In diesem Kapitel wurden zuerst die zum Test benutzten Datensätze beschrieben, dann die verwendeten Parameter für die getesteten Algorithmen gezeigt und anschließend die drei in dieser Arbeit beschriebenen Verfahren zur Problemzerlegung in verschiedenen Varianten an den Testdatensätzen getestet.

Dies war zum Einen der Test in der Standardvariante, wo die Verfahren ohne Erweiterungen zur Anwendung kamen und die Werte für die Totzeiten und Tiefpässe und die Anzahl der Teilprobleme vorgegeben wurden. Weiterhin wurden sie mit vorheriger Time Delay Estimation getestet, durch welche die Werte für die Totzeiten und Tiefpässe berechnet werden sollten. Diese Werte wurden anschließend von den Verfahren zur Problemzerlegung verwendet, wobei die Anzahl der Teilprobleme weiterhin vorgege-

ben wurde. Schließlich erfolgte ein Test der automatischen Ermittlung der Anzahl der Teilprobleme, in dem die Verfahren im Laufe der Evolution diese Anzahl automatisch evolvieren und eine Problemdekomposition in diese Anzahl Teilprobleme vornehmen sollten, wobei die Werte für die Totzeiten und Tiefpässe den Verfahren allerdings wieder zur Verfügung gestellt wurden.
Gegen Ende fand ein Vergleich der Verfahren dieser Arbeit mit denen aus [Torke, 2012] statt, wo die Problemzerlegung mit Hilfe von Mitteln aus der Informationstheorie, wie der Mutual Information und der Granger Kausalität, vorgenommen wurde.

Kapitel 6

Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war es, Verfahren zu entwickeln und zu untersuchen, mit denen komplexe Regelsysteme in mehrere kleine Teilprobleme zu zerlegt werden können, welche besser beherrschbar und überschaubarer als das Gesamtproblem sind. Dazu sollten evolutionäre Algorithmen verwendet werden.
Zu diesem Zwecke wurden drei Verfahren entworfen: Die Problemzerlegung durch Input Pruning, welche durch das evolutionäre Input Pruning die Abhängigkeiten der Problemgrößen ermittelt und mit einem Zusammenhangstest für den entstandenen Abhängigkeitsgraphen die Teilprobleme identifizieren kann. Und außerdem zwei Varianten der Problemzerlegung durch Partitionierung, bei denen jeweils die Problemgrößen in mehrere Partitionen zerlegt werden, welche den Teilproblemen entsprechen. Durch eine Bewertung der Partitionierung durch die Approximation der Problemgrößen mittels neuronaler Netze kann eine Evolution vorgenommen werden, welche schließlich in der optimalen Zerlegung resultiert. Die beiden Varianten unterscheiden sich dadurch, dass die Problemgrößen in der ersten Variante durch ein neuronales Netz für jede Partition, und in der zweiten Variante jeweils einzeln durch ein eigenes neuronales Netz approximiert werden.
Zusätzlich wurden einige Erweiterungsmöglichkeiten vorgestellt, um den Verfahren zum Einen zu ermöglichen, die Werte für die Totzeiten und Tiefpässe der Abhängigkeiten des Problems zu ermitteln, und zum Anderen die Anzahl der vorhandenen Teilprobleme automatisch zu evolvieren. Weiterhin wurde gezeigt, wie mit Hilfe

der Verfahren auch eine nicht exklusive Problemzerlegung vorgenommen werden kann. Schließlich wurden die Verfahren, jeweils mit und ohne Erweiterungen, anhand mehrerer künstlich erzeugter Datensätze auf ihre Tauglichkeit getestet.

Als Ergebnis zeigte sich, dass die Problemzerlegung durch Input Pruning im Allgemeinen zu viele nicht vorhandenen Abhängigkeiten ermittelt, welche oftmals zwischen Problemgrößen verschiedener Teilprobleme bestehen. Dies macht eine Problemzerlegung unmöglich und das Verfahren ist daher für diese Aufgabe eher ungeeignet. Die beiden Verfahren zur Problemzerlegung durch Partitionierung funktionieren bei kleinen Problemen sehr gut, nur mit zunehmender Anzahl an Problemgrößen und Abhängigkeiten bleiben sie zunehmend in lokalen Optima stecken, wo real vorhandene Teilprobleme teilweise auseinandergerissen und teilweise mehrere zu einem Teilproblem zusammengefasst werden. Die erste Variante der Problemzerlegung durch Partitionierung tendiert außerdem dazu, einzelne Partitionen mit nur einer einzigen Problemgröße zu erstellen, welche dann durch sich selbst approximiert wird, was als Problemzerlegung wenig Sinn macht. Es zeigte sich jedoch, dass selbst der Unterschied im Root Mean Squared Error der neuronalen Netze, welche die Problemgrößen approximieren, zwischen der optimalen Lösung und den meisten suboptimalen Problemzerlegungen oftmals nur sehr gering ist, diese also wahrscheinlich durchaus ebenfalls brauchbare Lösungen darstellen.
Weiterhin wurde offenbar, dass, vor allem bei Datensätzen mit hohem Rauschen, die Berechnung der Werte für die Totzeiten und Tiefpässe der Abhängigkeiten durch die Time Delay Estmation teilweise fehlerhafte Ergebnisse hervorbringt und sich dies in geringem Maße auch auf die Problemzerlegung negativ auswirkt.
Die automatische Berechnung der Anzahl der Teilprobleme dagegen erbrachte durchgehend sehr gute Ergebnisse. Sowohl bei der Anzahl der Teilprobleme als auch bei der damit durchgeführten Problemzerlegung ließ sich qualitativ kein Unterschied zu den Ergebnissen mit den Standardverfahren feststellen, wo die Anzahl der Teilprobleme vorgegeben wurde.

Als weiterführende Arbeit wäre es möglich, alternative Input Pruning Ansätze zu ver-

wenden, da sich bei der Evolution der Inputs der neuronalen Netze zwar bei fehlenden Abhängigkeiten die Fehlerrate deutlich verschlechtert, bei nicht vorhandenen Abhängigkeiten und damit zu vielen Inputs der Einfluss auf die Fehlerrate allerdings eher gering ist und so, wie hier beobachtet, oftmals zu viele Abhängigkeiten ermittelt werden. Auch Alternativen zur Time Delay Estimation, welche die Werte für die Totzeiten und Tiefpässe berechnet, sind denkbar, beispielsweise mit der Verwendung von Time Delay Neural Networks oder rekurrenten Netzen. Schließlich ist es möglich, auch ein Input Pruning für die neuronalen Netze bei der Problemzerlegung durch Partitionierung zu verwenden und so überflüssige Abhängigkeiten selbst innerhalb der Partitionen zu eliminieren. Dies würde die Fehlerrate weiter verbessern und sich somit positiv auf die Fitnessberechnung, damit die Evolution, aber auch auf die eventuell nachfolgende Verwendung als Regler auswirken.

Anhang A

Totzeiten und Tiefpässe der Testdatensätze

Gezeigt werden die Tabellen mit den Totzeiten und Tiefpässen der verwendeten künstlich erzeugten Testdatensätze. Der Zahlenwert des Tiefpasses gibt dabei die Zahl der vergangenen Zeiteinheiten an, über welche die Größe gemittelt wurde. Eine Totzeit von 3 und ein Tiefpass mit dem Wert von 2 bei einer Abhängigkeit zwischen $K9$ und $M7$ würde beispielsweise bedeuten, dass die $M7(t)$ vom Mittelwert von $K9(t-3)$ und $K9(t-4)$ abhängig ist.

A.1 Testdatensatz P1

Abhängigkeit	Totzeit	Tiefpass
K1 → M1	1	1
K2 → M1	1	1
K3 → M2	1	1

Tabelle A.1: Totzeiten und Tiefpässe des Datensatzes $P1$

A.2 Testdatensatz P1g

Abhängigkeit	Totzeit	Tiefpass
K1 → M1	1	1
M1 → PID1	1	1
M2 → PID2	1	1
PID1 → M1	1	1
PID2 → M2	1	1

Tabelle A.2: Totzeiten und Tiefpässe des Datensatzes $P1g$

A.3 Testdatensatz P2

Abhängigkeit	Totzeit	Tiefpass
K1 → M1	1	1
K2 → M1	1	1
K3 → M1	1	1
K3 → M2	1	1

Tabelle A.3: Totzeiten und Tiefpässe des Datensatzes $P2$

A.4 Testdatensatz P2g

Abhängigkeit	Totzeit	Tiefpass
K1 → M1	1	1
M1 → PID1	1	1
M2 → PID2	1	1
PID1 → M1	1	1
PID2 → M2	1	1
PID2 → M1	1	1

Tabelle A.4: Totzeiten und Tiefpässe des Datensatzes $P2g$

A.5 Testdatensatz P3

Abhängigkeit	Totzeit	Tiefpass
K1 → M1	1	1
K1 → M2	1	5
K2 → M1	1	1
K2 → M2	1	1
K3 → M1	1	1
K3 → M2	1	1
M1 → M2	1	5
M2 → M1	1	1

Tabelle A.5: Totzeiten und Tiefpässe des Datensatzes $P3$

A.6 Testdatensatz P3g

Abhängigkeit	Totzeit	Tiefpass
K1 → M1	1	1
K1 → M2	1	5
M1 → M2	1	5
M1 → PID1	1	1
M2 → M1	1	1
M2 → PID2	1	1
PID1→ M1	1	1
PID1 → M2	1	1
PID2→ M1	1	1
PID2 → M2	1	1

Tabelle A.6: Totzeiten und Tiefpässe des Datensatzes $P3g$

A.7 Testdatensatz P4gS

Abhängigkeit	Totzeit	Tiefpass
K1 → M1	2	3
K2 → M2	5	3
K3 → M2	1	2
K3 → M3	2	5
K4 → M3	5	2
K5 → M5	2	3
K6 → M5	5	2
K6 → M6	2	3
M1→ PID1	1	1
M2 → M4	2	3
M3 → M4	1	5
M4 → PID2	1	1
M5 → M6	5	2
M6 → PID3	1	1
PID1 → M1	5	2
PID2 → M2	5	3
PID3 → M5	5	2

