Universität Siegen
Department Maschinenbau
Institut für Produktionstechnik
Logistik für Produktionsunternehmen

Masterarbeit

Einsatz von Machine-Learning-Algorithmen zur Planung und Optimierung von Kommissioniersystemen

Name: B.Sc. Patrick Dickel
Matrikel-Nr.: 900317

Studiengang: Wirtschaftsingenieurwesen (M.Sc.)

1. Betreuer: Prof. Dr.-Ing. Dipl.-Oec. Ulrich Stache
2. Betreuer: M.Sc. Tobias Mauksch

Siegen, der 18. Oktober 2018

Kurzfassung

Die automatische Planung von Kommissioniersystemen ist eine sehr komplexe Aufgabe, da es eine Vielzahl möglicher Gestaltungsmerkmale und daraus resultierender Konfigurationen existieren. Zudem ist nicht bekannt, wie einzelne Merkmale auf Zielgrößen wirken bzw. wie bei gegebenen Randbedingungen ein optimales Kommissionersystem gestaltet werden sollte.

Es wurde hier das konventionelle Kommissionieren in der gassenungebundenen und gassengebundenen Betrachtung untersucht, da es sich hierbei um die am weitesten verbreitetste Kommissioniersystemvariante handelt und einige wesentliche konzeptionelle Unterschiede in den beiden Betrachtungsweisen vorherrschen. Zur Simulation der Systeme wurde in der Vergangenheit ein Excel-Tool entwickelt, welches jedoch nur sehr langsam rechnet. Es sollte daher durch sehr schnelle Metamodell ersetzt werden.

Nach einer umfassenden Versuchsplanung wurden im ersten Teil der Arbeit Metamodelle trainiert und anschließend mit Hilfe von Interpretationsmethoden untersucht. Dabei hat sich gezeigt, dass die Investitionskosten bei beiden Betrachtungen nahezu identisch sind. Weiterhin wurde deutlich, dass die gassenungebundene wesentlich mehr Kommissionierer benötigt, als die gassengebundene Betrachtung. Durch die Verwendung von Metamodellen die Rechenzeiten mindestens um den Faktor 1.756 eingespart werden, allerdings zu dem Preis der Modellungenauigkeiten.

Im zweiten Teil wurde für die gassenungebundene Betrachtung ein Genetische Algorithmus entwickelt, welcher optimale Konfigurationen für Kommissioniersystemkomponenten bei einstellbaren Leistungsanforderungen findet. Dazu wurden fünf unterschiedliche Kommissioniersystemgrößen getestet. Es hat sich gezeigt, dass der Algorithmus auch bei Konfigurationen eine optimale Lösung findet, bei denen zunächst keine zulässige Lösung vorhanden ist. Des Weiteren war festzustellen, dass der Algorithmus bei zwei Kommissioniersystemgrößen dazu neigt in das zweitbeste Optimum zu konvergieren. Durch die Verwendung von Restarts konnte den entgegengewirkt werden.

Abstract

The automatic planning of picking systems is a very complex task, since there are a variety of possible design features and resulting configurations. In addition, it is unknown how individual features act on target variables or how an optimal picking system should be designed under given constraints.

Conventional order picking in the “lane-unbound” and “lane-bound” approach have been studied since this is the most common picking system variation and there are some major conceptual differences between the two approaches. To simulate the systems, an Excel-tool was developed in the past, which, unfortunately, only calculates very slowly. It should therefore be replaced by very fast meta-model.

After a comprehensive experimental design, in the first part of this thesis metamodels were trained and then examined with the help of interpretation methods. It has been shown that the investment costs are almost identical for both approaches. Furthermore, it became clear that the lane-unbound approach needed way more order pickers than the lane-bound approach. By using meta-models, the computing times are saved by a factor of at least 1756, at the price of model inaccuracies.

In the second part, a genetic algorithm was developed for the lane-unbound approach, which can find optimal configurations for picking system components with adjustable performance requirements. For this purpose, five different picking system sizes were tested. It has been shown that the algorithm also finds an optimal solution for configurations in which no permissible solution is initially available. It has also been noted that the algorithm tends to converge to the second-best optimum for two picking system sizes. By using Restarts this could be counteracted.

Inhaltsverzeichnis

Abbildungsverzeichnis VIII

Tabellenverzeichnis XII

Quellcodeverzeichnis XIII

Abkürzungsverzeichnis XIV

1 Einleitung 1

1.1 Ausgangssituation und Problemstellung 1

1.2 Zielsetzung und Vorgehensweise 1

2 Theoretische Grundlagen 4

2.1 Kommissioniersysteme 4

2.1.1 Grundlagen der Kommissionierung 5

2.1.2 Technische Komponenten 12

2.1.3 Betriebsstrategien 14

2.1.4 Konventionelles Kommissionieren 19

2.2 Statistische Versuchsplanung 20

2.2.1 Faktorielle Versuchspläne und Effekte 22

2.2.2 Gleichverteilte Versuchspläne 26

2.3 Metamodelle 30

2.3.1 Machine-Learning-Grundlagen und Notation 30

2.3.2 Multi-layer Perceptron 35

2.3.3 Modellauswahl 51

2.4 Auswertungsmethoden 56

2.4.1 Relative Importance 57

2.4.2 Partial Dependence Plot 57

2.4.3 Individual Conditional Expectation Plot 59

2.5 Genetische Algorithmen ... 62
2.5.1 Kodierung ... 65
2.5.2 Initialisierung ... 67
2.5.3 Selection ... 67
2.5.4 Crossover ... 69
2.5.5 Mutation ... 70
2.5.6 Fitness-Funktion ... 72
2.5.7 Termination ... 75
2.5.8 Multimodale Probleme und Restart-Strategien ... 76
2.5.9 Parameter-Optimierung ... 77
3 Analyse von Wirkzusammenhängen ... 79
3.1 Vorgehensweise ... 79
3.2 Gassenungebundenes konventionelles Kommissionieren ... 83
3.2.1 Versuchsplanung ... 84
3.2.2 Metamodelltraining und -auswahl ... 104
3.2.3 Anwendung von Interpretationsmethoden ... 120
3.3 Gassengebundenes konventionelles Kommissionieren ... 134
3.3.1 Versuchsplanung ... 134
3.3.2 Metamodelltraining und -auswahl ... 140
3.3.3 Anwendung von Interpretationsmethoden ... 140
3.4 Vergleich der Kommissioniersysteme ... 151
3.5 Rechenzeiten und Rundungen ... 155
4 Optimierung von Kommissioniersystemen ... 159
4.1 Vorgehensweise ... 159
4.2 Zieldefinition ... 160

4.3 Komponenten des Genetischen Algorithmus 161
4.3.1 Kodierung 162
4.3.2 Initialisierung 164
4.3.3 Fitness-Funktion 167
4.3.4 Selection 173
4.3.5 Crossover 173
4.3.6 Mutation 174
4.3.7 Termination 176
4.4 Zusammengesetzter Genetischer Algorithmus 177
4.5 Testkonfigurationen 179
5 Fazit und Ausblick 187
6 Literaturverzeichnis 190
7 Anhang 202
A-1 Rechnerkonfiguration 202
A-2 Verwendete Programme 202
A-3 Verwendete Python-Pakete 203
A-4 Kommissioniersystem Konfigurationen 204
A-5 Technische Komponenten von Kommissioniersystemen 205
A-6 Modellauswahl Investitionskosten (gassengebunden) 206
A-7 Modellauswahl Betriebskosten (gassengebunden) 207
A-8 Modellauswahl Anzahl Kommissionierer (gassengebunden) 208
A-9 Modellauswahl TCO (gassengebunden) 209
A-10 PDPs Investitionskosten (gassenungebunden) 210
A-11 PDPs Betriebskosten (gassenungebunden) 212
A-12 PDP Anzahl Kommissionierer (gassenungebunden) 214

A-13 PDPs TCO (gassenungebunden) 216
A-14 PDPs Investitionskosten (gassengebunden) 218
A-15 PDPs Betriebskosten (gassengebunden) 220
A-16 PDPs Anzahl Kommissionierer (gassengebunden) 222
A-17 PDPs TCO (gassengebunden) 224
A-18 Spannweiten der Komponenten-Phänotypen 226
A-19 Optimale Konfiguration sehr kleines Kommissioniersystem 227
A-20 Optimale Konfiguration kleines Kommissioniersystem 228
A-21 Optimale Konfiguration mittleres Kommissioniersystem 229
A-22 Optimale Konfiguration großes Kommissioniersystem 230
A-23 Optimale Konfiguration sehr großes Kommissioniersystem 231
8 Selbstständigkeitserklärung 232

Abbildungsverzeichnis

Abbildung 1: Ein- und zweidimensionale Fortbewegung (Hompel et al. 2011, S. 24) 8
Abbildung 2: Person-zur-Ware Beispiel (Gudehus 2010, S. 684) 11
Abbildung 3: Ware-zur-Person Beispiel (Hompel et al. 2011, S. 42) 11
Abbildung 4: ABC-Zonen mit Streifensystem nach Hompel et al. 2010, S. 93 15
Abbildung 5: Gassenungebundene Betrachtung nach Hompel et al. 2011, S. 141–144 16
Abbildung 6: Kopf- und Zentralganglayout nach Hompel et al. 2011, S. 155 17
Abbildung 7: Schleifenstrategie mit/ohne Überspringen nach Hompel et al. 2011, S. 96 . 17
Abbildung 8: Stichgangstrategie mit/ohne Gangwiederholung nach Hompel et al. 2011, S. 97 18
Abbildung 9: Mittelpunkt-Heuristik nach Hompel et al. 2011, S. 98 18
Abbildung 10: Konventionelles Kommissionieren (Gudehus 2010, S. 670) 20
Abbildung 11: Schritte der Statistischen Versuchsplanung nach Kleppmann 2013 22
Abbildung 12: Vollständiger faktorieller 2^2-Versuchsplan 23
Abbildung 13: Effekt- und Wechselwirkungsdiagramme 24
Abbildung 14: Zweidimensionale Sobol'-Sequenz mit N=20 und N=100 27
Abbildung 15: Collapsing und Non-collapsing mit N=9 29
Abbildung 16: Zweidimensionales LHD (Maximin) mit N=20 und N=100 30
Abbildung 17: Bias-Variance Tradeoff 34
Abbildung 18: Over- und Underfitting 35
Abbildung 19: Schematische Darstellung eines Multi-layer Perceptron 37
Abbildung 20: Aufbau eines Neurons 38
Abbildung 21: Identische Abbildung 39
Abbildung 22: Sigmoid bzw. Logistische Funktion 39
Abbildung 23: Tangens Hyperbolicus 40
Abbildung 24: ReLU 40
Abbildung 25: Gradient Descent 44
Abbildung 26: Visualisierung der GD-Varianten 49
Abbildung 27: Early Stopping 50
Abbildung 28: Risidual-Plot 53

Abbildung 29: Beispielhafter ICE für ein x_2 (Goldstein et al. 2014) 60
Abbildung 30: Schritte eines Genetischen Algorithmus 64
Abbildung 31: Crossover in Anlehnung an Kramer 2017, S. 13 69
Abbildung 32: Uniform Crossover 70
Abbildung 33: Multimodales Optimierungsproblem 76
Abbildung 34: Vorgehensweise bei der Analyse von Wirkzusammenhängen 80
Abbildung 35: Ausschnitt aus TB "Kommissioniersystem" 82
Abbildung 36: Ausschnitt aus TB "Ergebnis-Zusammenfassung" 83
Abbildung 37: Voreinstellungen für LM „Fachbodenregal“ im TB „Technik“ 87
Abbildung 38: TB "Systemkonfiguration" 88
Abbildung 39: Voreinstellungen für FM „Niederhubwagen“ im TB „Technik“ 89
Abbildung 40: Voreinstellungen für LHM „Großer KLT“ im TB „Technik“ 90
Abbildung 41: Voreinstellungen für ITK „Pickliste“ im TB „Technik“ 90
Abbildung 42: Abhängigkeiten bei Basis 93
Abbildung 43: Abhängigkeiten bei Wegstrategie 95
Abbildung 44: Effektdiagramme für Layout, Verteilung und Ansteuerung 96
Abbildung 45: Effektdiagramm für Basis 97
Abbildung 46: Effektdiagramm für Wegstrategie 98
Abbildung 47: Wechselwirkung zwischen Wegstrategie und Geschwindigkeitsveränderung 99
Abbildung 48: Wechselwirkungen gassenungebundenes konventionelles Kommissionieren 100
Abbildung 49: Vorgehensweise der Hyperparameter-Optimierung 105
Abbildung 50: Verteilung der Fehler für y1_invest gug. 114
Abbildung 51: Ergebnisse der Testdaten für y1_invest gug. 115
Abbildung 52: Verteilung der Fehler für y2_betrieb gug. 116
Abbildung 53: Ergebnisse der Testdaten für y2_betrieb gug. 117
Abbildung 54: Verteilung der Fehler für y3_anz_kommi gug. 117
Abbildung 55: Ergebnisse der Testdaten für y3_anz_kommi gug. 118
Abbildung 56: Verteilung der Fehler für y4_tco gug. 119

Abbildung 57: Ergebnisse der Testdaten für y4_tco gug. 120
Abbildung 58: Relative Importance für y1_invest gug. 123
Abbildung 59: ICE-Plots für alg_anz_gasse und lm_anz_reihe gug. 123
Abbildung 60: 3D-PDP alg_anz_gassen und lm_anz_reihe für y1_invest gug. 124
Abbildung 61: ICE-Plots für lm_fach_kapazitaet und itk_kosten_fach gug. 125
Abbildung 62: 3D-PDP lm_fach_kapazitaet und itk_kosten_fach für y1_invest gug. 125
Abbildung 63: Relative Importance für y2_betrieb gug. 126
Abbildung 64: ICE-Plots für fm_max_a und fm_max_v gug. 127
Abbildung 65: c-ICE für fm_max_a und fm_max_v gug. 127
Abbildung 66: Relative Importance für y3_anz_kommi gug. 128
Abbildung 67: ICE-Plots für alg_auftrag_tag und itk_bearb_zeit gug. 129
Abbildung 68: c-ICE für fm_max_a und fm_max_v 129
Abbildung 69: ICE für alg_anz_gassen und 3D-PDP Gassenlänge für y3_anz_kommi gug. 130
Abbildung 70: Relative Importance für y4_tco gug. 131
Abbildung 71: 3D-PDP fm_max_v und itk_bearb_zeit für y4_tco gug. 132
Abbildung 72: c-ICE für FM gug. 132
Abbildung 73: c-ICE für ITK gug. 133
Abbildung 74: Effektdiagramme für Layout, Verteilung, Ansteuerung und Wegstrategie 136
Abbildung 75: Effektdiagramm für Basis 136
Abbildung 76: Wechselwirkungen gassengebundenes konventionelles Kommissionieren 138
Abbildung 77: Relative Importance für y1_invest gg. 141
Abbildung 78: c-ICE-Plot für alg_auftrag_tag und itk_bearb_zeit gg. 141
Abbildung 79: Relative Importance für y2_betrieb gg. 143
Abbildung 80: c-ICE für fm_max_a und fm_max_v gg. 144
Abbildung 81: c-ICE für fm_kosten und lhm_kosten gg. 144
Abbildung 82: 3D-PDPs lhm_kosten mit lm_anz_reihe bzw. lm_fach_kapazitaet für y2_betrieb gg. 145

Abbildung 83: Relative Importance für y3_anz_kommi gg. 146
Abbildung 84: 3D-PDP fm_max_v und itk_bearb_zeit für y3_anz_kommi gg. 146
Abbildung 85: 3D-PDP alg_auftrag_tag mit alg_max_pos_auftrag bzw. itk_bearb_zeit für y3_anz_kommi gg. 147
Abbildung 86: ICE für alg_anz_gassen und 3D-PDP Gassenlänge für y3_anz_kommi gg. 147
Abbildung 87: Relative Importance für y4_tco gg. 148
Abbildung 88: c-ICE für ITK gg. 149
Abbildung 89: c-ICE für FM gg. 150
Abbildung 90: Effektdiagramm der Zielgrößen 151
Abbildung 91: c-ICE für fm_kosten, itk_kosten_kommi in gug. und gg. 152
Abbildung 92: c-ICE für itk_bearb_zeit gug und gg. 153
Abbildung 93: Residual-Plot für die Betriebskosten mit Rundungen 158
Abbildung 94: Residual-Plot für die Anzahl der Kommissionierer mit Rundungen 158
Abbildung 95: Vorgehensweise bei der Optimierung durch einen GA 160
Abbildung 96: Ebenen der Kodierung 162
Abbildung 97: Fitness-Verlauf für mittleres Kommissioniersystem 181
Abbildung 98: Fitness-Verlauf für sehr großes Kommissioniersystem 183
Abbildung 99: Fitness-Verlauf für kleines Kommissioniersystem mit Restarts 185

Tabellenverzeichnis

Tabelle 1: Konventionelles Kommissionieren 19
Tabelle 2: Faktoren nach Typ 85
Tabelle 3: Spannweiten der quantitativen Faktoren 92
Tabelle 4: Durchschnittswerte der Zielgrößen des Vollfaktorplans (gassenungebunden) . 95
Tabelle 5: Top 5 des Vollfaktorplans (gassenungebunden) 101
Tabelle 6: Konfiguration für gassenungebundenes konventionelles Kommissionieren ... 102
Tabelle 7: Spannweiten der vier Zielgrößen gug. 113
Tabelle 8: Top 10 Ergebnisse der Validierungsdaten für y1_invest gug. 115
Tabelle 9: Top 10 Ergebnisse der Validierungsdaten für y2_betrieb gug. 116
Tabelle 10: Top 10 Ergebnisse der Validierungsdaten für y3_anz_kommi gug. 118
Tabelle 11: Top 10 Ergebnisse der Validierungsdaten für y4_tco gug. 119
Tabelle 12: Durchschnittswerte der Zielgrößen des Vollfaktorplans gg. 135
Tabelle 13: Top 5 des Vollfaktorplans (gassengebunden) 139
Tabelle 14: Konfiguration für gassengebundenes konventionelles Kommissionieren 139
Tabelle 15: Spannweiten der vier Zielgrößen gg. 140
Tabelle 16: Vergleich der Zielgrößen 151
Tabelle 17: Excel-Tool-Rechenzeiten bei N=30.000 Samples 156
Tabelle 18: MLP-Rechenzeiten bei N=30.000 Samples 157
Tabelle 19: Gruppierung von Phänotypen 163
Tabelle 20: Genotyp des Genetischen Algorithmus 164
Tabelle 21: Testergebnisse der Kommissioniersysteme 180
Tabelle 22: Hall of Fame Vorher/Nachher, mittleres Kommissioniersystem 181
Tabelle 23: Hall of Fame Vorher/Nachher, sehr großes Kommissioniersystem 182
Tabelle 24: Testergebnisse der Kommissioniersysteme mit Restarts 184
Tabelle 25: Hall of Fame Vorher/Nachher, kleines Kommissioniersystem mit Restarts .. 186

Quellcodeverzeichnis

Quellcode 1: Erzeugung des Testfeldes in MATLAB 103
Quellcode 2: Grundfunktion der Hyperparameter-Optimierung 106
Quellcode 3: Split in Trainings-, Test- und Validierungsdaten 107
Quellcode 4: Funktion zur Random Search für MLPs 108
Quellcode 5: Funktion zur Random Search für MLPs (Fortsetzung) 109
Quellcode 6: Funktion zum Training eines MLPs 111
Quellcode 7: Berechnung der Maße zur Regressionsgüte 112
Quellcode 8: Implementierung des Olden-Algorithmus 121
Quellcode 9: Benchmark-Funktion 156
Quellcode 10: Importierung von DEAP 161
Quellcode 11: Initialisierung der Anfangspopulation 165
Quellcode 12: Initialisierung der Anfangspopulation (Fortsetzung) 166
Quellcode 13: Statistik-Tools in DEAP 166
Quellcode 14: Bewertung eines Individuums 168
Quellcode 15: Leeres Phänotyp-Dictionary 169
Quellcode 16: Zusammensetzen des Phänotypen – FM- und LM-Anteil 169
Quellcode 17: Zusammensetzen des Phänotypen – Berechnung der Aufträge / Modulbreite 170
Quellcode 18: Berechnung der Stellplätze des Kommissioniersystems 170
Quellcode 19: Berechnung der Fitness (TCO) 171
Quellcode 20: Tournament-Selection in DEAP definieren 173
Quellcode 21: Uniform-Crossover in DEAP definieren 173
Quellcode 22: Hybride-Mutation 175
Quellcode 23: Termination-Kriterium 176
Quellcode 24: Hauptschleife des GA 178

Abkürzungsverzeichnis

ANN	Artificial Neural Network
BCGA	Binary-Coded Genetic Algorithm
BP	Backpropagation
BS	Batch Size
c-ICE	Centered Individual Conditional Expectation Plot
d-ICE	Derivative Individual Conditional Expectation Plot
DNS	Desoxyribonukleinsäure
ES	Early Stopping
FM	Fördermittel
GA	Genetischer Algorithmus
GD	Gradient Descent
gg.	gassengebunden
gug.	gassenungebunden
ICE	Individual Conditional Expectation Plot
ITK	Informationstechnische Kommissioniererführung
KDE	Kernel density estimation
KNN	Künstliches Neuronales Netzwerk
LHD	Latin Hypercube Design
LHM	Ladehilfsmittel
LM	Lagermittel
LR	Learning Rate
MAE	Mean Absolute Error
MLP	Multi-layer Perceptron
MSE	Mean Squared Error
NRMSE	Normalized Root Mean Squared Error
PDP	Partial Dependence Plot
PzW	Person-zur-Ware
R^2	Bestimmtheitsmaß
RCGA	Real-Coded Genetic Algorithm
ReLU	Rectified Linear Units
RI	Relative Importance
RMSE	Root Mean Squared Error
RS	Random State
SGD	Stochastic Gradient Descent
TB	Tabellenblatt
TCO	Total Cost of Ownership (über 10 Jahre)
TS	Tournament Selection
VDI	Verein Deutscher Ingenieure
WzP	Ware-zur-Person

1 Einleitung

1.1 Ausgangssituation und Problemstellung

Die automatische Planung von Kommissioniersystemen ist eine sehr komplexe Aufgabe: Zum einen ist nicht genau bekannt, welche bzw. wie viele Gestaltungsalternativen vorhanden und miteinander kombinierbar sind. Zum anderen ist die Wirkung eines Gestaltungsmerkmals auf Zielgrößen (bspw. Kommissionierleistung oder Kosten) oftmals nichtlinear und davon abhängig, welche Kombinationen von Gestaltungsmerkmalen gerade gewählt wurden.

Um die Wirkung von unterschiedlichen Kombinationen der Gestaltungsmerkmale auf bestimmte Zielgrößen berechnen zu können, wurde in der Vergangenheit ein Excel-Simulations-Tool entwickelt. Dies ist in der Lage, allein im Bereich des *Konventionellen Kommissionierens* nur auf Basis von kategorischen Variablen, bis zu 154.560 Varianten zu berechnen. Diese hohe Variantenvielfalt spiegelt einerseits den zur automatischen Planung notwendigen sehr hohen Detaillierungsgrad wider, andererseits ergibt sich daraus ein sehr hoher Rechenaufwand zur Lösung der großen Anzahl an Gleichungen. Auf Grund dieser hohen Komplexität und der dadurch bedingten hohen Rechenleistung ist das Excel-Tool nur sehr langsam, was beispielsweise eine simulationsbasierte Optimierung gar nicht bzw. nur sehr ineffizient möglich macht.

1.2 Zielsetzung und Vorgehensweise

Nachdem eine theoretische Grundlage geschaffen wurde, soll das erste Ziel dieser Arbeit die Erstellung von leistungsfähigen Metamodellen zur Berechnung ausgewählter Zielgrößen in Abhängigkeit gegebener Systemparameter für die bis dato am weitesten verbreitete Kommissioniersystem-Variante sein: *Konventionelles Kommissionieren*. Auf Grund einiger grundlegender Unterschiede in der Berechnung möglicher Zielgrößen, wird hier zwischen der *gassenungebundenen* (gug.) und *gassengebundenen* (gg.) *Betrachtung* unterschieden (Gudehus 2010, S. 669).

Dazu sollen zunächst die zur Berechnung der Zielgrößen notwendigen Variablen ausgewählt und anschließend (wenn nötig) durch deren Modifizierung für spätere

Arbeitsschritte handhabbar gemacht werden. Danach sollen Testfelder für die Durchführung von Simulationen im Excel-Tool erstellt und optimiert werden. Die Testfelder geben dabei an, welche Systemparameter als Input in das Excel-Tool gegeben werden. Softwareimplementierungen dafür finden sich in MATLAB und Python.

Mit Hilfe dieser Testfelder sollen dann die Simulationen (Experimente) im Excel-Tool vorgenommen werden. Die dadurch entstandenen Datenkombinationen aus vorgegebenen Systemparametern und berechneten Zielgrößen werden daraufhin zum Training der Metamodelle verwendet. Hier sollen mehrere Modelle trainiert, auf ihre Leistungsfähigkeit hin untersucht und anschließend das Modell mit der besten Performance zur weiteren Analyse ausgewählt werden. Zur Beurteilung der Leistungsfähigkeit bietet sich eine Analyse auf Basis von Testdaten an.

Anschließend sollen die Metamodelle der beiden Betrachtungsweisen genauer auf mögliche Gemeinsamkeiten und Interdependenzen zwischen einzelnen Variablen hin untersucht und Ergebnisse diskutiert werden. Dabei kommen als Visualisierungsmethoden der Partial Dependence Plot (*PDP)*, Individual Conditional Expectation Plot (*ICE)* als auch ein Ranking der einzelnen Variablen über die *Relative Importance* (RI) in Frage.

Das zweite Ziel besteht darin, ein Konventionelles Kommissioniersystem mit Hilfe eines Algorithmus und den Erkenntnissen aus dem vorigen Schritt optimieren zu können. Hierzu bietetet sich ein *Genetischer Algorithmus* (GA) auf Grund seiner vielfältigen Einsatzmöglichkeiten (Kramer 2017, S. 4) an.

Der Algorithmus soll in der Lage sein, eine optimale Konfiguration aus technischen Komponenten, für ein konventionelles Kommissioniersystem, bei einstellbaren Leistungsparametern zu finden. Dazu muss zunächst geklärt werden, inwiefern die Eigenschaften des GA zur Lösung des Optimierungsproblems genutzt werden können. Weiterhin muss vorgegeben werden, auf Grundlage welcher Größe die Optimierung durchgeführt werden soll und welche Nebenbedingungen zu berücksichtigen sind.

Ferner muss der GA in Python implementiert und im letzten Schritt auf seine Leistungsfähigkeit hin getestet werden.

Die in dieser Arbeit verwendete Rechnerkonfiguration ist in A-1 (S. 202), die Programme in A-2 (S. 202) sowie alle benutzten Python-Pakete in A-3 (S. 203) zu finden. Des Weiteren gilt die folgende Notation:

- Skalare: *kursive* Kleinbuchstaben (x, y, z)
- Vektoren: **fette** Kleinbuchstaben $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$
- Matrizen: **FETTE** Großbuchstaben $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})$
- Variablen und Funktionen im Quellcode:
  - ► Schriftart `consolas` (`funktion_x`, `variable_y`, `modell_z`)

2 Theoretische Grundlagen

2.1 Kommissioniersysteme

Wenn ein Kunde bei einem Lieferanten Artikel bestellt, gibt er dabei den gewünschten Artikel, die geforderte Menge und weitere Details der Bestellung genau an. Daraus ergeben sich für den Lieferanten zwei mögliche Konsequenzen:

Ist die Bestellung des Kunden sortenrein bzw. entspricht genau einer Lagereinheit, dann muss diese lediglich ausgelagert und versandt werden. Dieser Fall tritt jedoch vergleichsweise nur sehr selten ein. Typischer ist hingegen eine Bestellung, welche sich aus unterschiedlichen Artikeln mit verschiedenen Mengen zusammensetzt. Dies erfordert jedoch eine Vereinzelung und Entnahme der vom Kunden bestellten Artikel und wird als *Kommissionierung* bezeichnet (Hompel et al. 2011, S. 4).

Nach der VDI-Richtlinie VDI 3590 Blatt 1 wird das Kommissionieren wie folgt definiert:

> *„Kommissionieren hat das Ziel, aus einer Gesamtmenge von Gütern (Sortiment) Teilmengen (Artikel) auf Grund von Anforderungen (Aufträgen) zusammenzustellen.“*

Das effiziente Kommissionieren gilt als eine der anspruchsvollsten Aufgaben der Intralogistik, was einer Vielzahl von möglichen Techniken, Strategien und den daraus resultierenden Kombinationsmöglichkeiten geschuldet ist. Dabei trägt das Kommissionieren einen großen Anteil des möglichen Erfolges eines Logistikunternehmens. Dies liegt zum einen daran, dass die Kommissionierung der personalintensivste Bereich innerhalb der Distributionslogistik ist und damit eine erhebliche Kostenstelle darstellt. Zum anderen erwartet der Kunde, dass die Kommissionierung einwandfrei funktioniert, damit seine Bestellungen adäquat vom Lieferanten abgewickelt werden, um seine Bedürfnisse zu erfüllen (Gudehus 2010, S. 659; Hompel et al. 2011, S. 3).

Im Umfeld der Kommissionierung wird grundsätzlich zwischen den *Kundenaufträgen* und *Kommissionieraufträgen* unterschieden:

Kundenaufträge sind immer genau einem Kunden zugeordnet, wohingegen Kommissionieraufträge entweder aus mehreren ganzen Kundenaufträgen bestehen oder auch nur aus deren Teilen. Der Kommissionierauftrag umfasst dabei alle *Grundinformationen* der Artikel, die zur Kommissionierung notwendig sind. Als Grundinformationen werden hierbei die Informationen zur Artikelidentifikation und der Bestellmenge eines Artikels bezeichnet.

Jeder dieser Kommissionieraufträge umfasst eine oder mehrere *Positionen*. Es handelt sich hierbei um einzelne Zeilen des Auftrages mit Informationen zu den Artikeln (z.B. Entnahmeort und -menge), die kommissioniert werden sollen. Darüber hinaus umfassen die Positionen wiederum mehrere *Entnahmeeinheiten*, welche die kleinsten entnehmbaren Gebinde eines Artikels bezeichnen.

Der sogenannte *Pick* (z. Dt. „Greifen“) stellt das Herausnehmen einer einzelnen Entnahmeeinheit dar. Bei Kleinteilen kann dieser jedoch auch gleich mehrere Entnahmeeinheiten umfassen. Der Pick an sich wird durch einen *Kommissionierer* ausgeführt, bei dem es sich um einen Menschen, Automaten, Roboter oder eine Abzugsvorrichtung handeln kann.

Alle Entnahmeeinheiten eines bestimmten Artikels, welche dem Kommissionierer zur Entnahme zur Verfügung stehen, werden *Bereitstelleinheit* genannt.

Bei der *Sammel-* bzw. *Kommissioniereinheit* handelt es sich wiederum um einen Behälter oder eine Palette, in der die entnommenen Entnahmeeinheiten gesammelt werden. Für den Fall, dass ein Kommissionierauftrag mehrere Kundenaufträge umfasst, werden dem Kommissionierer auch dementsprechend mehrere Sammeleinheiten bereitgestellt bzw. von ihm mitgeführt (Gudehus 2010, S. 659; Hompel et al. 2010, S. 18; VDI-Richtlinie VDI 3590 Blatt 1).

2.1.1 Grundlagen der Kommissionierung

Gemäß der VDI-Richtlinie VDI 3590 Blatt 1 setzt sich ein Kommissioniersystem grundsätzlich aus drei Teilsystemen zusammen:

- Materialflusssystem
- Informationssystem

- Organisationssystem

2.1.1.1 Materialflusssystem

Das Materialflusssystem ist für den physischen Transport der Güter verantwortlich und kann nach der VDI-Richtlinie VDI 3590 Blatt 1 in folgende Funktionen unterteilt werden:

- Transport der Güter zur Bereitstellung
- Bereitstellung
- Fortbewegung des Kommissionierers
- Entnahme
- Transport der Entnahmeeinheit zur Abgabe
- Abgabe der Entnahmeeinheit
- Transport der Kommissioniereinheit zur Abgabe
- Abgabe der Kommissioniereinheit
- Rücktransport angebrochener Einheiten

Es gilt jedoch zu erwähnen, dass nicht jedes Kommissioniersystem all diese Funktionen abbildet. Nur die absoluten Grundfunktionen Bereitstellung, Entnahme und Abgabe sind in jedem Kommissioniersystem zwingend notwendig (Hompel et al. 2011, S. 21; Martin 2016, S. 405). Um nicht zu weit auszuschweifen, sollen im Folgenden auch nur diese Grundfunktionen, zusammen mit der Fortbewegung, näher beschrieben werden.

Eine Übersicht der Konfigurationsmöglichkeiten für Materialflusssysteme im Bereich der Kommissionierung kann in A-1 (S. 202) eingesehen werden.

Bereitstellung und Beschickung

Anhand der *Bereitstellungsart* kann unterschieden werden, wie der Kommissionierer die Güter zur Entnahme vorfindet (VDI-Richtlinie VDI 3590 Blatt 1).

Das erste Kriterium beschreibt den Ort der Bereitstelleinheit: Ist dieser an einem festen Platz, spricht man von einer *statischen* Bereitstellung. Muss die Bereitstelleinheit jedoch zur darauffolgenden Entnahme bewegt und im Anschluss an die Entnahme wieder rückgelagert werden, liegt eine *dynamische* Bereitstellung vor. Die stationären Bereitstellungsplätze können zudem nach deren Anordnung unterteilt werden: Liegen

diese nebeneinander, liegt der *eindimensionale Fall* vor. Sind sie zudem noch übereinander angeordnet, handelt es sich um den *zweidimensionalen Fall* (Gudehus et al. 2008, S. 676; Hompel et al. 2010, S. 21).