Tabelle A.7: Totzeiten und Tiefpässe des Datensatzes *P4gS*

A.8 Testdatensatz P9g

Abhängigkeit	Totzeit	Tiefpass
K1 → M1	2	7
K1 → M2	2	7
K1 → M3	1	5
K2 → M1	2	3
K3 → M2	4	4
K4 → M2	5	3
K5 → M3	2	4
K5 → M4	2	4
K6 → M5	2	4
M1 → PID1	1	1
M2 → PID2	1	1
M3 → PID3	1	1
M4 → M3	3	3
M5 → PID4	1	1
PID1 → M1	5	2
PID2 → M2	2	3
PID3 → M4	2	2
PID4 → M5	2	2

Tabelle A.8: Totzeiten und Tiefpässe des Datensatzes $P9g$

Anhang B

Ergebnisse der Tests der Verfahren

Im Folgenden sind die Ergebnisse der Tests grafisch und tabellarisch dargestellt.
Die Grafiken zeigen auf der linken Seite die korrekte Partitionierung mit den korrekten Abhängigkeiten, während rechts die vom Verfahren ermittelten Ergebnisse zu sehen sind.
Beim Input Pruning Verfahren sind dabei alle ermittelten Abhängigkeiten abgebildet, wobei die korrekten Abhängigkeiten schwarz, die in der Realität nicht vorhandenen Abhängigkeiten dagegen rot dargestellt sind. Die aus den Abhängigkeiten entstandenen Zusammenhangskomponenten sind jeweils dunkelrot umrandet.
Bei der Problemzerlegung durch Partitionierung in beiden Varianten, welche die Abhängigkeiten nicht explizit ermitteln, sondern nur die Partitionierung und damit die Problemzerlegung, sind die Partitionen dunkelrot umrandet. Die Abhängigkeiten wurden mit Hilfe des Vorwissens über die Testdatensätze nachträglich eingezeichnet, wobei korrekte Abhängigkeiten, welche zwischen zwei Problemgrößen einer Partition bestehen, schwarz und nicht korrekte Abhängigkeiten, welche zwischen Problemgrößen verschiedener Partitionen bestehen, rot eingefärbt sind.
In den Tabellen sind von links nach rechts die Art der Fitnessberechnung nach Kapitel 4.1, die, je nach Fitnessberechnung, Anzahl korrekter Abhängigkeiten oder Anzahl korrekt zugeordneter Problemgrößen, die Anzahl der insgesamt vorhandenen Abhängigkeiten oder Problemgrößen und schließlich der Fitnesswert abgetragen.

B.1 Test der Standardverfahren

B.1.1 Problemzerlegung durch Input Pruning

Testdatensatz $P1$

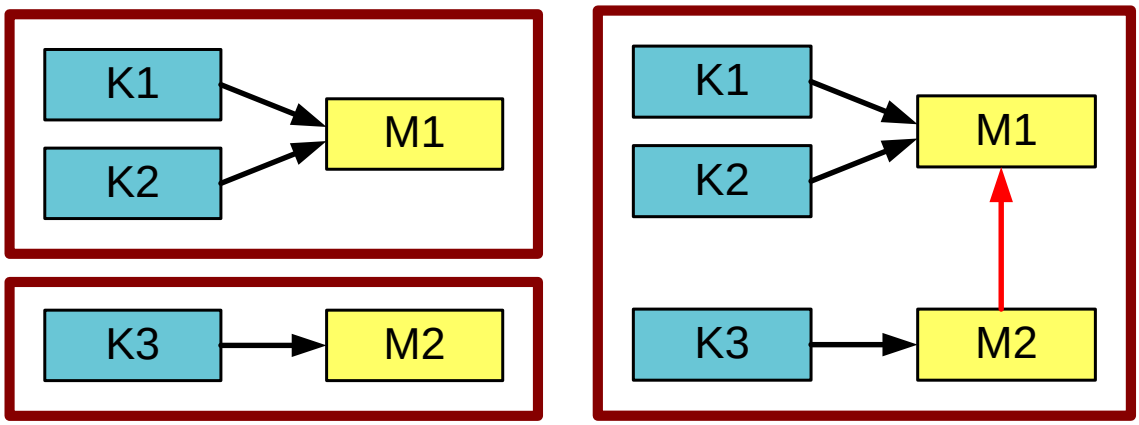

Abbildung B.1: Problemzerlegung des Testdatensatzes P1 durch Input Pruning.

Fitnessberechnung	Wert
F_{RMSE}	$2,48 \cdot 10^{-5}\%$

Tabelle B.1: Fitness des Datensatzes $P1$

Testdatensatz $P1g$

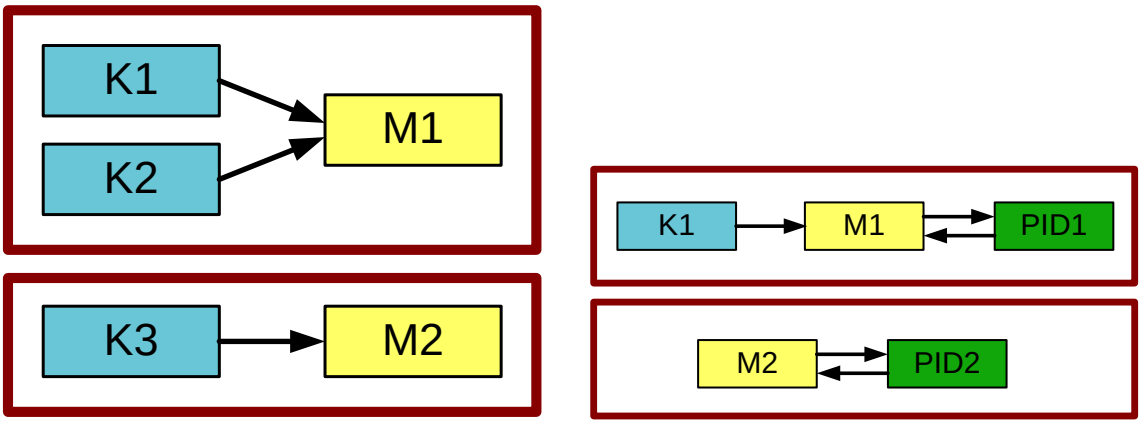

Abbildung B.2: Problemzerlegung des Testdatensatzes P1g durch Input Pruning.

Fitnessberechnung	Wert
F_{RMSE}	$2,82\%$

Tabelle B.2: Fitness des Datensatzes $P1g$

B.1.2 Problemzerlegung durch Partitionierung Variante 1

Testdatensatz $P1$

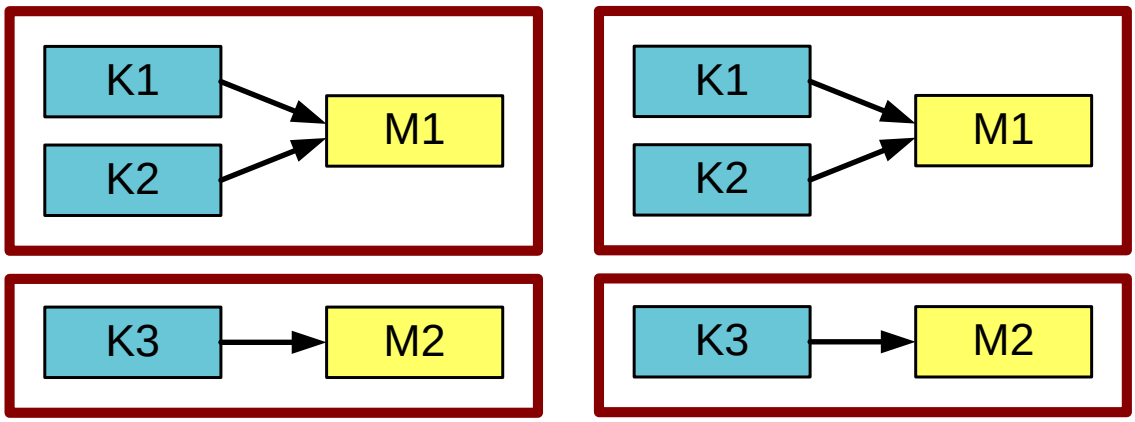

Abbildung B.3: Problemzerlegung des Testdatensatzes P1 durch Partitionierung Variante 1.

Fitnessberechnung	korrekt	Gesamt	Wert
F_{PG}	5	5	100%
F_{Abh}	3	3	100%
F_{RMSE}			$2,24 \cdot 10^{-4}\%$

Tabelle B.3: Fitness des Datensatzes $P1$

Testdatensatz $P1g$

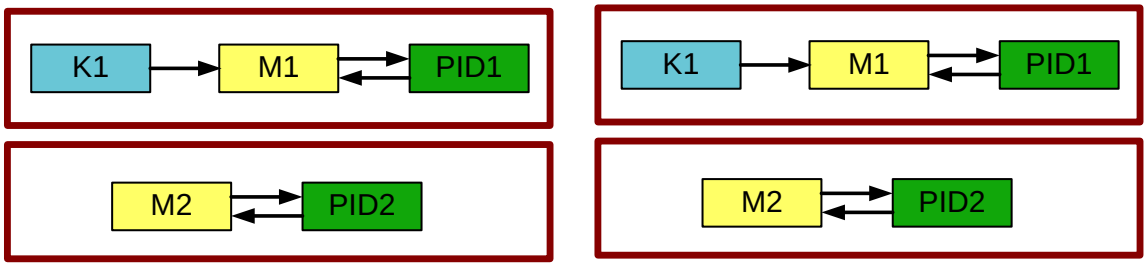

Abbildung B.4: Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 1.