Weiterhin wird zwischen *zentraler* und *dezentraler* Bereitstellung unterschieden. Damit wird der Ort der Entnahme beschrieben. Bei der zentralen Bereitstellung wird die Entnahme an einem einzigen Ort durchgeführt, wohingegen die dezentrale Bereitstellung eine Entnahme an verschiedenen Orten vorsieht (Hompel et al. 2011, S. 23).

Letztlich kann die Bereitstellung auch noch nach dem Ordnungsgrad der bereitgestellten Güter differenziert werden: Die *geordnete*, *teilgeordnete* und *ungeordnete* Bereitstellung. Dabei besagt der Ordnungsgrad, in welcher Lage und Orientierung die Teile vom Kommissionierer aufgefunden werden. Ein hoher Ordnungsgrad ist gerade dann von Bedeutung, wenn die Entnahme der Güter durch eine Maschine automatisch erfolgen soll (Hompel et al. 2011, S. 23).

Die *Beschickung* hingegen umfasst alle Transporte, welche erforderlich sind, um eine Bereitstellung der Güter für den Kommissionierer zu ermöglichen. Dabei kann diese entweder *räumlich kombiniert,* also von der Entnahmeseite, aus oder *räumlich getrennt* erfolgen. Bei der räumlich getrennten Beschickung werden die Bereitstellplätz von deren Rückseite aus beschickt (Gudehus et al. 2008, S. 676; VDI-Richtlinie VDI 3590 Blatt 1).

Fortbewegung

Die *Fortbewegung* beschreibt die Art und Weise, in der sich der Kommissionierer zum Bereitstellort der zu entnehmenden Güter bewegt (VDI-Richtlinie VDI 3590 Blatt 1).

Bei der *eindimensionalen* Fortbewegung bewegt sich der Kommissionierer auf einer horizontalen Ebene. Dabei kann er zu Fuß mit z.B. einem Handwagen von Bereitstellort zu Bereitstellort laufen (Abbildung 1, links) oder bspw. mit einem Horizontal-kommissioniergerät verfahren. Die *zweidimensionale* Bewegung ergibt sich durch eine Erweiterung des Bewegungsradius in die Vertikale (Abbildung 1, rechts). Dies hat vor allem den Vorteil, dass bei einer großen Zugriffsfläche und einer geringen Anzahl von Entnahmeorten Wegzeiten eingespart werden können. Des Weiteren kann so die Höhe des Raumes besser ausgenutzt und somit die benötigte Gesamtfläche des Kommissionierlagers

reduziert werden. Der Kommissionierer selbst kann z.B. mit einem Man-Up-Stapler zum Entnahmeort gelangen, wie es ebenfalls rechts in Abbildung 1 zu sehen ist. Letztlich könnte sich der Kommissionierer auch mit Hilfe eines Krans in allen *drei Dimensionen* bewegen. Dies hat jedoch für die Kommissionierung keine praktische Relevanz (Gudehus et al. 2008, S. 676–677; Hompel et al. 2010, S. 22).

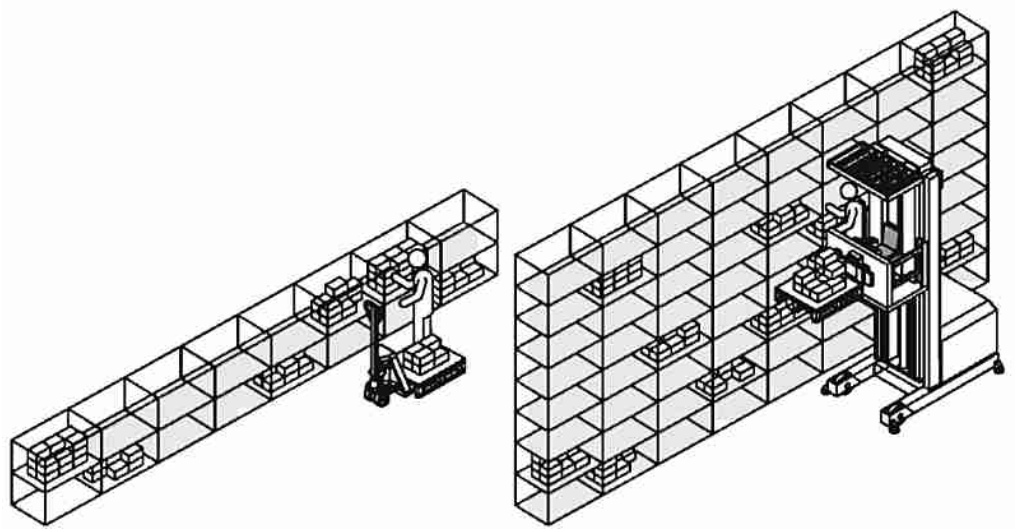


Abbildung 1: Ein- und zweidimensionale Fortbewegung (Hompel et al. 2011, S. 24)

Entnahme

Als *Entnahme* wird der Vorgang bezeichnet, in dem der Kommissionierer auf die bereitgestellten Güter zugreift (VDI-Richtlinie VDI 3590 Blatt 1).

Es bestehen einige technische Möglichkeiten, die Entnahme durchzuführen: Zum einen kann sie durch einen Menschen rein *manuell* oder mit einer mechanischen Greifhilfe ausgeführt werden oder *automatisch* bspw. durch einen Greifroboter.

Da die Entnahme selbst auch die Vereinzelung der bereitgestellten Güter in die Entnahmeeinheiten beinhaltet, müssen einige Vorkehrungen getroffen werden, um eine automatische Entnahme zu gewährleisten: Die Güter sollten eine gewisse Gleichartigkeit in ihrer Form sowie eine geeignete Oberfläche aufweisen. Zudem sollten die Entnahmeeinheiten entweder in einem gleichbleibenden Ordnungsschema gestapelt oder einzeln bereitgestellt werden (Gudehus et al. 2008, S. 677).

Abgabe

Die Form der *Abgabe* definiert den Ort und die Form der Ablage der kommissionierten Güter (VDI-Richtlinie VDI 3590 Blatt 1).

Bei dem Abgabeort kann es sich entweder um eine einzige *zentrale* Basisstation handeln oder unterschiedliche *dezentrale* Abgabeplätze (Gudehus et al. 2008, S. 677; Hompel et al. 2010, S. 23).

Zudem kann zwischen *statischer* und *dynamischer* Abgabe unterschieden werden: Bei der statischen Abgabe erfolgt die Abgabe der Entnahmeeinheit in eine fördertechnisch unbewegte Sammel- bzw. Kommissioniereinheit, wohingegen bei der dynamischen Abgabe die Entnahmeeinheit auf einen Stetigförderer erfolgt (z.B. ein Fließ- oder Rollenband) (Hompel et al. 2010, S. 23).

Auch bei der Abgabe kann wieder nach dem Ordnungsgrad der Teile (*geordnet*, *teilgeordnet* und *ungeordnet*) differenziert werden. Dies ist dann von Bedeutung, wenn nachfolgende Schritte automatisiert werden sollen. Je höher der Ordnungsgrad hier ist, desto einfacher ist eine Automatisierung der darauf folgenden Arbeitsschritte (Hompel et al. 2010, S. 23).

2.1.1.2 Informationssystem

Laut der VDI-Richtlinie VDI 3590 Blatt 1 setzt sich ein Informationssystem aus einzelnen Informationselementen zusammen, die notwendig sind, um Materialflussvorgänge in einem Kommissioniersystem zu initialisieren und/oder durchzuführen:

- Auftrag
- Kommissionierliste/-datei
- Position

Das Informationssystem hat einen großen Anteil an der Funktion und Effizienz des Kommissioniersystems. Insbesondere die korrekte Erfassung des Systemzustandes und der aktuellen Bestände ist hier maßgeblich. Weiterhin kann der Informationsfluss in vier Grundfunktionen gegliedert werden (Hompel et al. 2011, S. 28):

- Erfassung der Kundenaufträge

- Auftragsaufbereitung
- Weitergabe des Kommissionierauftrages
- Quittierung der Artikelentnahme

Für weitere Informationen wird an dieser Stelle auf Hompel et al. 2011, S. 28–32 verwiesen.

2.1.1.3 Organisationssystem

Bei Kommissioniersystemen setzt sich das Organisationssystem im Allgemeinen aus drei Teilsysteme zusammen (Hompel et al. 2011, S. 32; VDI-Richtlinie VDI 3590 Blatt 1):

- Aufbauorganisation (Anordnungsstruktur der Lagerbereiche; Infrarstruktur)
- Ablauforganisation (Abwicklung des Kommissionierens; Auftragsstruktur)
- Betriebsorganisation (Einlastung von Aufträgen; zeitliche Reihenfolge)

In Hompel et al. 2011, S. 32–41 und VDI-Richtlinie VDI 3590 Blatt 1 werden die drei Teilsysteme detailliert beschrieben.

2.1.1.4 Grundprinzipien

Neben den bisher beschriebenen Klassifizierungsmöglichkeiten von Kommissioniersystemen, können diese auch in die beiden Grundprinzipien *Person-zur-Ware*[1] (PzW) und *Ware-zur-Person*[2] (WzP) eingeteilt werden, wobei diese auch miteinander kombiniert werden können (Hompel et al. 2011, S. 42).

Das Prinzip PzW zeichnet sich dadurch aus, dass sich der Kommissionierer aktiv zur statisch, an einem festen Ort bereitgestellten Bereitstelleinheit hinbewegen muss, um diese zu entnehmen. Dabei ist die Bereitstelleinheit in entsprechenden Lagermitteln vorgehalten. Ein Beispiel hierfür wäre, dass sich der Kommissionierer mit einem Handwagen in einem Regallager von Entnahmeort zu Entnahmeort (eindimensional) bewegt und die geforderte Entnahmemenge in Sammelbehälter ablegt. Sind alle Positionen des Kommissionierauftrages abgearbeitet, gibt der Kommissionierer die Sammelbehälter an einer zentralen Abgabestelle ab (siehe Abbildung 2) (Bichler et al. 2010, S. 210; Hompel et al. 2011, S. 41).

[1] Wird auch als „Mann-zur-Ware“ bezeichnet
[2] Wird auch als „Ware-zum-Mann“ bezeichnet

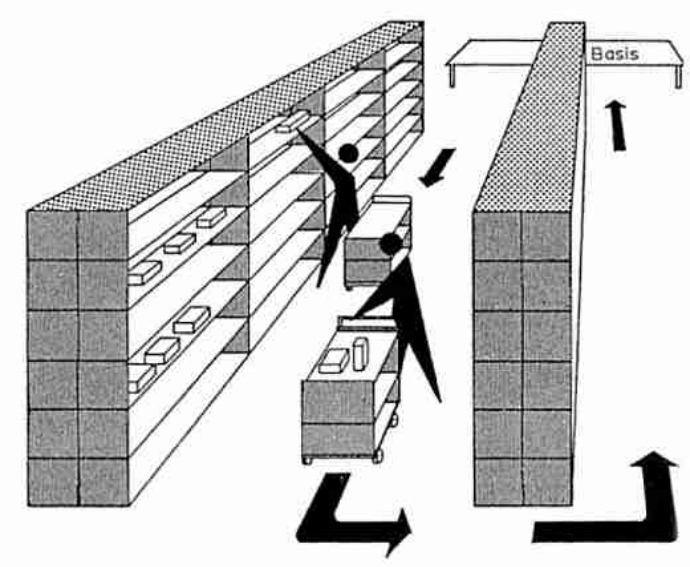


Abbildung 2: Person-zur-Ware Beispiel (Gudehus 2010, S. 684)

Bei dem Prinzip WzP werden die Bereitstelleinheiten i.d.R. automatisch zum Kommissionierer hinbewegt, welcher daraufhin die Entnahme der Güter ausführt. Anschließend wird die angebrochene Bereitstelleinheit entweder wieder zurück ins Lager transportiert oder in einer sogenannten Anbruchzone gesammelt. Ein Beispiel hierfür ist das sogenannte Kommissionierer-U (Abbildung 3). Der Kommissionierer ist hier statisch vor Kopf des Us angeordnet. Ihm werden die jeweiligen Bereitstelleinheiten über ein automatisiertes Fördersystem zugeführt und nach erfolgreicher Entnahme der Artikel wieder abtransportiert (Bichler et al. 2010, S. 212; Hompel et al. 2011, S. 41–42).

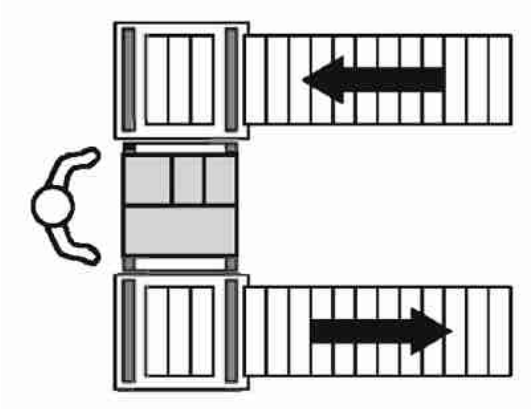


Abbildung 3: Ware-zur-Person Beispiel (Hompel et al. 2011, S. 42)

2.1.2 Technische Komponenten

Alle Kommissioniersysteme bestehen aus einer Reihe von miteinander verknüpften technischen Komponenten. Dabei existiert für jede dieser Komponenten eine Vielzahl von technischen Umsetzungen, wobei die Auswahl stets problemspezifisch ist. Eine Auswahl möglicher technischer Umsetzungen für jede dieser Komponenten ist in A-1 (S. 205) zu finden.

2.1.2.1 Lagermittel

Bei der Auswahl des passenden Lagermittels (LM) stehen drei Grundtypen zur Auswahl: statische, dynamische und automatische Lagerung (Bichler et al. 2010, S. 154, 174; Hompel et al. 2011, S. 43, 45): Bei der *statischen Lagerung* ruht das Lagergut an einem festen Ort im Regal, welches sich ebenfalls nicht bewegt. Bei der *dynamischen Lagerung* ist dies nicht der Fall. Hier wird das Lagergut nach der Einlagerung bewegt, wobei man hier nicht die dynamische Lagerung mit einer dynamischen Bereitstellung vertauschen sollte. Die dynamische Bereitstellung betrachtet die Beförderung der Lagereinheit zum Entnahmeort, wohingegen die dynamische Lagerung die Bewegung des Lagergutes während der Lagerung betrachtet. Die *automatische Lagerung* stellt praktisch eine nach Außen abgeschlossene Lager- und Kommissionier-Maschine dar, die i.d.R. als WzP-System fungiert.

2.1.2.2 Fördermittel

Die Fördertechnik dient dazu, Güter und Personen innerhalb eines Systems über relativ kurze Distanzen zu bewegen. Dabei werden die Fördermittel (FM) neben dieser Hauptaufgabe ebenfalls zum Verteilen, Sortieren, Puffern, Sammeln und zur Kommissionierung eingesetzt. Die Fördersysteme können nach den folgenden Kriterien differenziert werden (Gleißner und Femerling 2008, S. 99; Hompel et al. 2011, S. 45–46):

- *Unstetigförderer*: fördern in einzelnen Arbeitsspielen, unterbrochener Förderstrom
- *Stetigförderer*: erzeugen einen kontinuierlichen Förderstrom
- *Flurgebunden*: fahren auf dem Boden bzw. auf in den Boden eingelassenen Einrichtungen
- *Aufgeständert*: verfahren über dem Boden auf Stützen oder Schienen
- *Flurfrei*: bewegen sich auf Schienen, welche an der Hallendecke angebracht sind

Die Kommissionierleistung eines Systems wird in hohem Maße durch die Kapazität des FM für Sammelbehälter, in Kombination mit der Fortbewegungsart, Geschwindigkeit und Beschleunigung vorgegeben, wohingegen die für das FM notwendige Gangbreite und die damit erreichbare Greifhöhe den Raumbedarf eines Kommissioniersystems beeinflussen (Gudehus 2010, S. 683).

2.1.2.3 Handhabungsmittel

Bei den Vorgängen in einem Kommissioniersystem, bei denen ein menschlicher Kommissionierer nicht direkt beteiligt ist, werden Handhabungsmittel zum Greifen und Bewegen von Gütern und Ladeeinheiten verwendet. Eine Kombination aus Handhabungs- und Fördermitteln wird mitunter zur Sortierung von Gütern eingesetzt (bspw. ein Sorter) (Hompel et al. 2011, S. 50).

2.1.2.4 Ladehilfsmittel

Der Einsatz von Ladehilfsmitteln (LHM) ist für das Kommissionieren essenziell, da mit ihnen eine rationelle Bündelung von einzelnen Artikeln zu Ladeeinheiten erfolgen kann. Die Artikel werden dabei auf oder in ihnen gelagert bzw. transportiert. Des Weiteren kommen sie als Hilfsmittel zur Konsolidierung von Positionen des Kommissionier-auftrages und zum Transport der kommissionierten Artikel durch das gesamte Kommissioniersystem zum Einsatz. Sie werden dabei in *tragende* um *umschließende* LHM unterteilt (Gleißner und Femerling 2008, S. 117; Hompel et al. 2011, S. 51).

2.1.2.5 Informationstechnische Kommissioniererführung

Nachdem die Kundenaufträge in Form von Kommissionieraufträgen vorliegen, hat die *Informationstechnische Kommissioniererführung* (ITK) die Aufgabe, dem Kommissionierer die zum Kommissionieren notwendigen Informationen bereitzustellen. Dazu zählen Angaben zu Art und Menge der gewünschten Artikel sowie dem Ort, an dem diese bereitgestellt sind. Um die Arbeit effizienter zu gestalten, kann die ITK zusätzlich Informationen zum idealen Weg des Kommissionierers, mögliche Fehlmengen und Kundendaten bereitstellen. Die ITK wird zum einen in beleggebundene, in Form einer Pickliste, und beleglose Verfahren (bspw. Pick-by-voice) unterteilt. Bei der beleggebundenen Variante ist kein zentraler bzw. mobiler Rechner zur online

Koordinierung nötig – allerdings zu dem Preis, dass Kommissionierfehler erst nach einer zeitlichen Verzögerung auffallen (Hompel et al. 2011, S. 52; Thomas et al. 2008, S. 808).

2.1.3 Betriebsstrategien

Die Leistung und (Betriebs-)Kosten eines Kommissioniersystems lassen sich in hohem Maße von der Wahl der Betriebsstrategien beeinflussen. Diese Betriebsstrategien können in folgende Kategorien eingeteilt werden (Gudehus 2010, S. 703):

- Lagerplatzvergabestrategien
- Bearbeitungsstrategien
- Wegstrategien
- Entnahmestrategien
- Nachschubstrategien
- Leergutstrategien

Im Folgenden sollen jedoch nur die Lagerplatzvergabe- und die Wegstrategien näher betrachtet werden. Die übrigen Strategien bleiben hier außen vor, da sie im Rahmen dieser Arbeit nicht von Relevanz sind. Sie können jedoch in Gudehus 2010 und in Hompel et al. 2011 nachgeschlagen werden.

2.1.3.1 Lagerplatzvergabestrategien

Die Strategie der Lagerplatzvergabe ist ein wichtiges Hilfsmittel, um die Leistung eines Kommissioniersystems zu optimieren. Dabei wird für die Artikel festgelegt, in welchen Lagerplätzen sie zum Kommissionieren bereitgestellt werden. Dabei sind die Ziele dieser Strategien: Bessere Platzausnutzung des Lagers, kurze Wegzeiten sowie ein geringer Nachschubaufwand (Gudehus 2010, S. 704; Hompel et al. 2011, S. 92).

Die Lagerplatzvergabestrategien können in *Festplatzstrategien* und *Freiplatzstategien* gegliedert werden. Bei den Festplatzstrategien ist jeder Artikel des Sortiments einem festen Lagerplatz zugeordnet. Ist der Artikel zu einen gegebenen Zeitpunkt nicht verfügbar, bleibt das entsprechende Lagerfach leer. Der Vorteil liegt hier darin, dass es sehr einfach und ohne IT-Einsatz zu organisieren ist. Zudem kann die Pickfolge über die Anordnung der Lagerplätz konfiguriert werden (Gleißner und Femerling 2008, S. 131; Gudehus 2010, S. 705).

Bei der Freiplatzstrategie[3] wird jedem Artikel von einem zentralen Lagerverwaltungssystem ein freier Platz zugewiesen. Dabei besteht keinerlei Zusammenhang zwischen zwei aufeinander folgenden Einlagerungen. Die Zuweisung kann hier auf zwei Arten geschehen: Zum einen können die Lagerplätze den Artikeln so zugeordnet werden, dass dadurch die kürzesten Fahrzeiten (KFZ) zwischen dem Ein- und Auslagerpunkten der Artikel erreicht werden. Zum anderen können die Artikel auch in Zonen nach deren Umschlagshäufigkeit einsortiert werden. Dabei sind die Artikel mit einer hohen Anzahl an Zugriffen so angeordnet, dass sie vom Kommissionierer auf kurzen Wegen zu erreichen sind. Die Aufteilung in Zonen kann wiederum in die ABC-Zonung[4] und kontinuierlicher Zonung eingeteilt werden. Bei der ABC-Zonung existieren wiederum verschiedene Ansätze. Einer dieser Ansätze wird beispielhaft in Abbildung 4 gezeigt (Gleißner und Femerling 2008, S. 131; Gudehus 2010, S. 705; Hompel et al. 2011, S. 92–94).

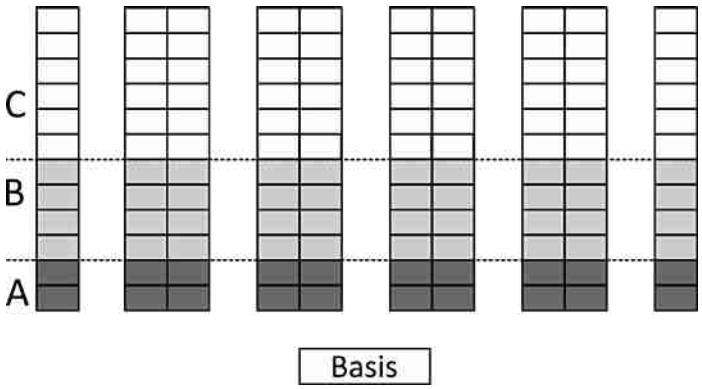


Abbildung 4: ABC-Zonen mit Streifensystem nach Hompel et al. 2010, S. 93

2.1.3.2 Wegstrategien

Wegstrategien eines Kommissioniersystems werden grundsätzlich nach PzW- und WzP-Systemen unterschieden. Da in dieser Arbeit der Fokus auf dem konventionellen Kommissionieren, auf Grund der weiten Verbreiteung, liegt, werden im folgenden Abschnitt lediglich Wegstrategien für PzW-Kommissioniersysteme kurz dargestellt (Hompel et al. 2011, S. 138).

[3] Wird auch als „Chaotische Lagerung“ bezeichnet

[4] ABC bezeichnet Einteilung der Artikel in Klassen nach deren Zugriffshäufigkeit. Dabei wird auf A-Artikel sehr oft zugegriffen, wohingegen auf C-Artikel selten zugegriffen wird. Die sogenannte „ABC-Analyse“ kann auch in Hompel et al. 2011 nachgelesen werden.

Bevor weiter auf die einzelnen Strategien eingegangen werden kann, müssen zwei Betrachtungsweisen unterschieden werden: Die *gassengebundene Betrachtung* geht davon aus, dass ein Kommissionierer nur innerhalb einer Gasse arbeitet, also an eine Gasse gebunden ist. Die *gassenungebundene Betrachtung* geht dagegen davon aus, dass sich die Kommissionierer zum Abarbeiten eines Kommissionierauftrages durch das gesamte Kommissioniersystem bewegen können und müssen (Hompel et al. 2011, S. 140).

Gassengebundene Betrachtung

Bei der gassengebundenen Betrachtung kann zum einen zwischen einer *ungeordneten* und *geordneten Ansteuerung*, der einzelnen Positionen des Kommissionierauftrages durch den Kommissionierer im Gang differenziert werden, zum anderen ergibt sich aus der Start- und Zielposition des Kommissionierers bei jedem Auftrag ein weiteres Unterscheidungsmerkmal (Hompel et al. 2011, S. 141–144).

Abbildung 5 zeigt die vier daraus resultierenden Fälle: In a) und b) sind die ungeordneten Zustände abgebildet, wohingegen c) und d) den Geordneten zeigen. Die Zahlen kennzeichnen die Reihenfolge der Entnahmen durch den Kommissionierer. Weiterhin sind bei a) und c) der Start und das Ziel identisch, bei b) und d) ist dies nicht der Fall.

Eine detaillierte Beschreibung ist in Hompel et al. 2011, S. 140–154 zu finden.

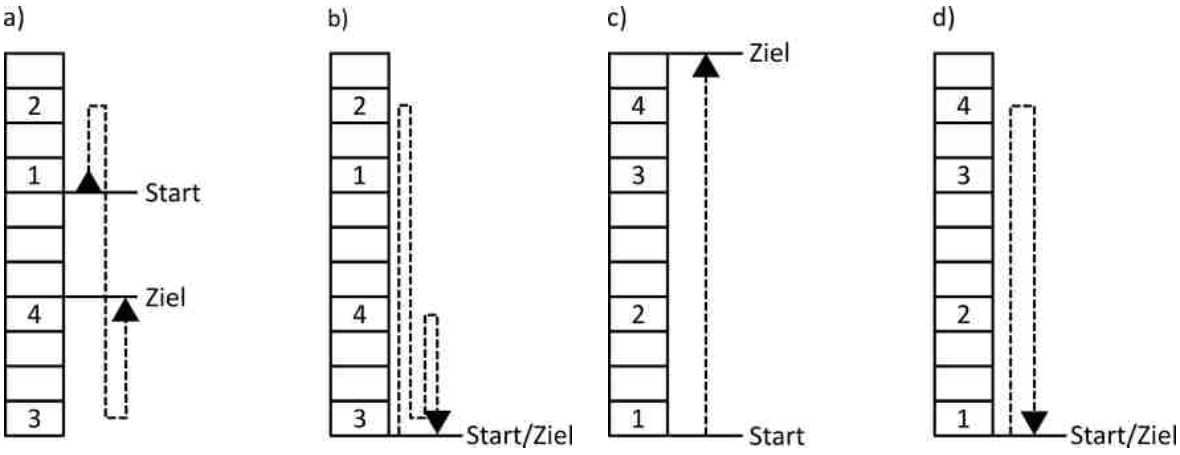


Abbildung 5: Gassenungebundene Betrachtung nach Hompel et al. 2011, S. 141–144

Gassenungebundene Betrachtung

Bei der gassenungebundenen Betrachtung kann das Layout eines Kommissionierlagers entweder die Form eines *Kopfganglayouts* oder eines *Zentralganglayouts* annehmen (siehe

Abbildung 6). Beim Zentralganglayout wird der zentrale Versorgungsgang mittig der Gassenlänge des Kopfganglayouts platziert (Hompel et al. 2011, S. 155, 163).

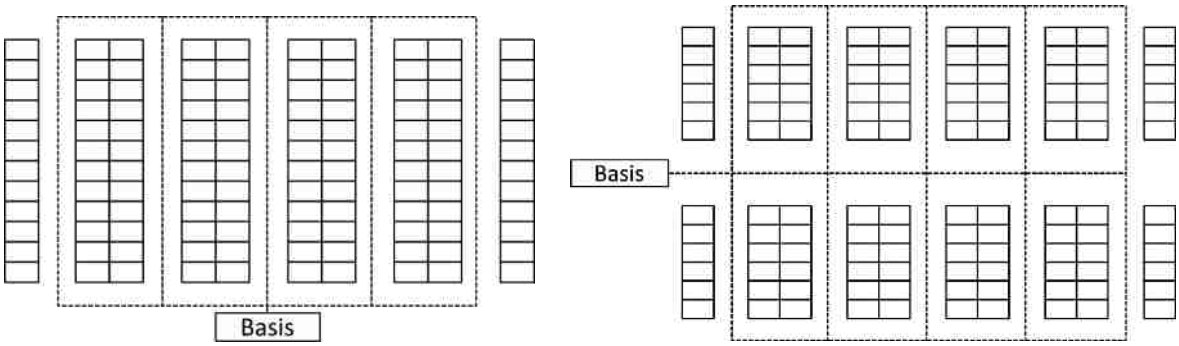


Abbildung 6: Kopf- und Zentralganglayout nach Hompel et al. 2011, S. 155

Dadurch, dass sich der Kommissionierer bei der gassenungebundenen Betrachtung durch das gesamte Kommissionierlager bewegen kann, ergeben sich dementsprechend wesentlich mehr Möglichkeiten, die Positionen eines Kommissionierauftrages anzusteuern. Dazu haben sich einige recht einfache Heuristiken etabliert, welche eine möglichst optimale Wegausnutzung garantieren sollen.

Bei der *Durchgang- oder Schleifenstrategie* durchschreitet bzw. durchfährt der Kommissionierer in Schleifenlinien das Kommissionierlager. Dabei kann er Gänge, in denen keine Positionen des Kommissionierauftrages liegen, auslassen, wie es bei der *Schleifenstrategie mit Überspringen* (Abbildung 7 a) der Fall ist. Er kann jedoch auch dazu gezwungen werden, alle Gänge zu durchlaufen, egal ob dort Artikel entnommen werden oder nicht. In diesem Fall handelt es sich um eine *Schleifenstrategie ohne Überspringen* (Abbildung 7 b) (Gudehus 2010, S. 711; Hompel et al. 2011, S. 95–96).

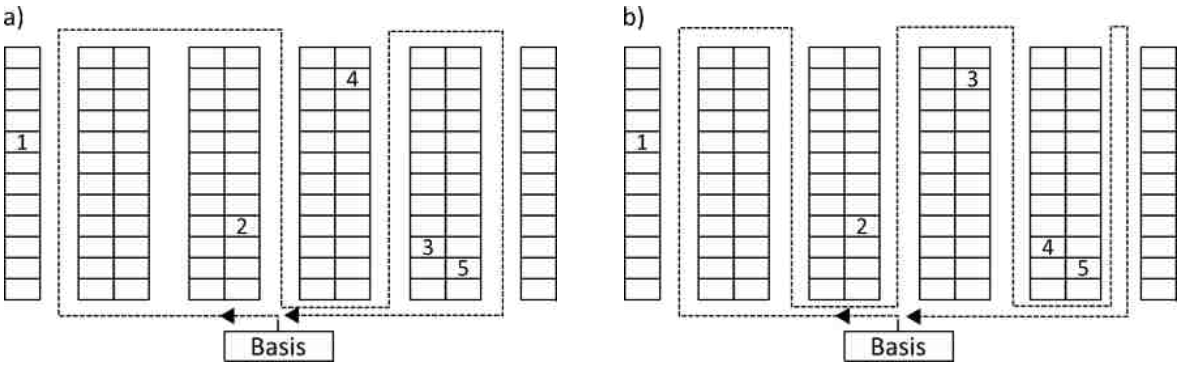


Abbildung 7: Schleifenstrategie mit/ohne Überspringen nach Hompel et al. 2011, S. 96

Für die *Stichgangstrategie* existieren ebenfalls zwei Varianten. Die *Stichgangstrategie mit Gangwiederholung* (siehe Abbildung 8 a) zeichnet sich dadurch aus, dass der Kommissionierer die Gassen nur von der Stirnseite aus, für jede Position einzeln, betritt. Bei der *Stichgangstrategie ohne Gangwiederholung* (Abbildung 8 b) ist es möglich, alle Positionen einer Gasse in nur einem einzigen Gang zu picken (Gudehus 2010, S. 711; Hompel et al. 2011, S. 97).

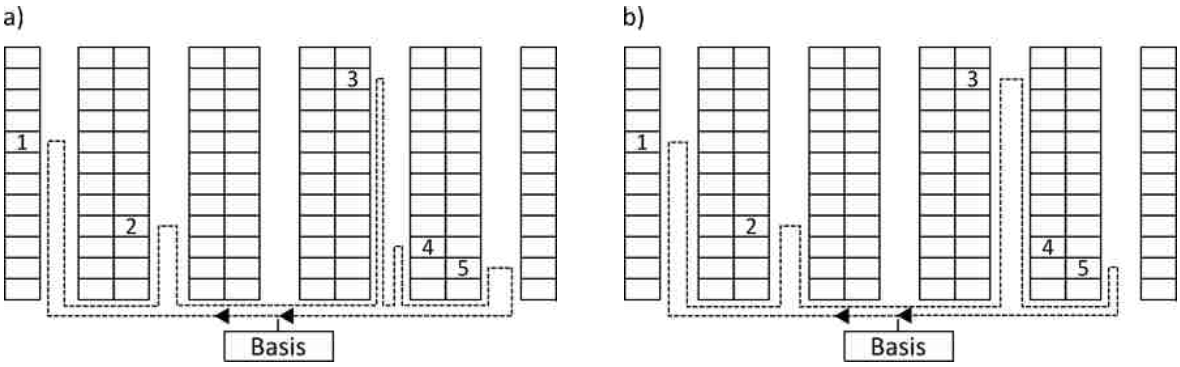


Abbildung 8: Stichgangstrategie mit/ohne Gangwiederholung nach Hompel et al. 2011, S. 97

Bei der *Mittelpunkt-Heuristik* (Abbildung 9) handelt es sich um eine Sonderform der Stichgangstrategie. Hierbei wird die Gasse in eine vordere und hintere Hälfte halbiert. Bei jedem Betreten einer Gasse (zur Entnahme der Artikel) durchschreitet der Kommissionierer sie maximal bis zur Hälfte (Hompel et al. 2011, S. 98).

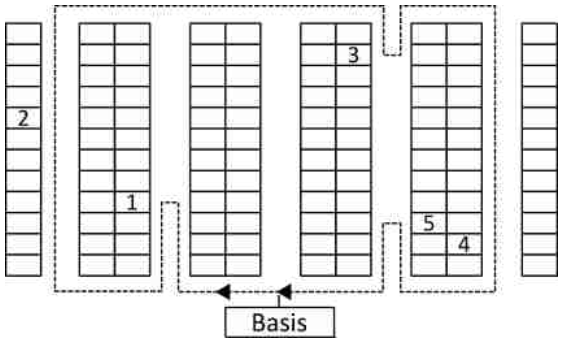


Abbildung 9: Mittelpunkt-Heuristik nach Hompel et al. 2011, S. 98

Es existieren noch weitere Ansätze, um die Wege des Kommissionierers zu optimieren, wie bspw. die *Largest-Gap-Heuristik*. Allerdings können diese durch das vorhandene Excel-Tool nicht abgebildet werden und bleiben deshalb außenvor.