Fitnessberechnung	korrekt	Gesamt	Wert
F_{PG}	5	5	100%
F_{Abh}	5	5	100%
F_{RMSE}			$1,56\%$

Tabelle B.4: Fitness des Datensatzes $P1g$

Testdatensatz *P4gS*

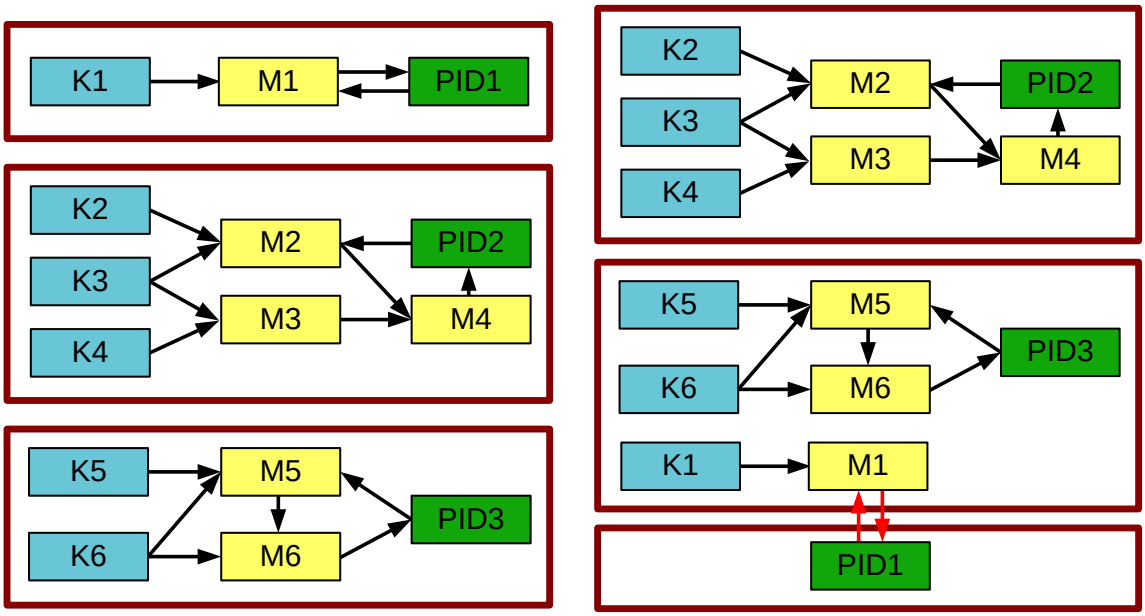

Abbildung B.5: Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 1.

Fitnessberechnung	korrekt	Gesamt	Wert
F_{PG}	13	15	$86,7\%$
F_{Abh}	15	17	$88,2\%$
F_{RMSE}			$2,02\%$

Tabelle B.5: Fitness des Datensatzes *P4gS*

B.1.3 Problemzerlegung durch Partitionierung Variante 2

Testdatensatz $P1$

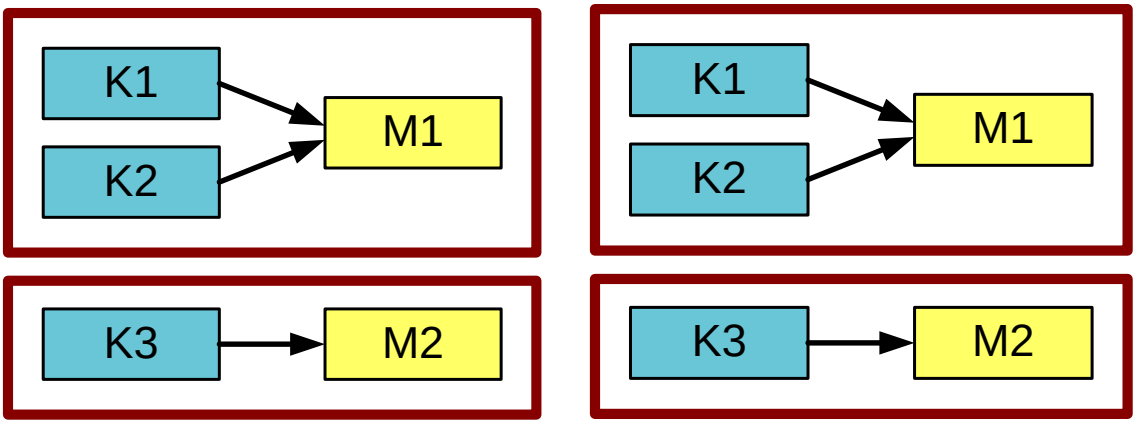

Abbildung B.6: Problemzerlegung des Testdatensatzes P1 durch Partitionierung Variante 2.

Fitnessberechnung	korrekt	Gesamt	Wert
F_{PG}	5	5	100%
F_{Abh}	3	3	100%
F_{RMSE}			$8,74 \cdot 10^{-4}\%$

Tabelle B.6: Fitness des Datensatzes $P1$

Testdatensatz $P1g$

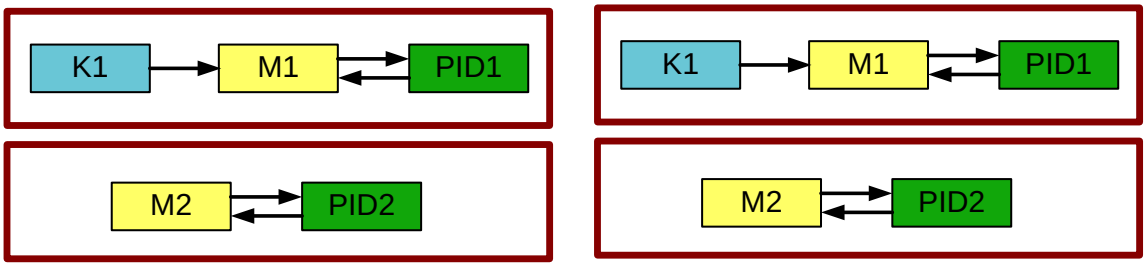

Abbildung B.7: Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 2.

Fitnessberechnung	korrekt	Gesamt	Wert
F_{PG}	5	5	100%
F_{Abh}	5	5	100%
F_{RMSE}			$3,02\%$

Tabelle B.7: Fitness des Datensatzes $P1g$

Testdatensatz *P4gS*

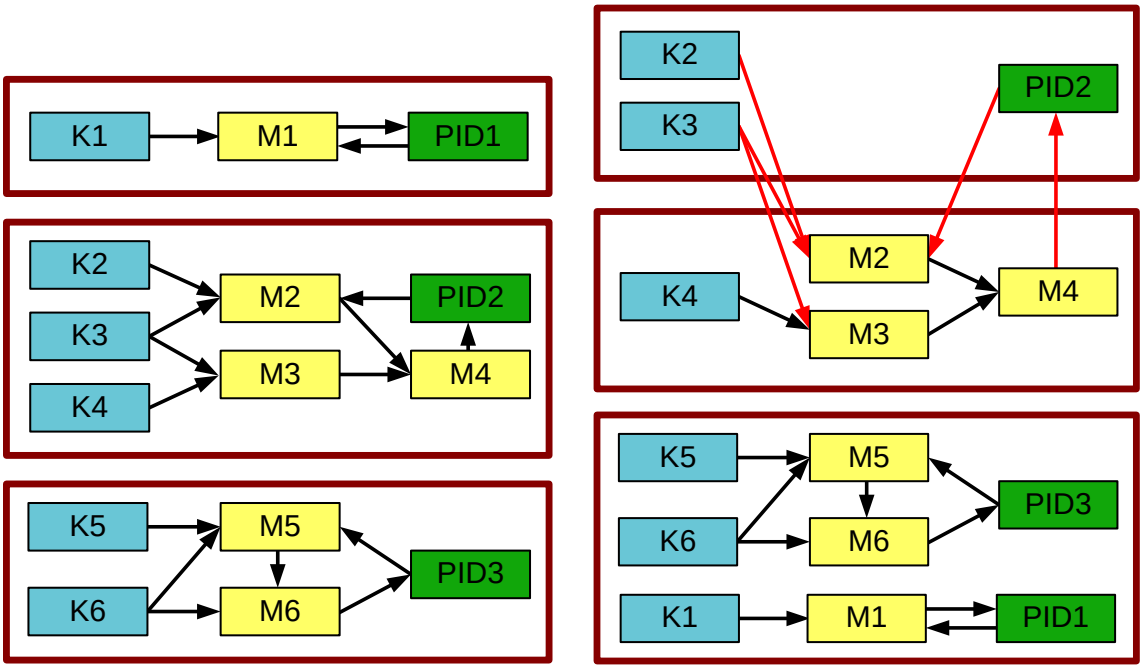

Abbildung B.8: Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 2.

Fitnessberechnung	korrekt	Gesamt	Wert
F_{PG}	9	15	$60,0\%$
F_{Abh}	12	17	$70,6\%$
F_{RMSE}			$5,25\%$

Tabelle B.8: Fitness des Datensatzes *P4gS*

B.2 Test der Time Delay Estimation

B.2.1 Problemzerlegung durch Input Pruning

Testdatensatz $P1$

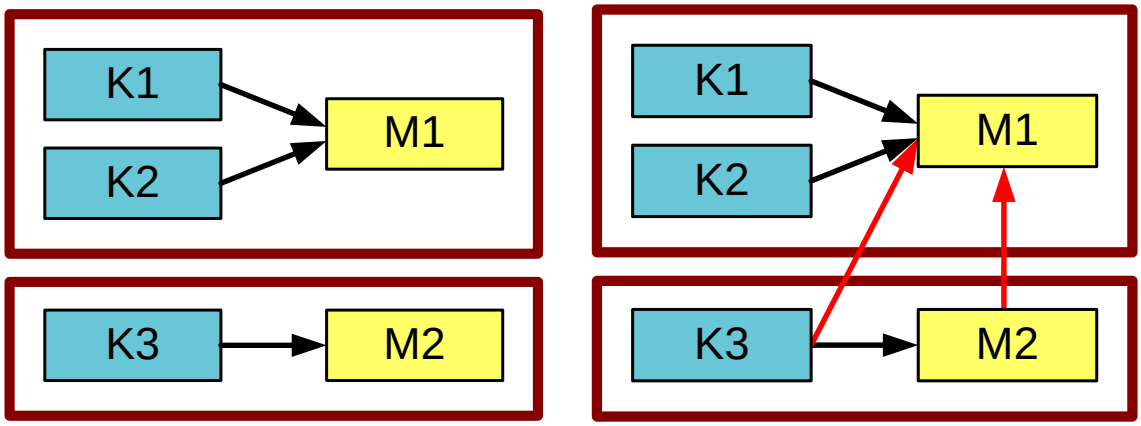

Abbildung B.9: Problemzerlegung des Testdatensatzes P1 durch Input Pruning.