2.1.4 Konventionelles Kommissionieren

Das *Konventionelle Kommissionieren* ist sozusagen der Klassiker unter den PzW-Kommissionierverfahren. Der sehr simple Grundaufbau erlaubt eine sehr hohe Flexibilität bzgl. der gelagerten Güter und des notwendigen Personaleinsatzes. Dadurch, dass die Investitionskosten ebenfalls niedrig sind, ist das Konventionelle Kommissionieren bis dato das am weitesten verbreitete Kommissionierverfahren (Gudehus 2010, S. 669; Hompel et al. 2011, S. 67).

Tabelle 1: Konventionelles Kommissionieren

Bereitstellung		Fortbewegung	Entnahme	Abgabe	
statisch	dezentral	eindimensional	manuell	statisch	zentral

Beim Konventionellen Kommissionieren (Tabelle 1) bewegt sich der Kommissionierer mit einem Sammelbehälter und/oder FM ebenerdig durch das Lager. Dabei bewegt sich der Kommissionierer in einem Rundgang nacheinander zu den statischen Bereitstellplätzen der Artikel seines Kommissionierauftrages, an denen er die geforderten Mengen manuell in die mitgeführten Sammelbehälter oder auf ein FM ablegt (Abbildung 10). Die Bereitstelleinheiten sind entweder nebeneinander am Boden oder in Regalen möglichst platzsparend angeordnet. Die ITK-Variante ist hier frei wählbar. Nachdem alle Positionen des Kommissionierauftrages durch den Kommissionierer abgearbeitet wurden, gibt er den bzw. die Sammelbehälter an einer zentralen Basis ab und beginnt einen neuen Auftrag (Gudehus 2010, S. 669; Hompel et al. 2011, S. 66–67).

Bei der Verwendung des Konventionellen Kommissionierens ergeben sich einige Vor- und Nachteile. Zu den Vorteilen zählen unter anderen (Gudehus 2010, S. 669):

- Minimaler technischer Aufwand
- Kurze Auftragsdurchlaufzeiten
- Hohe Flexibilität gegenüber Leistungsanforderungen
- Eignung für alle Warengrößen

Wohingegen die Nachteile nicht vernachlässigt werden sollten (Gudehus 2010, S. 670):

- Hoher Grundflächenbedarf
- Bei breitem Sortiment und großen Bereitstellmengen ergeben sich hohe Wegzeiten

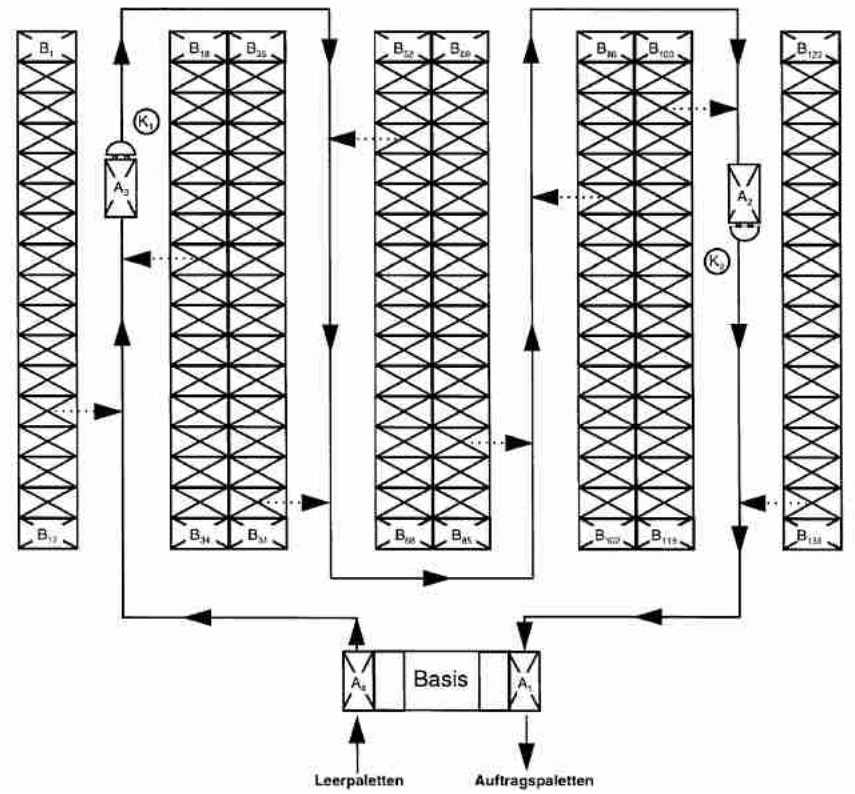


Abbildung 10: Konventionelles Kommissionieren (Gudehus 2010, S. 670)

Neben dem Konventionellen Kommissionieren existieren noch weitere PzW-Kommissioniersysteme, wie z.B. das *Kommissionieren im Hochregallager* oder im *Durchlaufregallager*. Diese werden bspw. in Gleißner und Femerling 2008, S. 94 und Hompel et al. 2011, S. 71 erläutert.

2.2 Statistische Versuchsplanung

Die Methode der *statistischen Versuchsplanung*[5] stammt schon aus den 20er Jahren des vergangenen Jahrhunderts und beschäftigt sich damit, Versuche so effizient zu planen, dass mit möglichst wenigen Versuchen ein Maximum an Informationen über ein System gewonnen werden kann. Dadurch können Zeit und Kosten für Versuche eingespart werden (Kleppmann 2013, S. 1; Siebertz et al. 2017, S. 1).

Um Versuche durchführen zu können, benötigt man zunächst ein System, welches erforscht werden soll. Dabei muss für das System genau definiert werden, wo es beginnt und wo es aufhört. Dies geschieht über *Systemgrenzen*. Weiterhin verfügt ein System über

[5] Engl. „Design of Experiment“ (DoE)

Inputs und *Outputs*. Bei den Inputs muss genau zwischen den Eingangsgrößen, die betrachtet werden, und denen, die vernachlässigt werden, differenziert werden. Eine Optimierung kann nur über die betrachteten Größen geschehen, wobei beachtet werden muss, dass die vernachlässigten Inputs das System weiterhin (z.B. als Störgrößen) beeinflussen können. Werden zu viele Größen betrachtet, sinkt der Wirkungsgrad der Untersuchung. Werden jedoch zu wenige mit einbezogen, bleiben viele Faktoren ungenutzt (Siebertz et al. 2017, S. 3).

Die Outputs, auch *Zielgrößen* oder *Qualitätsmerkmale* genannt, sind die messbaren Ergebnisse eines einzelnen Versuches. Sie bieten die Ausgangsbasis zur Unterscheidung zwischen guten und schlechten Versuchen. Dabei müssen die Zielgrößen kontinuierlich sein, ansonsten können keine Effekte der Eingänge berechnet werden. Ist dies nicht der Fall, müsste man auf den Einsatz von Hilfsgrößen zurückgreifen, die nichtkontinuierliche Größen umrechnen. Die Wahl der Systemgrenzen und Zielgrößen ist maßgeblich für den Erfolg bzw. Misserfolg des Versuchsplans verantwortlich (Kleppmann 2013, S. 12; Siebertz et al. 2017, S. 4).

Die Eingangsgrößen, welche als relevant angesehen werden und in den Versuchsplan einfließen sollen, werden *Faktoren* bzw. *Parameter* genannt. Dabei können die Faktoren in zwei Gruppen eingeteilt werden: Die Faktoren, welche analysiert werden und die Faktoren, welche beobachtet und weitestgehend konstant gehalten werden. Es sollten natürlich vornehmlich die Faktoren analysiert werden, bei denen man nach aktuellem Kenntnisstand davon ausgehen kann, dass sie die größte Wirkung auf die Zielgrößen des Systems haben. Zudem sollten im Zweifelsfall lieber zu viele als zu wenige Faktoren aufgenommen werden (Kleppmann 2013, S. 14; Siebertz et al. 2017, S. 5).

Für jeden der ausgewählten Faktoren müssen nun Werte bestimmt werden, die dieser Faktor einnehmen kann. Diese einzelnen Werte werden *Stufen* oder *Level* genannt. Dabei werden für jeden Faktor mindestens zwei unterschiedliche Stufen festgelegt. Der Effekt eines Faktors auf die Zielgröße hängt dabei i.d.R. vom Abstand zwischen den beiden Stufen ab. Ist dieser groß, ist der Effekt (bei weitestgehend linearen Systemen) auch groß. Die Faktoren können hier wiederum in zwei Gruppen unterteilt werden; diesmal nach der Art

der Faktorstufen in *quantitative* (*numerische*) und *qualitative* (*kategorische*) Faktoren. Die Stufen von quantitativen Faktoren können als Zahl auf einer Messskala dargestellt werden. Bei qualitativen Faktoren liegen nur Beschreibungen oder Bezeichnungen vor. Dementsprechend können hier auch keine Zwischenwerte der Stufen berechnet werden. Die einzelnen Stufen werden im Versuchsplan meist codiert dargestellt. Ein hoher Wert des Faktors bspw. als $+$ oder 1. Ein niedriger Wert dementsprechend als $-$ oder 0 (Kleppmann 2013, S. 14–15; Siebertz et al. 2017, S. 5–6).

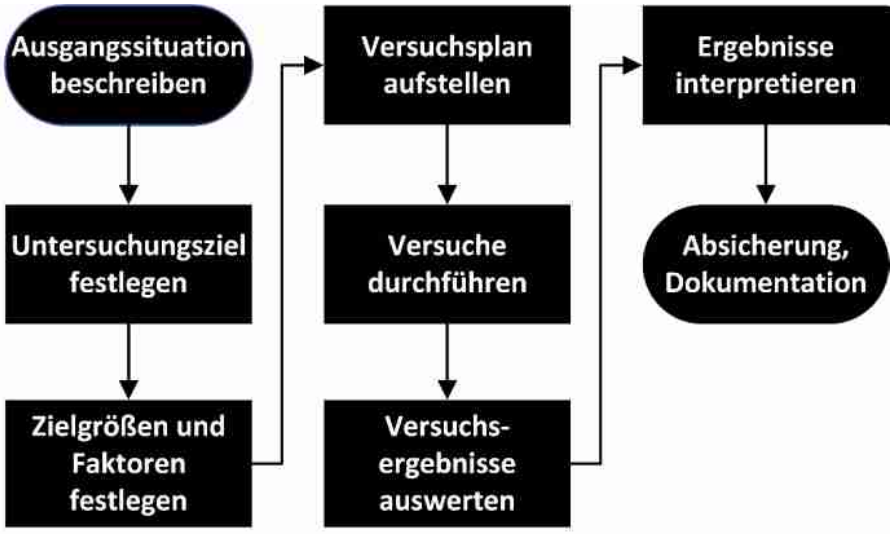


Abbildung 11: Schritte der Statistischen Versuchsplanung nach Kleppmann 2013

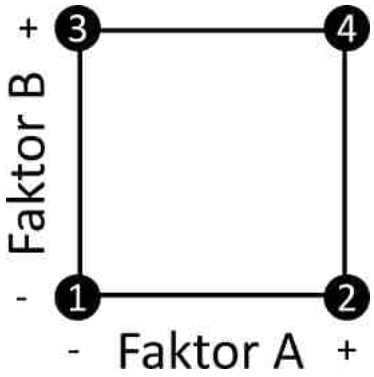
Abbildung 11 zeigt die notwendigen Schritte der statistischen Versuchsplanung. Auf die einzelnen Schritte wird hier nicht weiter eingegangen, da diese doch recht selbsterklärend sind. Bei Bedarf kann eine detaillierte Beschreibung der einzelnen Schritte in Kleppmann 2013 (Kapitel 3) nachgeschlagen werden.

2.2.1 Faktorielle Versuchspläne und Effekte

Vollständig faktorieller Versuchsplan

Der *vollständig faktorielle Versuchsplan*, auch *Vollfaktorplan* genannt, ist vom Prinzip der einfachste überhaupt vorstellbare faktorielle Versuchsplan. Hier wird jede mögliche Kombination der einzelnen Faktorstufen getestet. Dadurch ergeben sich bei Anzahl von n_F Faktoren und n_S Stufen je Faktor $N = {n_S}^{n_F}$ Versuche bzw. Samples. Abbildung 12 zeigt einen beispielhaften Vollfaktorplan mit zwei Faktoren (A, B) auf jeweils zwei Stufen $(+, -)$, also $2^2 = 4$ Faktorkombinationen mit den entsprechenden Zielgrößen y. Daher wird dieser

Versuchsplan auch als „2^2-Plan“ bezeichnet. Der linke Teil der Abbildung soll die vier Ecken des Lösungsraumes darstellen (Kleppmann 2013, S. 101; Siebertz et al. 2017, S. 7).



Versuch	Faktoren		Ergebnis
	A	B	
1	-	-	$y_1 = 30$
2	+	-	$y_2 = 70$
3	-	+	$y_3 = 50$
4	+	+	$y_4 = 15$

Abbildung 12: Vollständiger faktorieller 2^2-Versuchsplan

Aus den faktoriellen Versuchsplänen können *Effekte* der Faktoren relativ leicht berechnet werden. Der Effekt beschreibt die Wirkung eines Faktors auf das betrachtete System und berechnet sich aus der Differenz des Mittelwertes bei der Einstellung $+$ und des Mittelwertes bei der Einstellung $-$. Demzufolge wird dadurch die mittlere Zielgrößenveränderung beim Einstellungswechsel des Faktors beschrieben. Diese Prozedur wird als *Kontrastmethode* bezeichnet (Kleppmann 2013, S. 102; Siebertz et al. 2017, S. 12):

$$\bar{y}_{A_+} = \frac{y_2 + y_4}{2} \qquad \bar{y}_{A_-} = \frac{y_1 + y_3}{2}$$

$$\bar{y}_{B_+} = \frac{y_3 + y_4}{2} \qquad \bar{y}_{B_-} = \frac{y_1 + y_2}{2}$$

$$\text{Effekt}_A = \bar{y}_{A_+} - \bar{y}_{A_-} = \frac{y_2 + y_4}{2} - \frac{y_1 + y_3}{2}$$

$$\text{Effekt}_B = \bar{y}_{B_+} - \bar{y}_{B_-} = \frac{y_3 + y_4}{2} - \frac{y_1 + y_2}{2}$$

Nicht allzu selten kommt es vor, dass die Faktoren für sich genommen keinen großen Effekt auf das Qualitätsmerkmal haben. In Kombination mit den Einstellungen anderer Faktoren können sich diese Effekte jedoch gegenseitig beeinflussen. Man spricht hier von *Wechselwirkungen*. Der *Wechselwirkungseffekt* zeigt an, wie stark der Effekt des einen Faktors von dem anderen abhängt und vice versa. Der Wechselwirkungseffekt von B auf A berechnet sich durch:

$$(\text{Effekt}_A | B_+) = \left(\bar{y}_{A_+} \middle| B_+\right) - \left(\bar{y}_{A_-} \middle| B_+\right) = \frac{y_4}{1} - \frac{y_3}{1}$$

$$(\text{Effekt}_A|B_-) = \left(\bar{y}_{A_+}|B_-\right) - \left(\bar{y}_{A_-}|B_-\right) = \frac{y_2}{1} - \frac{y_1}{1}$$

$$\text{Wechselwirkungseffekt}_{AB} = \frac{(\text{Effekt}_A|B_+)}{2} - \frac{(\text{Effekt}_A|B_-)}{2} = \frac{y_1 - y_2 - y_3 + y_4}{2}$$

Es würde im Übrigen auf das gleiche Ergebnis hinauslaufen, wenn man den Wechselwirkungseffekt von A auf B untersucht (Kleppmann 2013, S. 102; Siebertz et al. 2017, S. 15–18).

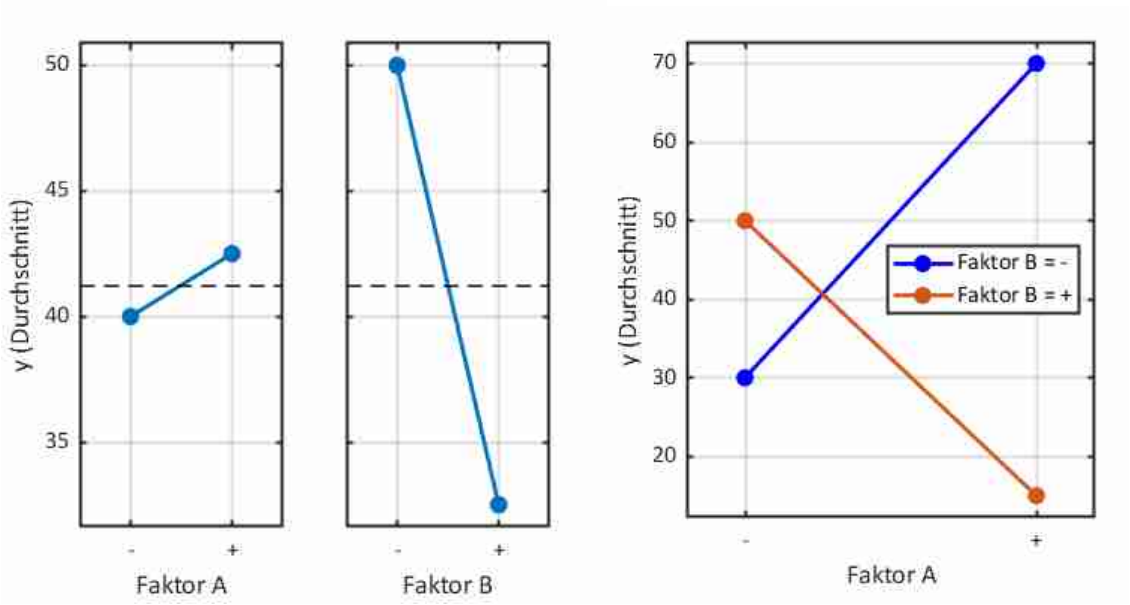


Abbildung 13: Effekt- und Wechselwirkungsdiagramme

Effekte und Wechselwirkungseffekte lassen sich nicht nur berechnen, sondern anschließend auch standardisiert in *Effekt-Diagrammen*[6] (Abbildung 13) darstellen (Kleppmann 2013, S. 103–104; Siebertz et al. 2017, S. 13–14):

Auf der horizontalen Achse werden die einzelnen Faktoren und ihre Stufeneinstellungen abgetragen. Da die Faktoren i.d.R. in unterschiedlichen Einheiten vorliegen (Meter, Liter, Watt usw.), ist diese Achse dimensionslos.

Die vertikale Achse zeigt den Wert der jeweiligen Zielgröße an. Werden mehrere Zielgrößen betrachtet, müssen auch dementsprechend viele Effektdiagramme angefertigt werden.

[6] Engl. „Effects plot“

Nun können die Mittelwerte der einzelnen Faktorstufen ($\bar{y}_{A_+}$, $\bar{y}_{A_-}$ usw.) eingezeichnet werden. Diese werden dann, für jeden Faktor einzeln, mit einer Geraden verbunden. Die Steigung dieser Geraden zeigt den Effekt des Faktors an.

Zur optischen Kontrolle kann zusätzlich noch der Gesamtmittelwert der Zielgröße $\bar{y}$ als weitere Linie eingezeichnet werden. Bei zwei Faktorstufen gilt: $\bar{y} = \frac{\bar{y}_{A_+} + \bar{y}_{A_-}}{2} = \frac{\bar{y}_{B_+} + \bar{y}_{B_-}}{2}$. Dementsprechend muss die Line des Gesamtmittelwertes die Effektlinien genau in deren Mitte schneiden.

Wenn zusätzlich noch die Wechselwirkungseffekte dargestellt werden sollen, müssen mehrere Linien pro Faktor gezeichnet werden: Hierzu werden die bedingten Mittelwerte für die Faktorstufen ($(\bar{y}_{A_+}|B_+)$, $(\bar{y}_{A_-}|B_+)$ usw.) eingetragen und nach der gleichen Manier verbunden, wie bei den einfachen Effekten.

Die Effektdiagramme von Abbildung 13 (die beiden auf der linken Seite) zeigen, dass der Effekt des Faktors A wesentlich kleiner ist als der von Faktor B. Zudem beeinflusst A die Zielgröße eher positiv, wohingegen ein großes B negativ auf sie einwirkt. Aus dem Wechselwirkungsdiagramm (rechte Seite) wird klar, dass die Einstellung von B vorgibt, in welche Richtung der Effekt von A verstärkt wird.

Teilfaktorierte Versuchspläne

Die *teilfaktorierten Versuchspläne,* oder auch *Screening-Versuchspläne*, sind die eigentliche Stärke der statischen Versuchsplanung. Ab einer gewissen Anzahl von Faktoren und Stufen sind Vollfaktorpläne, auf Grund der hohen Anzahl von Versuchen, nicht mehr handhabbar. Teilfaktorierte Versuchspläne können hier mit relativ geringem Informationsverlust die Anzahl der benötigten Versuche drastisch verringern (z.T. >90%). *Irreguläre Felder nach Plackett-Burman* sowie die *regulären Felder nach dem Yates-Standard* sind die bekanntesten Vertreter. Allerdings spielen sie in dieser Arbeit keine Rolle und wurden deshalb nur der Vollständigkeit halber erwähnt (Kleppmann 2013, S. 128; Siebertz et al. 2017, 28-30).

2.2.2 Gleichverteilte Versuchspläne

Mitunter sind faktorielle Versuchspläne jedoch nicht ausreichend. Insbesondere bei Computer-Experimenten, für die keine Vorabinformationen über die Struktur des Lösungsraums vorhanden sind, sollten die Punkte des Testfeldes möglichst gleichverteilt im gesamten Lösungsraum liegen. Nur so kann gewährleistet werden, dass die Struktur des Lösungsraumes vollständig erfasst werden kann. Somit bilden *gleichverteilte Versuchspläne* die notwendige Datenbasis zur Erstellung guter Metamodelle (Ebert et al. 2015; Santner et al. 2003; Siebertz et al. 2017, S. 198).

Sobol'-Sequenz

Die *Sobol'-Sequenz* wurde im Jahr 1967 von dem russischen Mathematiker *Ilya Meyerovich Sobol'* entwickelt und dient eigentlich zur numerischen Berechnung von hochdimensionalen Integralen (Sobol' 1967).

Dabei gehört sie zu den sogenannten LP_τ-Sequenzen und den *quasi-Monte Carlo Methoden* (QMC). Diese Art von Sequenzen versuchen, eine Menge von Punkten in einem s-dimensionalen Einheitswürfel $I^s = [0; 1]^s$ zu erstellen, welche möglichst gleichverteilt sind. Als Maß für die Gleichverteilung der Punkte kann die *Diskrepanz* herangezogen werden: Je kleiner diese ist, desto gleichverteilter sind die Punkte im Raum. Dabei weist die Sobol'-Sequenz das bisher angenommene Mindestmaß der Diskrepanz bei $N \to \infty$ auf $(\log^s N)$. Dieses Maß erfüllen auch anderen „Low-discrepancy"-Sequenzen (z.B. die Halton- und Faure-Sequenz), allerdings ist deren Berechnung wesentlich aufwendiger. Außerdem konvergiert die Sobol'-Sequenz schon bei kleinen N zu einer sehr guten Gleichverteilung der Punkte. Siehe dazu Abbildung 14 (Bratley und Fox 1988; Siebertz et al. 2017, S. 204; Sobol' 1998, 1967).

Die Sobol'-Sequenz wurde weiterhin unter Berücksichtigung der folgenden drei wünschenswerten Eigenschaften einer solchen Sequenz entwickelt (Sobol' 1967):

1. Beste mögliche Gleichverteilung, wenn $N \to \infty$
2. Gute Verteilung bei verhältnismäßig kleinen Anfangsmengen
3. Algorithmus mit sehr kurzen Rechenzeiten

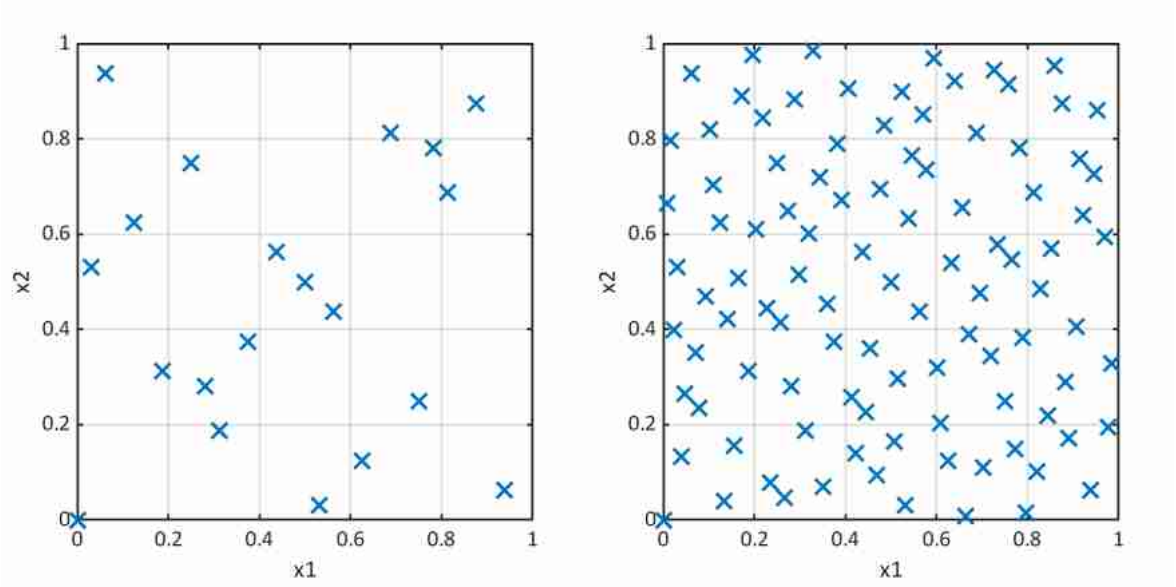


Abbildung 14: Zweidimensionale Sobol'-Sequenz mit N=20 und N=100

Im Folgenden sind die notwendigen Schritte zur Erstellung einer Sobol'-Sequenz, nach Bratley und Fox 1988; Joe und Kuo 2008; Sobol' et al. 2011, beschrieben:

Die Berechnung der Sobol'-Sequenz basiert auf den unabhängigen Eigenschaften von *primitiven Polynomen*. Primitive Polynome P zeichnen dich dadurch aus, dass sie irreduzibel sind, d.h. nicht weiter faktorisiert werden können, also nicht als Produkt zweier Polynome mit niedrigerer Ordnung geschrieben werden können[7]. Bei $P = x^3 + x + 1$ ist dies bspw. der Fall. Des Weiteren werden die Dimensionen einer Sobol'-Sequenz als einzelne Vektoren mit N Einträgen berechnet. Es wird demnach zunächst nur die Dimension j betrachtet:

Die allgemeine Darstellung eines primitiven Polynoms P mit dem Grad s_j lautet

$$P \equiv x^{s_j} + a_{1,j}x^{s_j-1} + a_{2,j}x^{s_j-2} + \cdots + a_{s_j-1,j}x + 1$$

wobei die Koeffizienten $a_{1,j}, a_{2,j}, \ldots, a_{s_j-1,j}$ nur einen Wert von 0 oder 1 annehmen können. Dabei ist die Wahl des Grades und der Koeffizienten beliebig, vorausgesetzt, dass P dadurch primitiv ist. Hat man sich für ein P entschieden, kann nun die 1. Stufe der sogenannten *direction numbers* $m_{k,j}$ berechnet werden:

$$m_{k,j} = 2a_{1,j}m_{k-1,j} \oplus 2^2 a_{1,j}m_{k-1,j} \oplus \ldots \oplus 2^{s_j-1}a_{s_j-1,j}m_{k-s_j+1,j} \oplus 2^{s_j}m_{k-s_j,j} \oplus m_{k-s_j,j}$$

[7] Als Beispiel, bei dem dies möglich ist: 1. binomischen Formel mit $x^2 + 2xy + y^2 = (x + y)(x + y)$

Der Operant $\oplus$ beschreibt den bitweisen XOR-Operator von zwei binären Zahlen. Die Zahlen müssen demnach von Basis 10 in Basis 2 und anschließend wieder zurück transformiert werden. Je nachdem, wie hoch der Grad von P, durch die Wahl von s_j, angesetzt wurde, müssen Startwerte für $m_{1,j}, m_{2,j}, \dots, m_{s_j,j}$ festgelegt werden. Diese können unter den folgenden Bedingungen frei gewählt werden: Sie sind ungerade, $1 \leq k \leq s_j$ und $m_{k,j} < 2^k$.

Hat man nun ein $m_{k,j}$ berechnet, fließt dieses wieder in die Gleichung ein. Für die Berechnung von $m_{k+1,j}$ werden die Werte in der Gleichung praktisch um eine Stelle nach rechts geschoben und $m_{k,j}$ vorne eingesetzt.
Im Anschluss daran kann die zweite Stufe der direction numbers $v_{k,j}$ durch

$$v_{k,j} = \frac{m_{k,j}}{2^k}$$

berechnet werden. Nun können auch die eigentlichen Punkte $x_{i,j}$ des Vektors in der Dimension j berechnet werden. Dies geschieht durch:

$$x_{i,j} = i_1 v_{1,j} \oplus i_2 v_{2,j} \oplus \dots$$

wobei $i_1, i_2, \dots, i_k$ die einzelnen Bits der binären Schreibweise des Laufzählers i, des Punktes $x_{i,j}$ sind: $i = (\dots i_3 i_2 i_1)_2$. Man sieht also, dass genauso viele direction numbers wie benötigte Bits für die binäre Darstellung von N berechnet werden müssen.

Um die ohnehin schon recht kurzen Rechenzeiten noch weiter zu drücken, kann statt der normalen binären Darstellung auch der sogenannte *Gray-Code*[8] verwendet werden. Dabei bleiben die Zahlen der Sobol'-Sequenz gleich, allerdings erscheinen sie in einer anderen Reihenfolge (Antonov und Saleev 1979).

Dadurch, dass es sich hierbei um eine Sequenz handelt, können bei Bedarf weitere Punkte einfach hinzugefügt werden, indem die Sequenz fortgesetzt wird.

Eine Beispielrechnung zur Sobol'-Sequenz ist in Joe und Kuo 2008 zu finden.

[8] Beim Gray-Code handelt es sich auch um eine binäre Darstellung, allerdings verändert sich hier immer nur ein Bit bei $i - 1$ nach i (Joe und Kuo 2008).

In dieser Arbeit wird zur Erzeugung von Sobol'-Sequenzen die Funktion `sobolset()` in MATLAB R2018a verwendet[9].

Latin Hypercube Design

Ein weitverbreitetes Verfahren zur Erstellung von gleichverteilten Versuchsplänen ist das sogenannte *Latin Hypercube Design* (LHD). Bei einem Latin Hypercube ist die Projektion eines jeden Punktes im Lösungsraum auf die Achsen in allen Dimensionen einmalig. Dieser Zustand wird dann als *Non-collapsing*[10] bezeichnet. Abbildung 15 zeigt den Unterschied zwischen einer *Collpasing* und *Non-collapsing* Anordnung der 9 Punkte im Raum (Ebert et al. 2015).

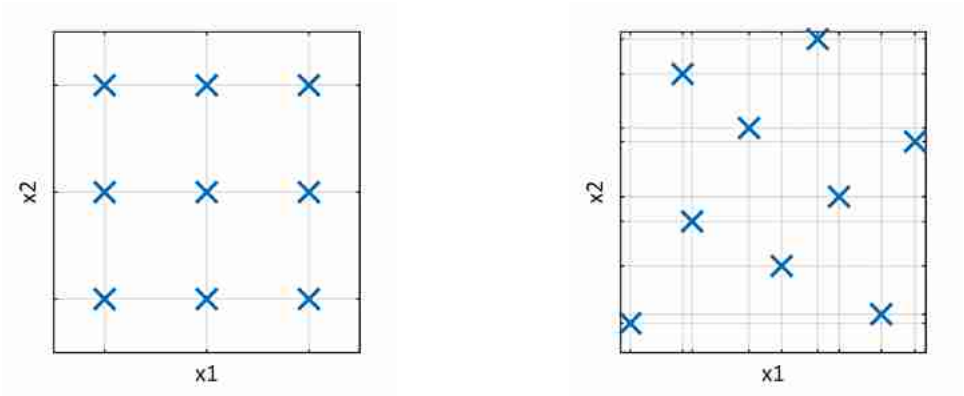


Abbildung 15: Collapsing und Non-collapsing mit N=9

Eine Non-collapsing Anordnung der Punkte garantiert jedoch nicht deren Gleichverteilung im Lösungsraum. Daher muss der Latin Hypercube mit Hilfe einer Verlustfunktion optimiert werden. Diese kann auf Grundlage zweier Kriterien berechnet werden: *Minimax* (mM) oder *Maximin* (Mm). Beim Minimax wird die Distanz eines jeden Punktes zu seinem nächsten Punkt minimiert, wohingegen beim Maximin die Distanz zwischen zwei benachbarten Punkten maximiert werden soll. Das Ergebnis ist letztendlich das gleiche, allerdings benötigt Maximin wesentlich weniger Rechenleistung als Minimax. Dennoch ist die Optimierung des LHD im Vergleich zur Erstellung eines nicht optimierten LHD enorm rechenaufwendig, da die Anzahl der möglichen Testfelder mit n_F Faktoren und n_S Samples ${n_S}^{n_F}$ entspricht (Ebert et al. 2015; Johnson et al. 1990; Siebertz et al. 2017, S. 207).