Fitnessberechnung	Wert
F_{RMSE}	$1,79 \cdot 10^{-5}\%$

Tabelle B.9: Fitness des Datensatzes $P1$

Testdatensatz $P1g$

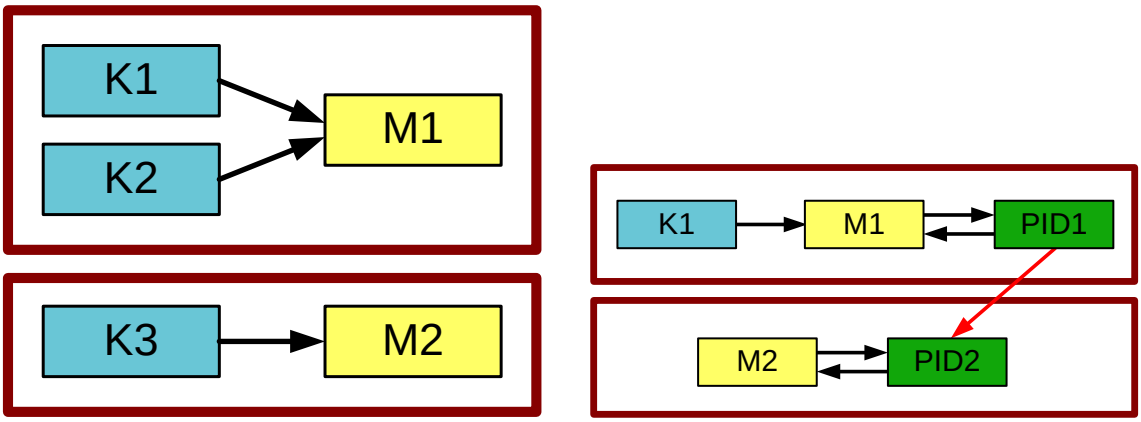

Abbildung B.10: Problemzerlegung des Testdatensatzes P1g durch Input Pruning.

Fitnessberechnung	Wert
F_{RMSE}	$4,20\%$

Tabelle B.10: Fitness des Datensatzes $P1g$

B.2.2 Problemzerlegung durch Partitionierung Variante 1

Testdatensatz $P1$

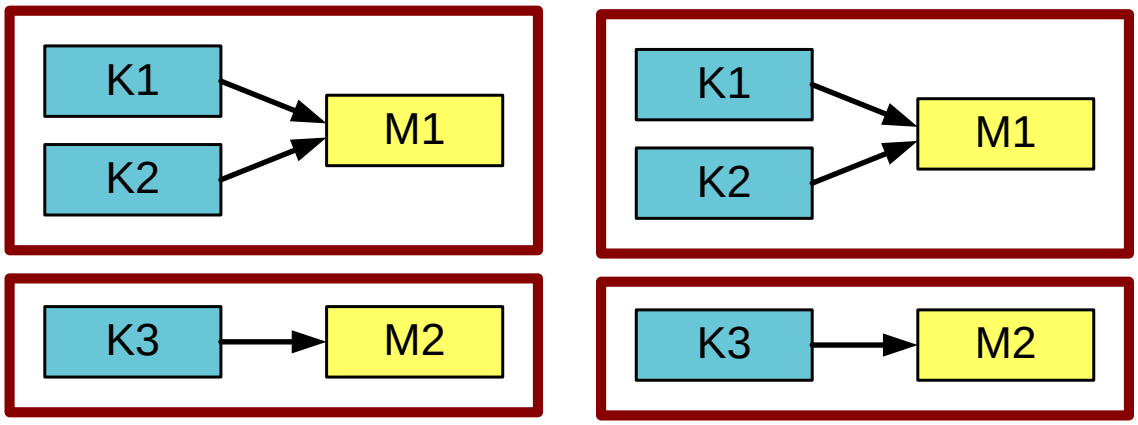

Abbildung B.11: Problemzerlegung des Testdatensatzes P1 durch Partitionierung Variante 1.

Fitnessberechnung	korrekt	Gesamt	Wert
F_{PG}	5	5	100%
F_{Abh}	3	3	100%
F_{RMSE}			$4,20 \cdot 10^{-4}\%$

Tabelle B.11: Fitness des Datensatzes $P1$

Testdatensatz $P1g$

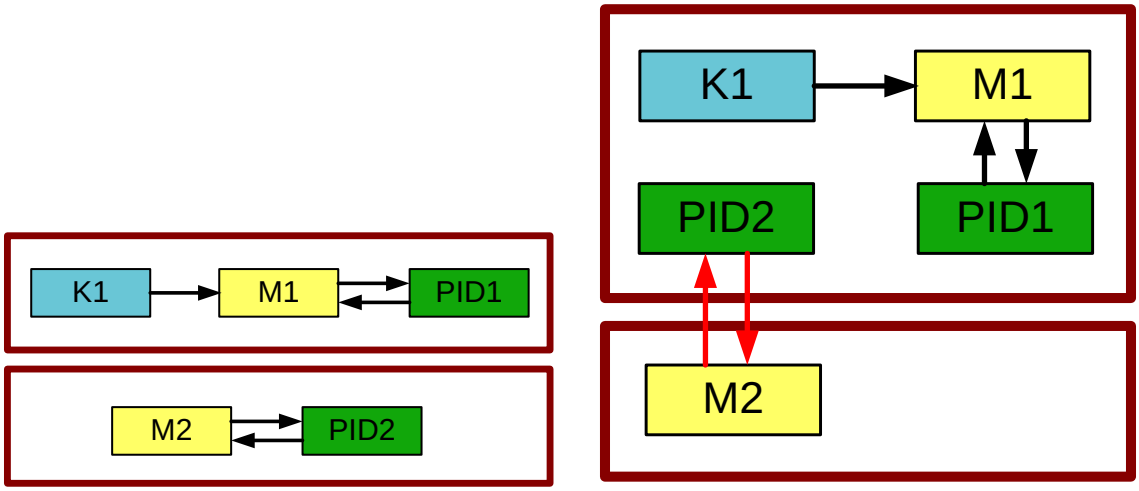

Abbildung B.12: Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 1.

Fitnessberechnung	korrekt	Gesamt	Wert
F_{PG}	4	5	80%
F_{Abh}	3	5	60%
F_{RMSE}			$2,95\%$

Tabelle B.12: Fitness des Datensatzes $P1g$

Testdatensatz *P4gS*

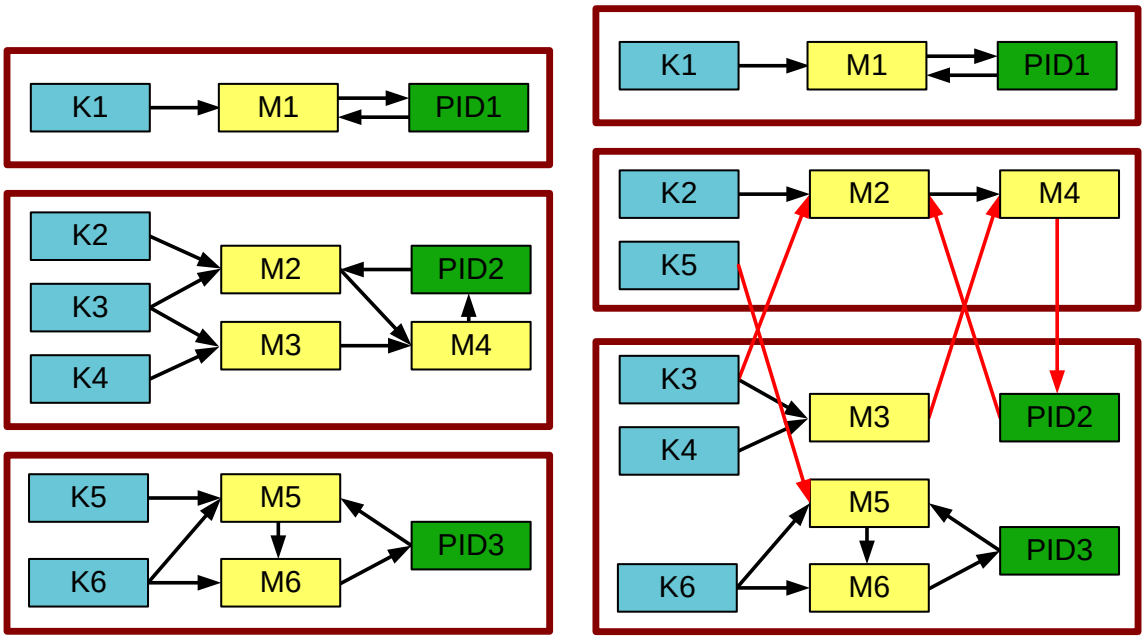

Abbildung B.13: Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 1.

Fitnessberechnung	korrekt	Gesamt	Wert
F_{PG}	10	15	$66,7\%$
F_{Abh}	12	17	$70,6\%$
F_{RMSE}			$2,37\%$

Tabelle B.13: Fitness des Datensatzes *P4gS*

B.2.3 Problemzerlegung durch Partitionierung Variante 2

Testdatensatz $P1$

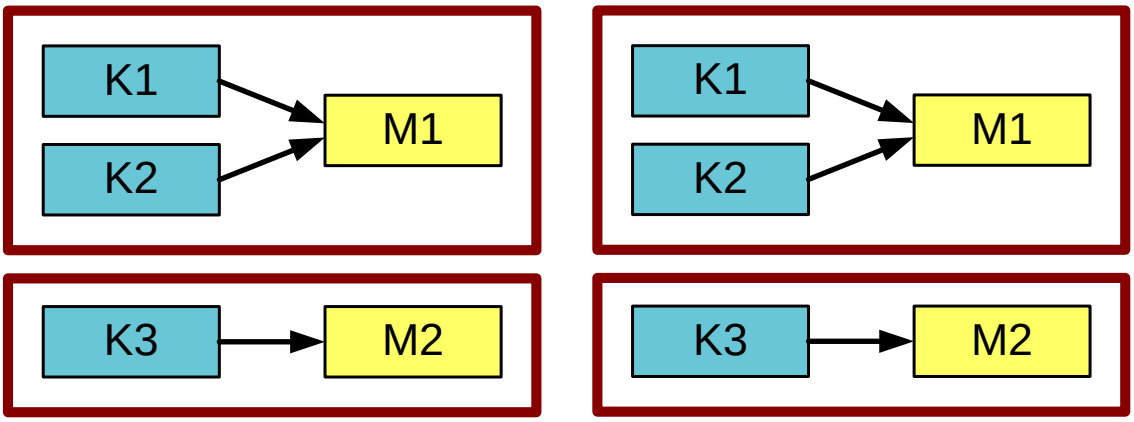

Abbildung B.14: Problemzerlegung des Testdatensatzes P1 durch Partitionierung Variante 2.

Fitnessberechnung	korrekt	Gesamt	Wert
F_{PG}	5	5	100%
F_{Abh}	3	3	100%
F_{RMSE}			$1,37 \cdot 10^{-3}\%$

Tabelle B.14: Fitness des Datensatzes $P1$

Testdatensatz $P1g$

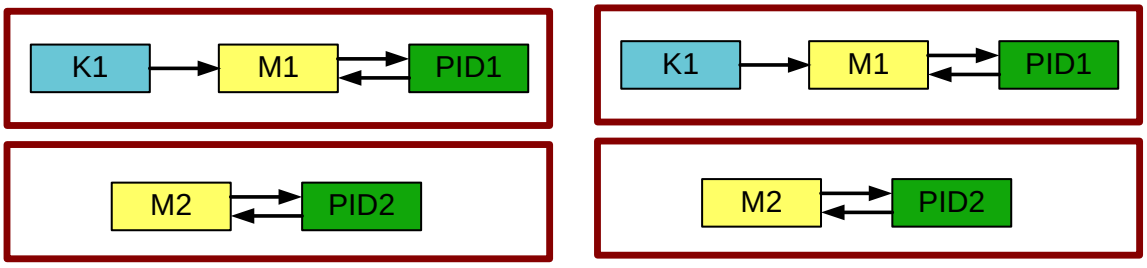

Abbildung B.15: Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 2.

Fitnessberechnung	korrekt	Gesamt	Wert
F_{PG}	5	5	100%
F_{Abh}	5	5	100%
F_{RMSE}			$4,86\%$

Tabelle B.15: Fitness des Datensatzes $P1g$

Testdatensatz *P4gS*

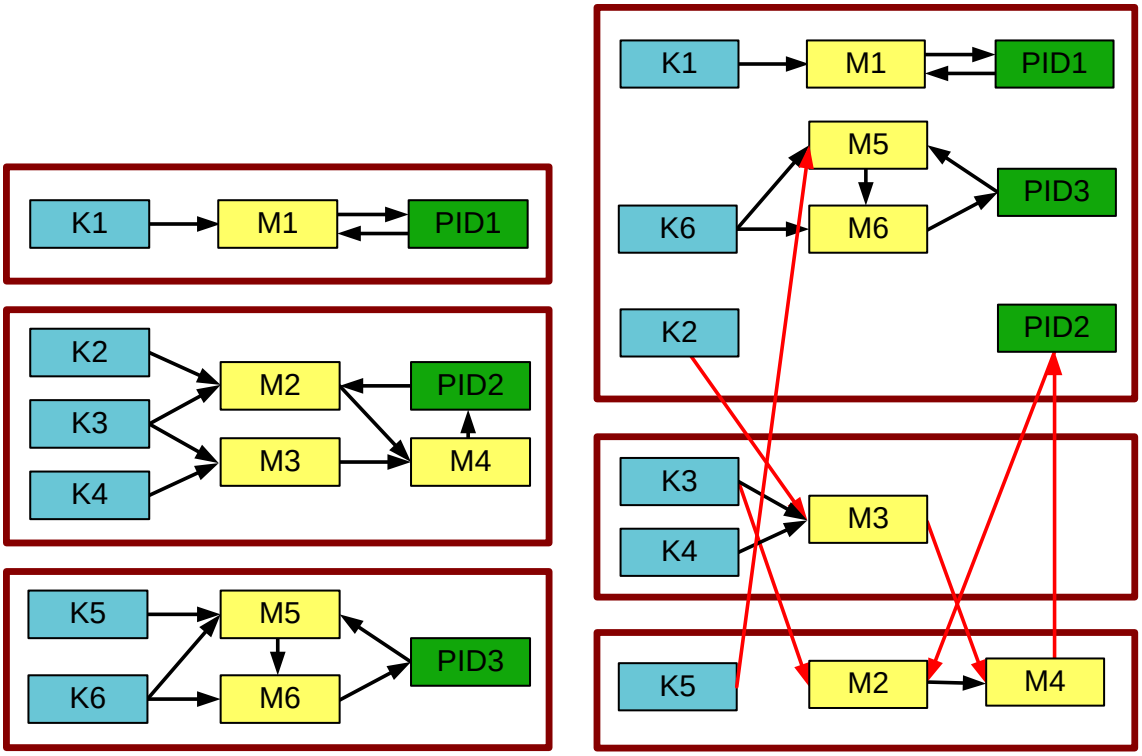

Abbildung B.16: Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 2.

Fitnessberechnung	korrekt	Gesamt	Wert
F_{PG}	7	15	$46,7\%$
F_{Abh}	11	17	$64,7\%$
F_{RMSE}			$4,39\%$

Tabelle B.16: Fitness des Datensatzes $P4gS$

B.3 Test der Ermittlung der Anzahl der Teilprobleme

B.3.1 Problemzerlegung durch Partitionierung Variante 1

Testdatensatz $P1$

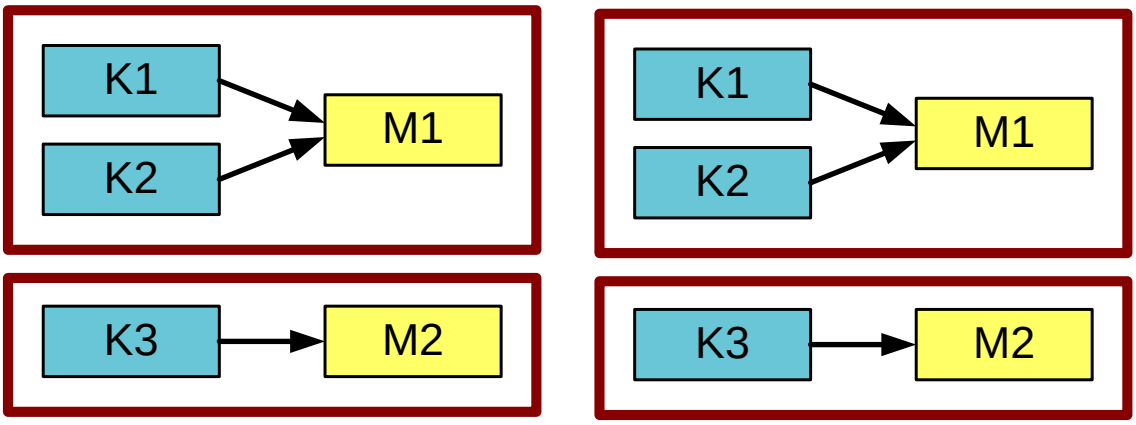

Abbildung B.17: Problemzerlegung des Testdatensatzes P1 durch Partitionierung Variante 1.

Fitnessberechnung	Wert
Anzahl Teilprobleme	2
F_{RMSE}	$5,06 \cdot 10^{-4}\%$

Tabelle B.17: Fitness des Datensatzes $P1$

Testdatensatz $P1g$

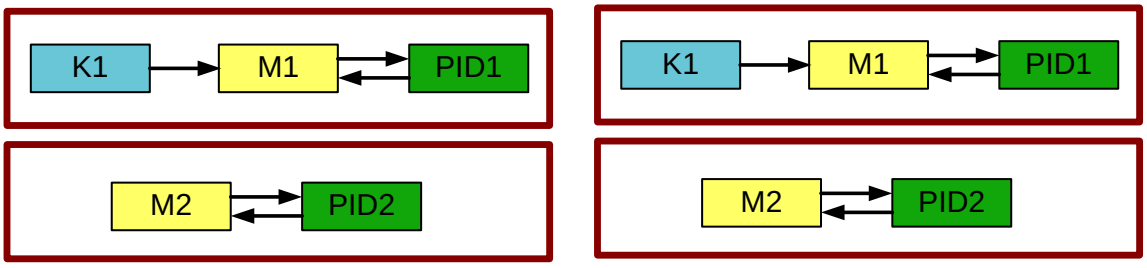

Abbildung B.18: Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 1.