[9]Eine ausführliche Dokumentation zu dieser Funktion ist online unter https://www.mathworks.com/help/stats/sobolset-class.html zu finden.
[10] Zu Deutsch: „nicht-kollabierend“

Abbildung 16 zeigt beispielhaft ein Maximin optimiertes LHD mit 20 und 100 Samples.

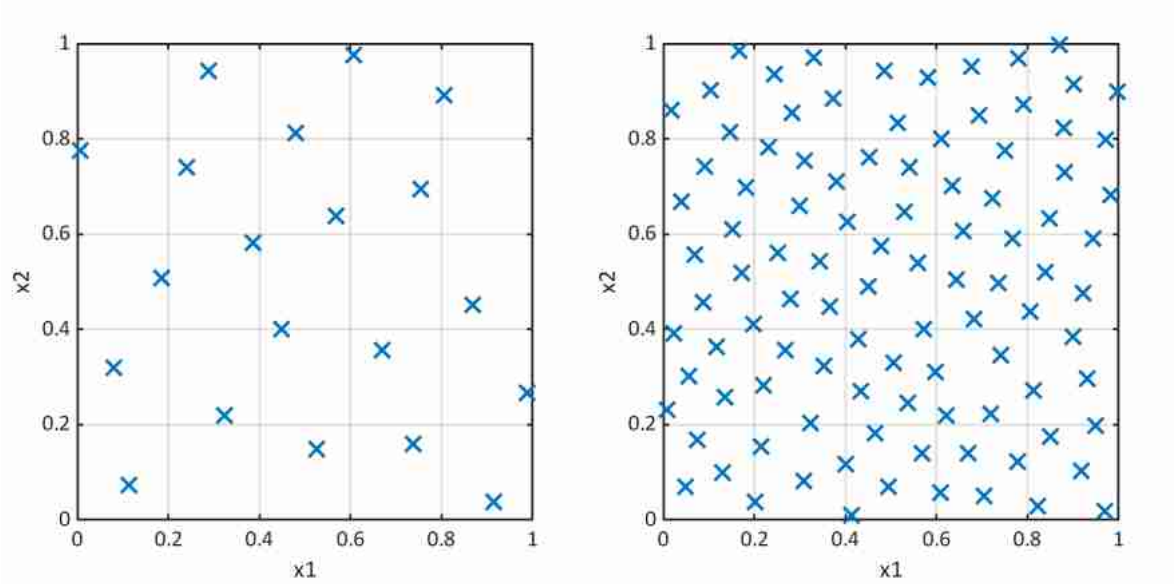


Abbildung 16: Zweidimensionales LHD (Maximin) mit N=20 und N=100

2.3 Metamodelle

2.3.1 Machine-Learning-Grundlagen und Notation

Machine-Learning-Algorithmen haben alle etwas gemeinsam: Sie bestehen aus einem Input und einem Output. Bei dem Input handelt es sich um eine Menge an Variablen (Messwerte oder Voreinstellungen), welche den Output eines Systems direkt beeinflussen. Diese Variablen werden als *Prädiktoren* oder *unabhängige Variablen* bezeichnet. Im Englischen findet man meistens die Bezeichnungen *feature, predictor* oder *independent variable*. Der Output kann aus einer oder mehreren Variablen bestehen und wird als *Zielgröße* oder *abhängige Variable* deklariert. Die englischen Äquivalente dazu sind die *outcome*, *response, target* oder *dependent variable* (Friedman und Meulman 2003; Hastie et al. 2017, S. 9).

Das Ziel von Machine-Learning-Algorithmen ist es, bei gegebenen (neuen) Prädiktoren-Werten die Zielgröße(n) möglichst genau zu schätzen. Demnach werden Machine-Learning-Algorithmen nach der Form der Zielgröße in zwei Gruppen aufgeteilt: Klassifikations- und Regressions-Probleme.

Ist die Zielgröße qualitativ, kann sie also nur einen diskreten Wert aus einer finiten Menge (an Klassen) annehmen, handelt es sich um ein *Klassifikations-Problem*. Beispielsweise

kann bei der Erkennung, ob ein Patient, bei gegebenen Vitaldaten, unter einer bestimmten Krankheit leidet oder nicht, die Zielgröße nur einen Wert aus $\mathcal{G} = \{\text{gesund}, \text{krank}\}$ annehmen. Bei *Regressions-Problemen* hingegen nimmt die Zielgröße einen quantitativen (kontinuierlichen) Wert an. Ein Beispiel hierfür wäre der Restwert eines Autos in Euro bei gegebenem Kilometerstand, Baujahr, Unfallfreiheit usw.: $\hat{y} = 9.315{,}49€$ (Hastie et al. 2017, S. 9–11).

Neben der Zielgröße können die einzelnen Prädiktoren auch qualitative oder quantitative Formen annehmen. In Anbetracht des o.g. Beispiels für den Restwert eines Autos würde es sich bei dem Kilometerstand um einen quantitativen Prädiktoren handeln. Im Vergleich dazu wäre die Unfallfreiheit des Autos nur mit *ja* oder *nein* zu beantworten und dementsprechend ein qualitativer Prädiktor. Die Art der Durchmischung von qualitativen und quantitativen Prädiktoren bestimmt in großem Maße die Wahl des passenden Algorithmus (Hastie et al. 2017, S. 10).

Auf Grund der Beschaffenheit der hier vorliegenden Problemstellung werden im weiteren Verlauf ausschließlich Regressions-Probleme behandelt.

Die Prädiktoren werden mit dem Vektor $\boldsymbol{x}$ bezeichnet (Vektoren werden durch fette Kleinbuchstaben dargestellt), wobei einzelne Prädiktoren in $\boldsymbol{x}$ mit dem Index $j \in \{1, \dots, p\}$ gekennzeichnet werden: x_j.

Die Zielgröße wird mit einem y bezeichnet. Eine geschätzte Zielgröße aus dem Prädiktoren-Vektor $\boldsymbol{x}$ wird darüber hinaus mit einem Dach versehen: $\hat{y}$ (engl. *yhat*).

Der Algorithmus, welcher eine geschätzte Zielgröße $\hat{y}$ liefern soll, kann als Schätzfunktion in Abhängigkeit von einem Prädiktoren-Vektor $\boldsymbol{x}$ geschrieben werden: $\hat{y} = \hat{f}(\boldsymbol{x})$. Diese Funktion wird auch als *Modell* bezeichnet.

Zum Anlernen des Algorithmus sind bereits bekannte Kombinationen aus Prädiktoren und Zielgrößen $(\boldsymbol{x}, y)$ notwendig[11]. Die sogenannten *Trainingsdaten* $\{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$ mit $i \in$

[11] Hier kann auch zwischen *überwachtem* und *unüberwachtem* Lernen unterschieden werden. Im Gegensatz zum überwachten Lernen werden beim unüberwachtem Lernen werden keine Zielgrößen benötigt. Der Algorithmus dient „nur“ dazu Zusammenhänge in den (Prädiktor-)Daten aufzunehmen.

$\{1, \dots, N\}$ Einträgen, wobei $\boldsymbol{x_i} = \left(x_{i,1}, x_{i,2}, \dots, x_{i,p}\right)$ die Beobachtungen im jeweiligen Eintrag i darstellt. Einzelne Skalare (konkrete Ausprägungen) werden mit dem Kleinbuchstaben x_{ij} bezeichnet. Angelehnt an das Auto-Restwert-Beispiel würde $x_{3,1}$ für den ersten Prädiktor (Kilometerstand) mit der dritten Beobachtung (bspw. 50.000 km) stehen.

Jede $(\boldsymbol{x}, y)$-Kombination kann als Hyperebene in einem $(p+1)$-dimensionalen Raum mit $\boldsymbol{x} \in \mathbb{R}^p$ und $y \in \mathbb{R}^1$, betrachtet werden. Ziel ist es das Modell mit Hilfe der Trainingsdaten möglichst nahe an diese Hyperebene heranzuführen, um in Zukunft bei einem neuen $\boldsymbol{x}$ eine verlässlich geschätzte Zielgröße $\hat{y}$ ausgeben zu können. Wie dies genau geschieht, hängt vom jeweiligen Algorithmus ab.

Der Begriff Machine-Learning ist hier eigentlich irreführend. Der Algorithmus an sich lernt nur dadurch, dass er seine anfangs schlechte Prognose durch iteratives Anpassen immer weiter verbessert. Die meisten Softwarelösungen für Machine-Learning-Algorithmen verfügen über Konfigurationsmöglichkeiten, die sogenannten *Hyperparameter*. Diese Parameter bestimmen in hohem Maße, wie der Algorithmus aus den Daten lernt. Dabei existiert eine große Anzahl unterschiedlicher Machine-Learning-Algorithmen wie z.B. *Neuronale Netze*, *Random Forest*, *Support Vector Machines* oder auch *Kernel Smoother* (Friedman und Meulman 2003; Goldstein et al. 2014; Hastie et al. 2017, S. 10–11; Welling et al. 2016).

2.3.1.1 Trainings-, Validierungs- und Testdaten

Ist man in der glücklichen Situation und hat viele Datensätze zur Verfügung, sollte man diese rein zufällig in drei Teile aufteilen: Die *Trainings*-, *Validierungs*- und *Testdaten*.

Die Trainingsdaten werden, wie der Name schon sagt, dazu verwendet, die jeweiligen Modelle zu trainieren. Die Validierungsdaten werden im Anschluss zur Evaluation der Modelle in Bezug auf deren Vorhersagegenauigkeit verwendet. Auf dieser Basis sollte dann auch das beste Modell zur späteren Verwendung ausgewählt werden (Modellauswahl). Die Testdaten sind im Idealfall wirklich nur ganz am Ende der Modellauswahl zur finalen Bewertung des ausgewählten, besten Modells einmalig anzuwenden (Modellbewertung). Dadurch, dass die Testdaten für das Modell komplett neu sind, kann man über sie den

echten Fehler in der Vorhersagegenauigkeit des Modells ermitteln. Würde man jedoch die Testdaten, ähnlich wie die Validierungsdaten, mehrmals benutzen, um ein Modell auszuwählen, welches den kleinsten Fehlerwert auf die Testdaten hat, könnte der Testdatensatz den realen Fehler des ausgewählten Modells doch deutlich unterschätzen (Hastie et al. 2017, S. 222).

Eine generelle Regel dazu, in welchem Verhältnis die Daten aufgeteilt werden müssen, gibt es nicht und ist auch immer auf die jeweilige Anwendung bezogen. Allerdings empfiehlt Hastie et al. 2017 als grobe Ausgangswerte für eine Aufteilung: 50% Trainingsdaten sowie jeweils 25% Validierungs- und Testdaten.

2.3.1.2 Bias-Variance Tradeoff

Bei der Auswahl bzw. beim Training von Modellen sollte der sogenannte *Bias-Variance Tradeoff*[12] beachtet werden. Dieser spielt insbesondere bei Daten, welche mit Rauschen behaftet sind, eine wichtige Rolle.

Angenommen, es existiert ein Datensatz, aus dem man immer wieder unterschiedliche Teildatensätze herausnehmen würde, um ein Modell mit einer gewissen Komplexität zu trainieren, dann beschreibt der *Bias,* wie groß die (durchschnittliche) Abweichung des Modelloutputs gegenüber den echten Datenpunkten ist, also wie stark die Schätzungen der Modelle generell von den echten Werten abweichen. Die *Variance* hingegen betrachtet nur die Streuung der Schätzungen über diese Modelle für einen fixierten Datenpunkt – dementsprechend, wie sehr sich die Schätzung der Modelle für diesen einen Punkt über die einzelnen Modelle hinweg ändert (Fortmann-Roe 2012; LeCun et al. 1998, S. 12).

$$Err(\boldsymbol{x}) = Bias^2 + Variance + \sigma^2$$

Der Gesamtfehler $Err(\boldsymbol{x})$ eines Modells setzt sich dann aus dem quadrierten Bias, der Variance sowie einem nichtreduzierbaren Fehler (σ^2) zusammen. Abbildung 17 zeigt die Verläufe des Bias, der Variance sowie des daraus resultierenden Gesamtfehlers des Modells in Abhängigkeit der Modellkomplexität. Es zeigt sich, dass der Bias mit

[12] Wird im Deutschen als „*Verzerrung-Varianz-Dilemma*“ bezeichnet. Diese Bezeichnung ist in der Fachwelt jedoch recht unüblich, deshalb wird hier weiterhin der englische Begriff verwendet.

zunehmender Komplexität abnimmt, wohingegen die Variance ansteigt (Hastie et al. 2017, S. 37–38).

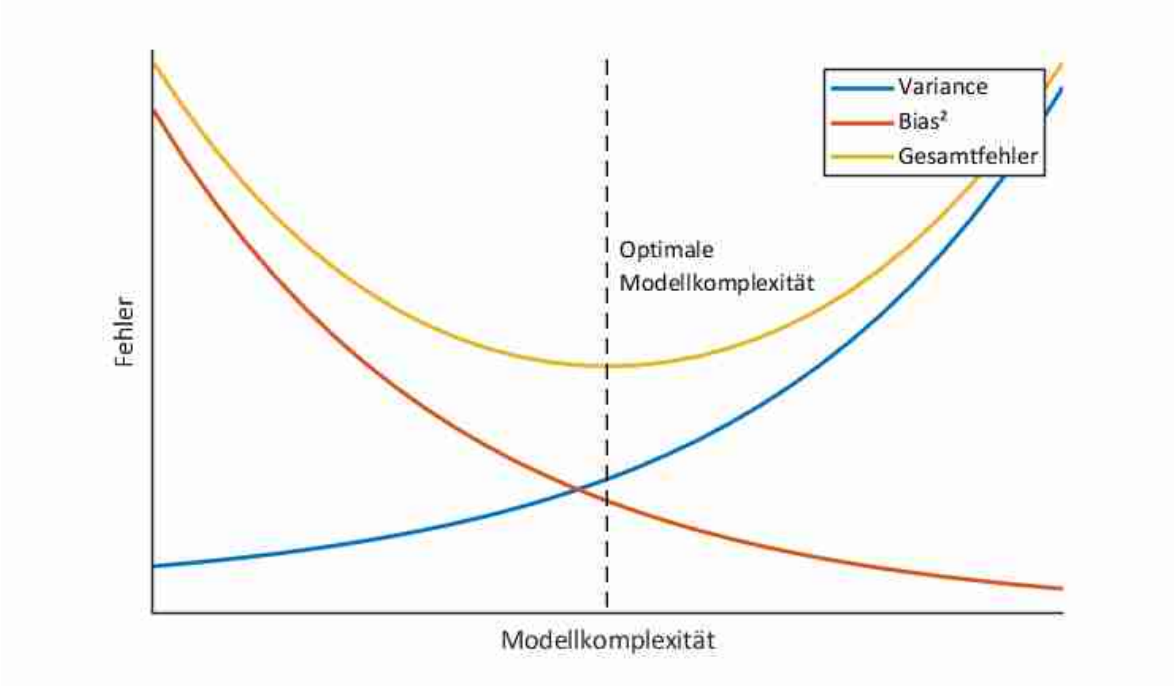


Abbildung 17: Bias-Variance Tradeoff

Man könnte eigentlich davon ausgehen, dass je komplexer ein Modell ist, desto besser kann es eine tieferliegende Funktion in den Daten aufnehmen und darstellen. Dies ist jedoch auf Grund des Rauschens in den Daten nicht der Fall. Abbildung 18 zeigt die durch drei Modelle (mit unterschiedlicher Komplexität) geschätzte Funktion aus einer Reihe von Datenpunkten (Briscoe und Feldman 2011; LeCun et al. 1998, S. 12):

Das Modell mit der niedrigen Komplexität (gelber Verlauf) wird nur sehr rudimentär von den Datenpunkten beeinflusst. Dadurch erkennt das Modell die tieferliegende Funktion in den Daten auch nur sehr schlecht. Folglich reagiert es auch nicht sonderlich sensibel auf den Input neuer Daten, insbesondere nicht auf deren (anderes) Rauschen. Dementsprechend ist der Bias hoch und die Variance niedrig. Dieser Zustand wird als *Underfitting* bezeichnet.

Der blaue Verlauf hingegen weist eine viel zu hohe Anpassung an die Daten auf, das sogenannte *Overfitting*. Durch die hohe Komplexität des Modells nimmt es nicht nur den grundlegenden Trend in den Daten auf, sondern wertet das Rauschen ebenfalls als eine

spezielle Eigenschaft davon (niedriger Bias). Würde man nun dem Modell neue, unbekannte Datenpunkte geben, aus denen es Schätzungen hervorbringen soll, würde das andersartige Rauschen dieser Daten fehlinterpretiert werden und die Schätzung wäre fehlerbehaftet (hohe Variance).

Der orangene Verlauf zeigt den Idealfall: das Modell hat eine vergleichsweise mittlere Komplexität und erkennt den Trend in den Daten, jedoch bleibt das Modell allgemein genug, um das Rauschen nicht als Teil des Problems zu sehen.