Fitnessberechnung	Wert
Anzahl Teilprobleme	2
F_{RMSE}	$1,56\%$

Tabelle B.18: Fitness des Datensatzes $P1g$

Testdatensatz *P4gS*

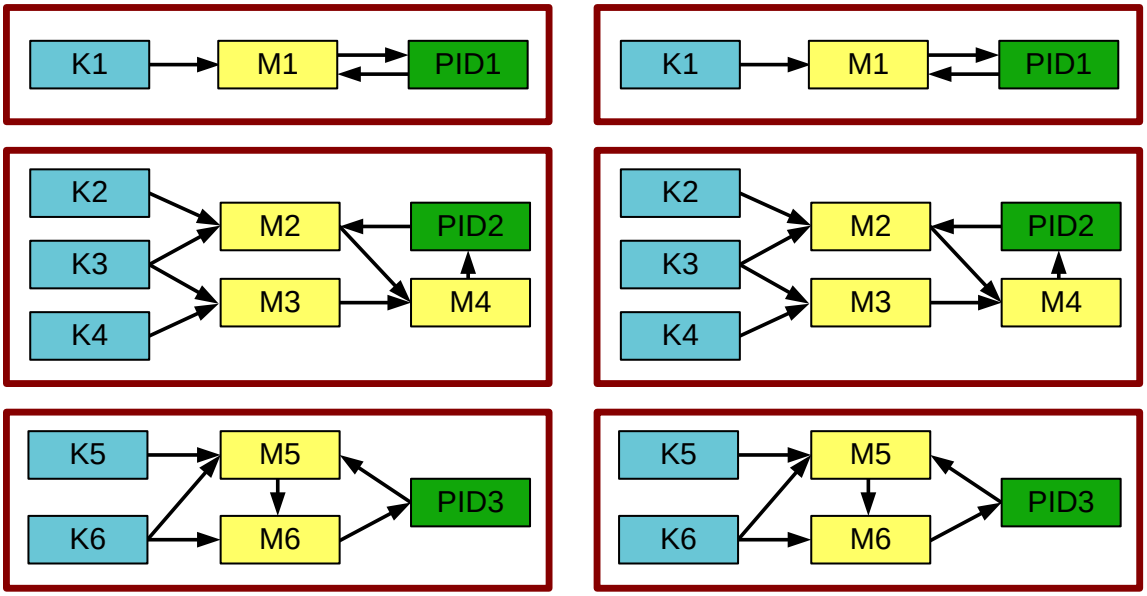

Abbildung B.19: Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 1.

Fitnessberechnung	Wert
Anzahl Teilprobleme	3
F_{RMSE}	$2,37\%$

Tabelle B.19: Fitness des Datensatzes *P4gS*

B.3.2 Problemzerlegung durch Partitionierung Variante 2

Testdatensatz $P1$

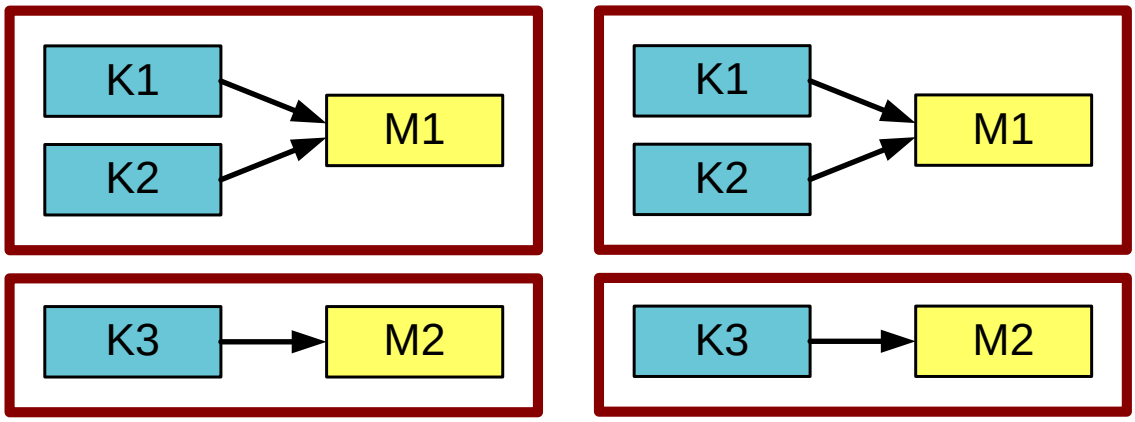

Abbildung B.20: Problemzerlegung des Testdatensatzes P1 durch Partitionierung Variante 2.

Fitnessberechnung	Wert
Anzahl Teilprobleme	2
F_{RMSE}	$7,15 \cdot 10^{-4}\%$

Tabelle B.20: Fitness des Datensatzes $P1$

Testdatensatz $P1g$

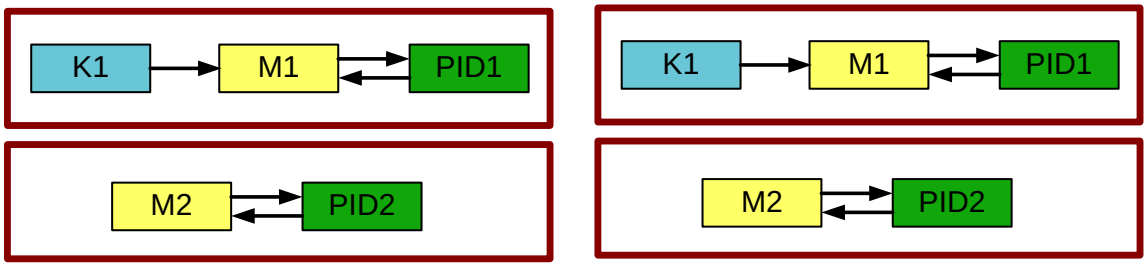

Abbildung B.21: Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 2.

Fitnessberechnung	Wert
Anzahl Teilprobleme	2
F_{RMSE}	$3,07\%$

Tabelle B.21: Fitness des Datensatzes $P1g$

Testdatensatz *P4gS*

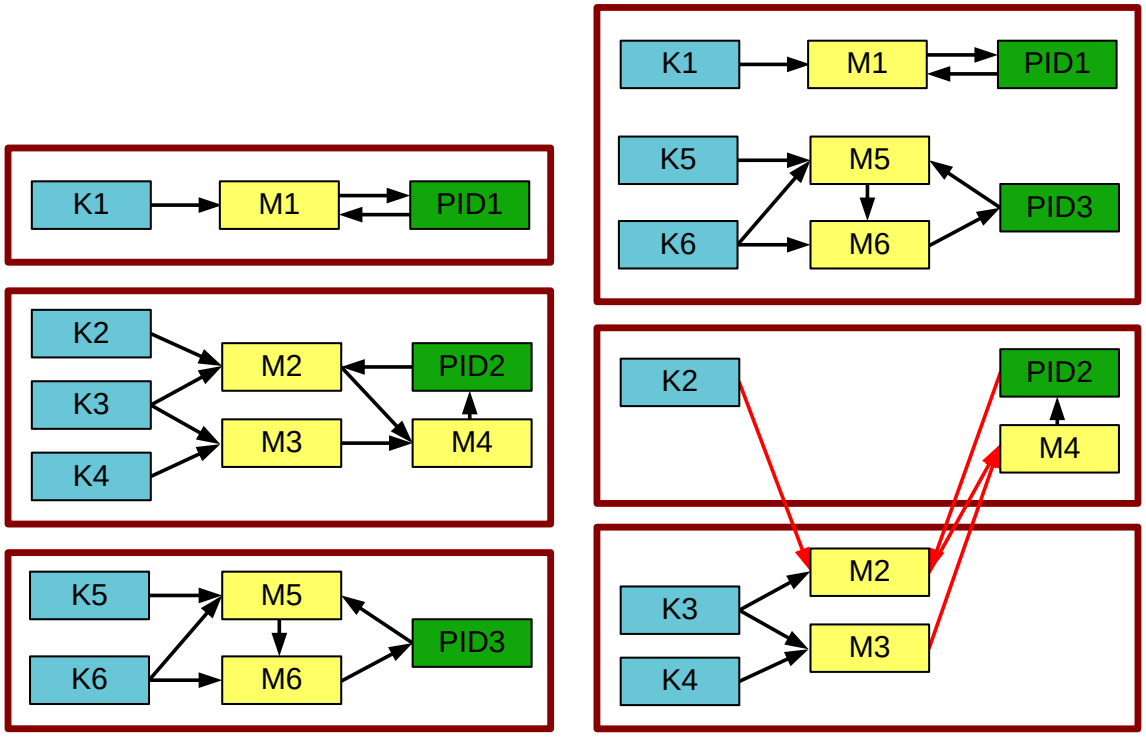

Abbildung B.22: Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 2.

Fitnessberechnung	Wert
Anzahl Teilprobleme	3
F_{RMSE}	$4,62\%$

Tabelle B.22: Fitness des Datensatzes *P4gS*

Anhang C

Ergebnisse der Time Delay Estimation

Dargestellt sind die Ergebnisse der Ermittlung der Totzeiten und Tiefpässe durch die Time Delay Estimation. In den Zeilen sind die Eingangsgrößen, in den Spalten die Ausgangsgrößen aufgelistet. So gibt beispielsweise ein Eintrag in der Zeile für K3 und der Spalte für M2 die vermutete Totzeit für eine mögliche Abhängigkeit von K3 zu M2 an.

C.1 Testdatensatz P1

C.1.1 Totzeiten für Datensatz P1

Problemgrößen	M1	M2
K1	1	2
K2	1	3
K3	8	1
M1	1	7
M2	1	1

Tabelle C.1: Ergebnisse der Time Delay Evaluation für Datensatz $P1$

C.1.2 Tiefpässe für Datensatz P1

Problemgrößen	M1	M2
K1	1	7
K2	1	10
K3	9	1
M1	1	1
M2	3	1

Tabelle C.2: Ergebnisse für die Tiefpässe für Datensatz $P1$

C.2 Testdatensatz P1g

C.2.1 Totzeiten für Datensatz P1g

Problemgrößen	M1	M2	PID1	PID2
K1	7	3	2	2
M1	1	9	1	8
M2	8	8	8	10
PID1	1	4	1	2
PID2	9	7	6	10

Tabelle C.3: Ergebnisse der Time Delay Evaluation für Datensatz $P1g$

C.2.2 Tiefpässe für Datensatz P1g

Problemgrößen	M1	M2	PID1	PID2
K1	2	10	1	10
M1	1	1	7	1
M2	2	9	7	3
PID1	1	10	1	10
PID2	2	10	10	10

Tabelle C.4: Ergebnisse für die Tiefpässe für Datensatz $P1g$

C.3 Testdatensatz P4gS

C.3.1 Totzeiten für Datensatz P4gS

Problemgrößen	M1	M2	M3	M4	M5	M6	PID1	PID2	PID3
K1	2	3	10	10	3	7	10	8	8
K2	1	4	3	7	2	8	1	10	5
K3	5	1	2	1	2	4	3	9	5
K4	6	10	1	6	6	3	10	5	1
K5	10	10	10	2	10	9	1	7	3
K6	5	10	8	1	2	1	10	2	10
M1	1	7	4	5	6	9	10	2	3
M2	6	1	3	2	2	3	1	10	1
M3	2	2	1	6	2	2	8	10	1
M4	5	1	3	1	1	7	4	9	9
M5	2	10	7	3	1	5	2	1	1
M6	1	6	10	4	1	1	10	6	10
PID1	10	1	10	1	3	1	1	1	9
PID2	7	8	1	10	9	10	8	1	7
PID3	4	3	7	7	10	10	8	2	1

Tabelle C.5: Ergebnisse der Time Delay Evaluation für Datensatz *P4gS*

C.3.2 Tiefpässe für Datensatz P4gS

Problemgrößen	M1	M2	M3	M4	M5	M6	PID1	PID2	PID3
K1	3	9	9	8	10	9	10	10	10
K2	1	3	2	3	1	6	8	10	6
K3	10	9	5	10	10	10	10	10	10
K4	9	9	10	9	9	9	10	9	9
K5	10	7	10	10	7	10	10	10	9
K6	9	10	10	10	5	10	10	10	10
M1	1	10	8	10	10	10	10	10	9
M2	6	1	10	3	6	10	10	10	6
M3	8	10	4	10	10	10	10	10	10
M4	8	1	3	1	5	10	9	10	7
M5	10	10	10	8	1	2	10	8	3
M6	8	10	8	9	2	1	10	10	10
PID1	10	7	1	2	1	2	1	3	8
PID2	8	9	1	9	10	9	10	1	10
PID3	9	9	10	10	8	10	9	10	1

Tabelle C.6: Ergebnisse für die Tiefpässe für Datensatz $P4gS$

Anhang D

Liste verwendeter Formelzeichen

a	Problemgröße $a \in A$
A	Menge der Abhängigkeiten eines Problems, bestehend aus Tupeln zweier Problemgrößen (a_1, a_2)
α	Parameter für die Bayes'sche Regularisierung, Vorwissen über den Grad der Glattheit der zu approximierenden Funktion
$B(x)$	Bin eines Histogramms, in welchem Datenpunkt x liegt
β	Parameter für die Bayes'sche Regularisierung, Vorwissen über die Höhe des Rauschens der zu approximierenden Funktion
C	Menge der korrekten Abhängigkeiten, welche zwischen Problemgrößen der selben Partition bestehen
D	Trainingsdatensatz eines neuronalen Netzes
e	Pufferwert für die Anzahl der benötigten Gewichte beim Optimal Brain Surgeon
E	Fehlerfunktion bei den Lernverfahren für neuronale Netze
ϵ	$= y - t$, einfache Fehlerfunktion für neuronale Netze
F	Fitness eines Individuums bei der Evolution
F_{PG}	Fitness, berechnet nach der Zugehörigkeit der Problemgrößen
F_{Abh}	Fitness, berechnet über die korrekten Abhängigkeiten
g^j	Gewicht für Experte j bei der Mixture of Experts Architektur
H	$= \partial^2 E \,/\, (\partial w_i \, \partial w_j)$, Hessematrix für das Training neuronaler Netze

$H(X)$	Entropie von X
I	Einheitsmatrix
$I(X;Y)$	Mutual Information zwischen X und Y
J	$= \partial\epsilon_i \,/\, \partial w_j$, Jacobi-Matrix für das Training neuronaler Netze
λ	Anzahl der Nachkommen bei der Evolution
λ	Lernrate für das Levenberg-Marquardt Lernverfahren
M	Menge der Problemgrößen eines Problems
M	Netzwerkmodell eines neuronalen Netzes, bestehend aus den Neuronen und ihren Verbindungen
μ	Größe der Population bei der Evolution
n	Anzahl Generationen bei der Evolution
n_{max}	maximale Anzahl Durchläufe beim Training neuronaler Netze
N	Anzahl Trainingspattern beim Training neuronaler Netze
N	Anzahl der Datenpunkte zur Abschätzung einer Wahrscheinlichkeitsdichtefunktion
N_{Hidden}	Anzahl Hiddenneuronen eines neuronalen Netzes
N_p	Anzahl der Partitionen einer Partitionierung
N_w	Anzahl der Gewichte eines neuronalen Netzes
N_y	Anzahl der Outputneuronen eines neuronalen Netzes
$N(B)$	Anzahl der Datenpunkte innerhalb des Bins B eines Histogramms
$p_{crossover}$	Wahrscheinlichkeit für ein Auftreten von Crossover bei der Evolution
$p_{crossover_swap}$	Wahrscheinlichkeit für einen Austausch der Gene beim Auftreten von Crossover bei der Evolution
p_{mut}	Mutationswahrscheinlichkeit bei der Evolution
$p(x)$	Wahrscheinlichkeitsdichtefunktion von X
$\hat{p}(x)$	Abschätzung der Wahrscheinlichkeitsdichtefunktion von X
$p_{Sel}(i)$	Wahrscheinlichkeit für die Selektion von Individuum i bei der Evolution

P_{Ind_i}	i. Partition der ermittelten Partitionierung, bestehend aus der Menge der enthaltenen Problemgrößen, verwendet zur Berechnung von F_{PG}
P_{true_i}	i. Partition der wahren Partitionierung, bestehend aus der Menge der enthaltenen Problemgrößen, verwendet zur Berechnung von F_{PG}
$P(x)$	Wahrscheinlichkeit für das Bin eines Histogramms, in welchem Datenpunkt x liegt
r_1, r_2	Parameter für lokale Varianzanpassung nach Nissen
$RMSE$	Root Mean Squared Error, Fehlermaß zur Bewertung neuronaler Netze
σ	Standardabweichung der Datenpunkte bei der Abschätzung der Mutual Information
σ_i	Standardabweichung für Gen i bei der Addition eines gaußverteilten Zufallsvektors bei der Mutation
σ_w	Standardabweichung der Gewichte eines neuronalen Netzes
σ_D	Standardabweichung des Rauschens auf den Daten
$\sigma(i)$	Permutationsfunktion $\sigma : \{1, .., N_P\} \rightarrow \{1, ..., N_P\}$ zum Permutieren der Problemgrößen
SSE	Summe der quadrierten Fehler, Fehlermaß zur Bewertung neuronaler Netze
SSW	Summe der quadrierten Gewichte, Gütemaß zur Bewertung der Glattheit neuronaler Netze
t	Vektor mit den Trainingspattern für das Training neuronaler Netze
t_i	i. Trainingspattern eines neuronalen Netzes
t_{ij}	i. Trainingspattern für das j. Outputneuron eines neuronalen Netzes
T_i	Totzeit für Problemgröße i
$\hat{T}_i$	abgeschätzte Totzeit für Problemgröße i
w	Vektor mit den Gewichten eines neuronalen Netzes
W	Binbreite eines Histogramms

W_X	Binbreite eines Histogramms für die X-Achse
x_i	Ereignis oder Wert einer Zufallsvariablen X
x_i	i. Inputvektor eines neuronalen Netzes
X	Zufallsvariable
y	Vektor mit den Netzausgaben eines neuronalen Netzes
$y(x_i)$	Ausgabe eines neuronalen Netzes für Inputvektor x_i
$y_j(x_i)$	Ausgabe des j. Outputneurons eines neuronalen Netzes für den Inputvektor x_i
y_i	Ereignis oder Wert einer Zufallsvariablen Y
Y	Zufallsvariable

Algorithmenverzeichnis

3.1 Levenberg-Marquardt Lernverfahren 35
3.2 Mutual Information Berechnung mit Hilfe von Histogrammen 41
4.1 Levenberg-Marquardt Lernverfahren mit Bayes'scher Regularisierung . 68
4.2 Optimal Brain Surgeon Algorithmus 72
4.3 Problemzerlegung durch Input Pruning 79
4.4 Problemzerlegung durch Partitionierung Variante 1 84
4.5 Problemzerlegung durch Partitionierung Variante 2 91
4.6 Time Delay Estimation durch Mutual Information 97
4.7 Ermittlung der Anzahl Teilprobleme bei der Problemzerlegung durch Partitionierung 103
4.8 Nicht exklusive Problemzerlegung durch Partitionierung 107

Abbildungsverzeichnis

2.1 Darstellung des grundlegenden Ablaufs in einem Kohlekraftwerk 5
2.2 Darstellung der Abhängigkeiten im Beispielszenario Kohlekraftwerk . . 6
2.3 Approximation eines Regelsystems durch ein neuronales Netzwerk . . . 7
2.4 Übersicht über verschiedene Arten der Problemdekomposition 8
2.5 Darstellung der zeitlichen Problemdekomposition am Beispielproblem . 9
2.6 Mixture of Experts - Architektur für das Beispielproblem 10
2.7 Darstellung der nichthierarchisch funktionalen Problemdekomposition am Beispielproblem 11
2.8 Architektur der nichthierarchisch funktionalen Problemdekomposition für das Beispielproblem 12
2.9 Veranschaulichung der hierarchisch funktionalen Problemdekomposition anhand des funktionalen Verlaufs der Problemgrößen 14
2.10 Architektur der hierarchisch funktionalen Problemdekomposition 15
2.11 Veranschaulichung der Verringerung der Punktdichte in höherdimensionalen Räumen 17
2.12 Veranschaulichung des Nutzens einer Problemzerlegung 18

3.1 Darstellung der Ausbreitung vorteilhafter Merkmale 23
3.2 Veranschaulichung der Rekombination 25
3.3 Veranschaulichung der Mutation 25
3.4 Diagramm zum Ablauf evolutionärer Algorithmen 27
3.5 Darstellung der Wahrscheinlichkeitsverteilung bei der linearen Rangauswahl 29

3.6 Darstellung des Uniform-Crossovers 31
3.7 Darstellung der Zusammensetzung der Fehlerfunktion des Levenberg-Marquardt Algorithmus 34
3.8 Darstellung des Zusammenhangs zwischen Entropie und Mutual Information 37
3.9 Approximation einer Beispielverteilung durch Histogramme 39

4.1 Veranschaulichung der Fitnessberechnung über die Zugehörigkeit der Problemgrößen 47
4.2 Veranschaulichung der Fitnessberechnung über die Abhängigkeiten . . . 49
4.3 Veranschaulichung der Fitnessberechnung über die Fehlerrate 52
4.4 Veranschaulichung der Reduktion der Anzahl der Gewichte bei einer Problemdekomposition 55
4.5 Veranschaulichung der Glattheit der Ausgabefunktion bei unterschiedlich großen Gewichten 56
4.6 Illustration des Datenmodells der Bayes'schen Regularisierung 58
4.7 Illustration des Gewichtsmodells der Bayes'schen Regularisierung . . . 60
4.8 Illustration der Wahl von Parameter α bei der Bayes'schen Regularisierung 63
4.9 Illustration des Nutzens kleiner Gewichte in einem neuronalen Netz . . 69
4.10 Veranschaulichung der Aufgabe der Problemzerlegenden Verfahren am Beispielproblem 74
4.11 Illustration der Problemzerlegung durch Input Pruning 75
4.12 Illustration der Problemzerlegung durch Input Pruning für das Beispielproblem 77
4.13 Illustration der Nachteile der Problemzerlegung durch Input Pruning . 80
4.14 Illustration der Problemzerlegung durch Partitionierung für das Beispielproblem 82
4.15 Training der neuronalen Netze bei der Problemzerlegung durch Partitionierung 83
4.16 Approximation einer Problemgröße durch sich selbst 86

4.17 Illustration der Problemzerlegung durch Partitionierung Variante 2 für das Beispielproblem 88
4.18 Training der neuronalen Netze bei der Problemzerlegung durch Partitionierung Variante 2 89
4.19 Illustration der Notwendigkeit der Time Delay Estimation 94
4.20 Veranschaulichung des Herausrechnens der Zeitverzögerung für das neuronale Netz 96
4.21 Darstellung der Mutation der Partitionen zur Problemzerlegung 99
4.22 Darstellung der Mutation in eine neue Partition am Beispielproblem . . 100
4.23 Illustration der automatischen Berechnung der Anzahl der Teilprobleme 102
4.24 Beispiel für ein nicht exklusiv trennbares Problem 105
4.25 Darstellung eines minimalen Schnittes am Beispielproblem 106

5.1 Abhängigkeiten Testdatensatz P1 116
5.2 Abhängigkeiten Testdatensatz P1g 118
5.3 Abhängigkeiten Testdatensatz P2 119
5.4 Abhängigkeiten Testdatensatz P2g 120
5.5 Abhängigkeiten Testdatensatz P3 121
5.6 Abhängigkeiten Testdatensatz P3g 122
5.7 Abhängigkeiten Testdatensatz P4gS 123
5.8 Abhängigkeiten Testdatensatz P9g 125
5.9 Problemzerlegung des Testdatensatzes P1 durch Input Pruning in der Standardvariante 132
5.10 Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 1 134
5.11 Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 2 135
5.12 Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 1 140
5.13 Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 1 141

5.14 Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 1 143
5.15 Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 2 144
5.16 Problemzerlegung des Testdatensatzes P4gS durch das Mutual Information Verfahren 146
5.17 Problemzerlegung des Testdatensatzes P4gS durch das Granger Kausalität-Verfahren 148

B.1 Problemzerlegung des Testdatensatzes P1 durch Input Pruning 166
B.2 Problemzerlegung des Testdatensatzes P1g durch Input Pruning 167
B.3 Problemzerlegung des Testdatensatzes P1 durch Partitionierung Variante 1 168
B.4 Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 1 169
B.5 Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 1 170
B.6 Problemzerlegung des Testdatensatzes P1 durch Partitionierung Variante 2 171
B.7 Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 2 172
B.8 Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 2 173
B.9 Problemzerlegung des Testdatensatzes P1 durch Input Pruning 174
B.10 Problemzerlegung des Testdatensatzes P1g durch Input Pruning 175
B.11 Problemzerlegung des Testdatensatzes P1 durch Partitionierung Variante 1 176
B.12 Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 1 177
B.13 Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 1 178

B.14 Problemzerlegung des Testdatensatzes P1 durch Partitionierung Variante 2 179
B.15 Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 2 180
B.16 Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 2 181
B.17 Problemzerlegung des Testdatensatzes P1 durch Partitionierung Variante 1 183
B.18 Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 1 184
B.19 Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 1 185
B.20 Problemzerlegung des Testdatensatzes P1 durch Partitionierung Variante 2 186
B.21 Problemzerlegung des Testdatensatzes P1g durch Partitionierung Variante 2 187
B.22 Problemzerlegung des Testdatensatzes P4gS durch Partitionierung Variante 2 188

Literaturverzeichnis

[Azam, 2000] Azam, F. (2000). *Biologically Inspired Modular Neural Networks*. Dissertation, Virginia Polytechnic Institute and State University.

[Bishop, 1995] Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

[Cun et al., 1990] Cun, L. Y., J. S. Denker und S. A. Solla (1990). *Optimal Brain Damage*. In: *Advances in Neural Information Processing Systems*, S. 598–605. Morgan Kaufmann.

[Darwin, 1859] Darwin, C. R. (1859). *On the origin of species*. John Murray, London.

[Debes, 2010] Debes, K. (2010). *Softcomputing 2: Evolutionäre Algorithmen*. Vorlesungsskript, TU Ilmenau.

[Duro et al., 2010] Duro, R. J., F. Bellas, P. Caamaño und G. Varela (2010). *Automatic model decomposition and reuse in an evolutionary cognitive mechanism*. Evolving Systems, 1:129–141.

[Foresee und Hagan, 1997] Foresee, F. D. und M. T. Hagan (1997). *Gauss-Newton Approximation to Bayesian Learning*. International Conference on Neural Networks, 3:1930–1935.

[Gross und Schaffernicht, 2009] Gross, H. M. und E. Schaffernicht (2009). *Angewandte Neuroinformatik*. Vorlesungsskript, TU Ilmenau.

[Hassibi et al., 1993] Hassibi, B., D. Stork und G. Wolff (1993). *Optimal Brain Surgeon and General Network Pruning*. IEEE International Conference on Neural Networks, 1:293 – 299.

[Jacob, 1997] Jacob, C. (1997). *Principia Evolvica*. Dpunkt.Verlag, Heidelberg.

[Jacobs et al., 1991] Jacobs, A. J., M. I. Jordan und A. G. Barto (1991). *Task Decomposition Through Competition in a Modular Connectionist Architecture: The What and Where Vision Tasks*. Cognitive Science, 15:219–250.

[Kaltenhäuser, 2010] Kaltenhäuser, R. (2010). *Schätzung von Mutual Information aus Daten*. Bachelorarbeit, TU Ilmenau.

[Khare et al., 2005] Khare, V., X. Yao, B. Sendhoff, Y. Jin und H. Wersing (2005). *Co-evolutionary Modular Neural Networks for Automatic Problem Decomposition*. In: *Congress on Evolutionary Computation (CEC)*, S. 2691–2698. IEEE Press.

[Ku und Mak, 1997] Ku, K. W. C. und M. W. Mak (1997). *Exploring the effects of Lamarckian and Baldwinian learning in evolving recurrent neural networks*. In: *IEEE International Conference on Evolutionary Computation*, S. 617 – 621.

[MacKay, 1991] MacKay, D. J. C. (1991). *Bayesian Interpolation*. Neural Computation, 4:415–447.

[Mayr, 2005] Mayr, E. (2005). *Das ist Evolution*. Goldmann Verlag, München.

[Nguyen et al., 2006] Nguyen, M. H., H. A. Abbass und R. I. Mckay (2006). *A novel mixture of experts model based on cooperative coevolution*. Neurocomputing, 70:155–163.

[Nissen, 1997] Nissen, V. (1997). *Einführung in Evolutionäre Algorithmen*. Vieweg+Teubner, Braunschweig.

[Potter und De Jong, 2000] Potter, M. A. und K. A. De Jong (2000). *Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents*. Evolutionary Computation, 8(1):1–29.

[Scott, 1992] Scott, D. W. (1992). *Multivariate density estimation: theory, practice and visualization*. John Wiley & Sons, New York.

[Shannon, 1948] Shannon, C. E. (1948). *A mathematical theory of communication*. The Bell System Technical Journal, 27:379–423.

[Torke, 2012] Torke, P. R. (2012). *Automatische Problemdekomposition zur Zerlegung homöostatischer Regelkreise*. Diplomarbeit, TU Ilmenau.

[Wakunda, 1995] Wakunda, J. (1995). *Verteilte Evolutionsstrategien auf dem Supercomputer Intel Paragon*. Diplomarbeit, Universität Stuttgart.

[Wen und Wan, 2009] Wen, F. und Q. Wan (2009). *Time Delay Estimation Based on Mutual Information Estimation*. CISP '09. 2nd International Congress on Image and Signal Processing, S. 1–5.