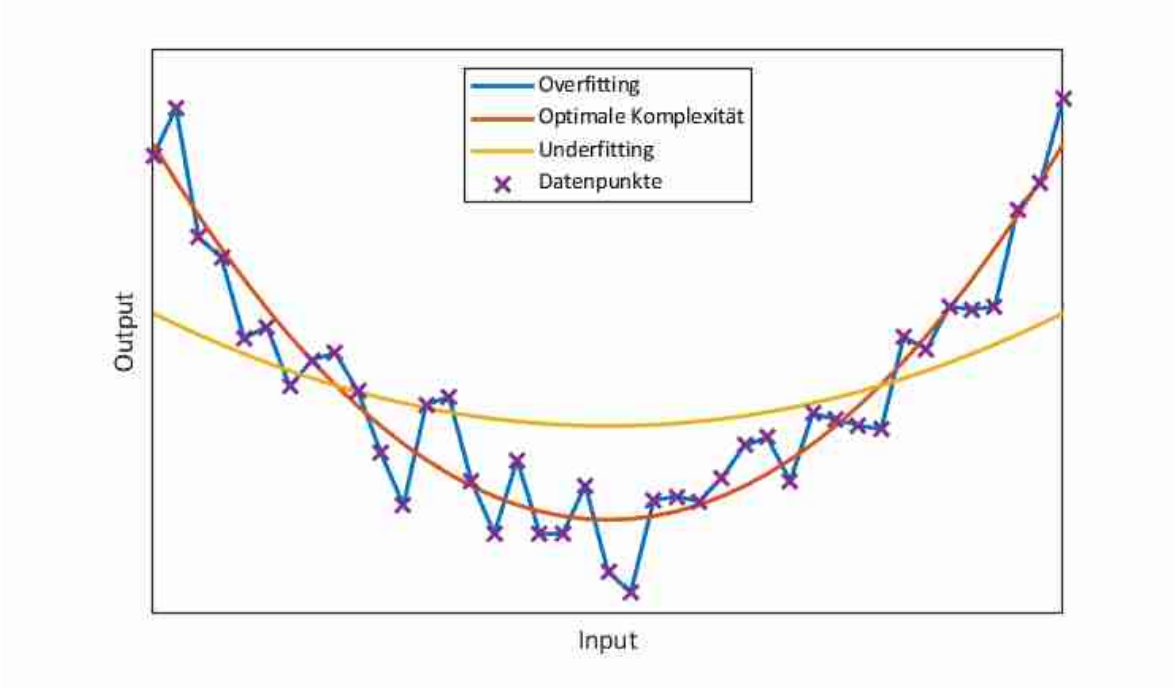


Abbildung 18: Over- und Underfitting

2.3.2 Multi-layer Perceptron

Der Aufbau eines *Künstlichen Neuronalen Netzwerkes*[13] *(KNN)* ist an die Architektur des menschlichen Gehirns angelehnt. Es besteht aus verschiedenen Schichten (engl. Layer), wobei jede Schicht wiederum eine Vielzahl von kleinen Elementen, die auch *Neuronen* genannt werden, enthält. Jedes dieser Neuronen sammelt an seinem Input Informationen, verarbeitet diese und gibt sie als Antwort an seinem Output wieder aus.

[13] Engl. „Artificial Neural Network“ (ANN)

Aus der Gesamtheit an KNN können auf Grund der Anordnung und Verbindungen der einzelnen Elemente verschiedene Arten von KNN herausgelöst werden. Die in der Praxis wohl am meisten verwendete KNN-Art ist das *Multi-layer Perceptron*[14] (MLP). Dies liegt vor allen daran, dass es sehr universell einsatzbar ist, da es sich für eine Vielzahl unterschiedlicher Problemstellungen eignet (Regression, Klassifikation, Optimierung usw.) (Walde 2005, S. 9–10).

In dieser Arbeit wird als Softwarelösung für ein MLP der *MLPRegressor* aus dem Machine-Learning-Paket *scikit-learn* in Python verwendet[15].

2.3.2.1 Funktionsweise

Der schematische Aufbau eines MLP ist in Abbildung 19 zu sehen. Wie schon erwähnt, besteht es aus mehreren Schichten, wobei sich jede Schicht aus einer Vielzahl von Neuronen zusammensetzt. Bei den Schichten wird zwischen drei Arten unterschieden:

Die erste Schicht ist die *Input-Layer.* Sie ist der Eingang des Netzwerkes und nimmt die Informationen des Input-Vektors $\boldsymbol{x}$ auf. Außerdem ist sie die einzige passive Schicht, d.h. sie reicht die Informationen nur weiter und verarbeitet sie nicht. Des Weiteren ist die Zahl der Neuronen in dieser Ebene auf die Anzahl der Prädiktoren begrenzt.

Auf die Input-Layer folgt eine beliebige Anzahl von *Hidden-Layer*[16], wobei die Anzahl der Neuronen pro Schicht frei wählbar ist. Die Anzahl der Neuronen pro Hidden-Layer bestimmt dabei in hohem Maße die Komplexität des MLPs und damit auch, wie gut es ein Problem erfassen kann und dementsprechend genaue Vorhersagen treffen kann. Dabei ist die optimale Kombination aus Hidden-Layer- und Neuronen-Anzahl für jedes Problem eine andere.

Die letzte Ebene des MLPs ist die *Output-Layer*. Sie präsentiert den Schätzwert des MLP $\hat{y}$. Die Output-Layer kann dabei aus nur einem oder mehreren Neuronen bestehen.

[14] Wird im Deutschen als „Mehrschichtiges Perzeptron“ bezeichnet. Diese Bezeichnung ist in der Fachwelt jedoch recht unüblich, deshalb wird hier weiterhin der englische Begriff verwendet.
[15] Eine ausführliche Dokumentation dazu ist online unter http://scikit-learn.org/stable/ sowie in Pedregosa et al. 2011 zu finden.
[16] Wird im Deutschen als „Verdeckte Schicht“ bezeichnet.

Beim MLP sind Neuronen einer Schicht mit sämtlichen Neuronen der vorigen und nachstehenden Schicht verbunden. Bei den Verbindungen zwischen den Neuronen handelt es sich um Gewichtungen, die den Output des vorigen Neurons verändert weitergeben. Je nach Größe des Eingangs (Reizes) werden die Neuronen aktiviert (Überschreitung eines Bias-Wertes) und reichen ihre Informationen an die nächste Ebene weiter (Hammerstrom 1993; Goh 1995; Nielsen 2015).

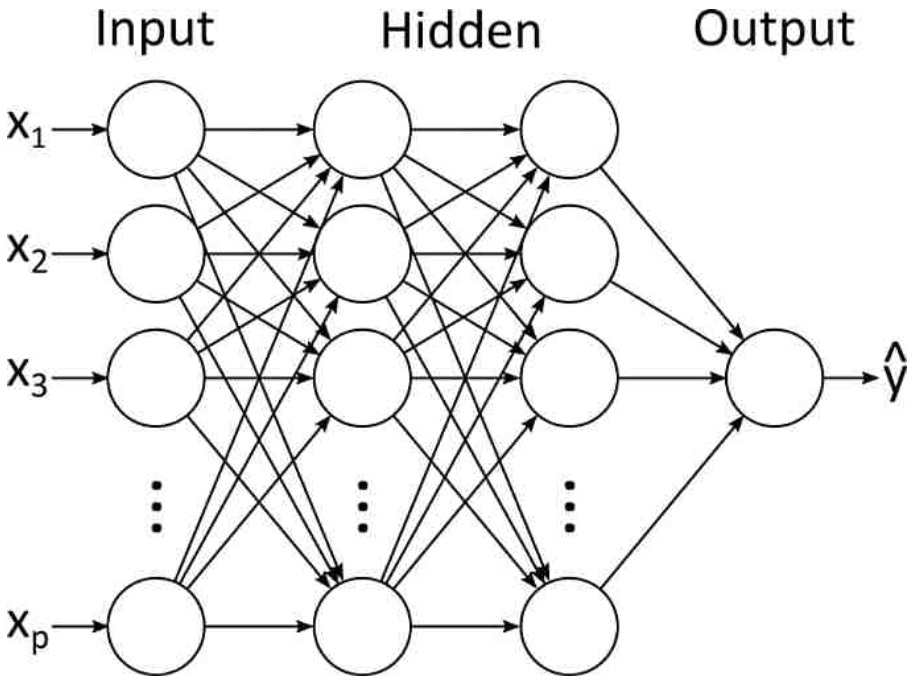


Abbildung 19: Schematische Darstellung eines Multi-layer Perceptron

Abbildung 20 zeigt die Funktionsweise eines einzelnen Neurons m in der Hidden- bzw. Output-Layer im Detail (Hagan et al. 2014, S. 2–7; Hastie et al. 2017, S. 392–394):

Zu Beginn stehen die *Aktivierungen* (engl. Activations) der vorigen Schicht, welche mit $a_1, a_2, a_3, \dots, a_k$ gekennzeichnet sind. Es handelt sich dabei um die Output-Werte der Neuronen in der Schicht vor dem Neuron m bzw. um den Input-Vektor $\boldsymbol{x}$, wenn sich das Neuron in der ersten Hidden-Layer befindet (mit $a_1 = x_1$, $a_2 = x_2$ usw.).

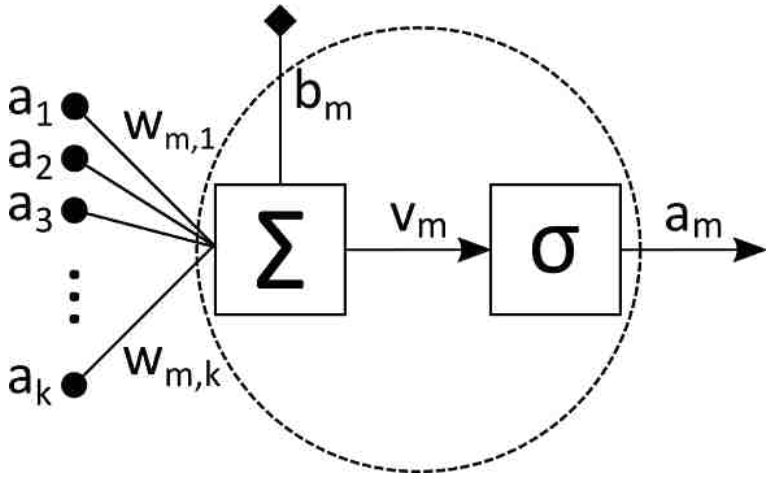


Abbildung 20: Aufbau eines Neurons

Diese Aktivierungen werden daraufhin mit der entsprechenden Gewichtung $w_{m,1}, w_{m,2}, w_{m,3}, \dots, w_{m,k}$ multipliziert, die Produkte aufsummiert und mit dem Bias-Wert b_m verrechnet:

$$v_m = \sum_{k=1}^{K} w_{m,k}\, a_k - b_m$$

Anmerkung: Der Bias-Wert b_m kann auch als zusätzliche Gewichtung $w_{m,0}$ mit einer Aktivierung $a_0 = 1$ geschrieben werden: $b_m = 1 \cdot w_{m,0}$.

Zur Bestimmung der Aktivierung a_m des Neurons m wird das Ergebnis aus der vorigen Gleichung v_m in eine sogenannte *Aktivierungsfunktion* σ (engl. Transfer Function) gegeben:

$$a_m = \sigma(v_m)$$

Die gleiche Rechenoperation wird nun für alle anderen Neuronen der Schicht durchgeführt, um anschließend genau nach dem gleichen Muster die nächste Schicht berechnen zu können. Wäre das Neuron m das einzige Neuron in der Output-Layer, dann würde $a_m = \hat{y}$ gelten. Da hier die Informationen von vorne nach hinten durchgegeben werden, spricht man auch von einem *Feed-forward* KNN.

Im Folgenden sind vier Aktivierungsfunktionen mit deren Namen, Funktion σ, 1. Ableitung der Funktion σ' (welche beim Training des MLPs wichtig wird) sowie deren Verlauf aufgelistet (Buscema 1998, S. 246–248; Glorot et al. 2011, S. 318; Pedregosa et al. 2011; Walde 2005, S. 13):

1. *Identische Abbildung* mit:

$$\sigma(v) = v$$

$$\sigma'(v) = 1$$

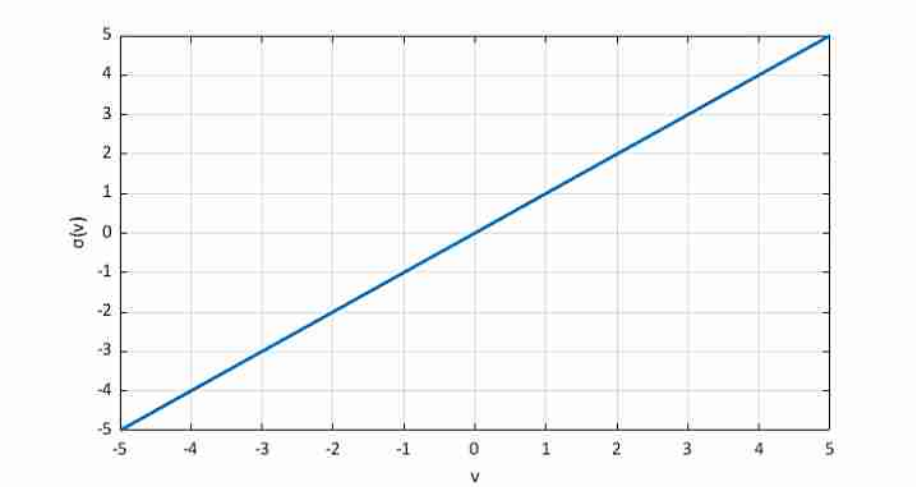


Abbildung 21: Identische Abbildung

2. *Sigmoid bzw. Logistische Funktion* mit:

$$\sigma(v) = \frac{1}{1 + e^{-v}}$$

$$\sigma'(v) = \sigma(v)\big(1 - \sigma(v)\big)$$

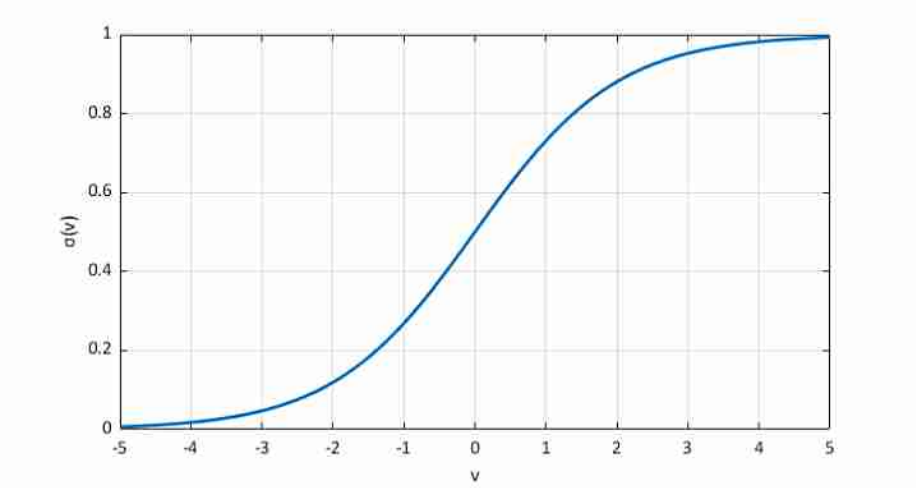


Abbildung 22: Sigmoid bzw. Logistische Funktion

3. *Tangens Hyperbolicus Funktion* mit:

$$\sigma(v) = \tanh(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}}$$

$$\sigma'(v) = 1 - \sigma(v)^2$$

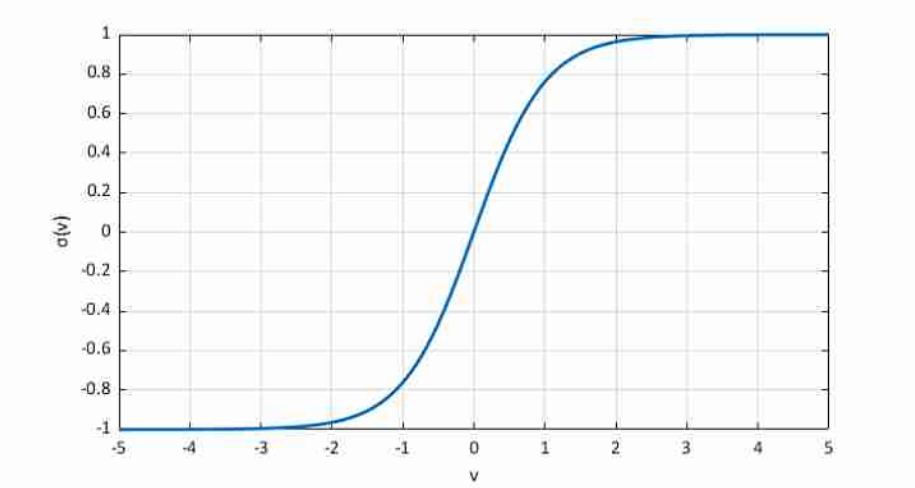


Abbildung 23: Tangens Hyperbolicus

4. *Rectified Linear Units* (ReLU) mit:

$$\sigma(v) = \max(0; v) = \begin{cases} 0, & v < 0 \\ v, & v \geq 0 \end{cases}$$

$$\sigma'(v) = \begin{cases} 0, & v < 0 \\ 1, & v \geq 0 \end{cases}$$

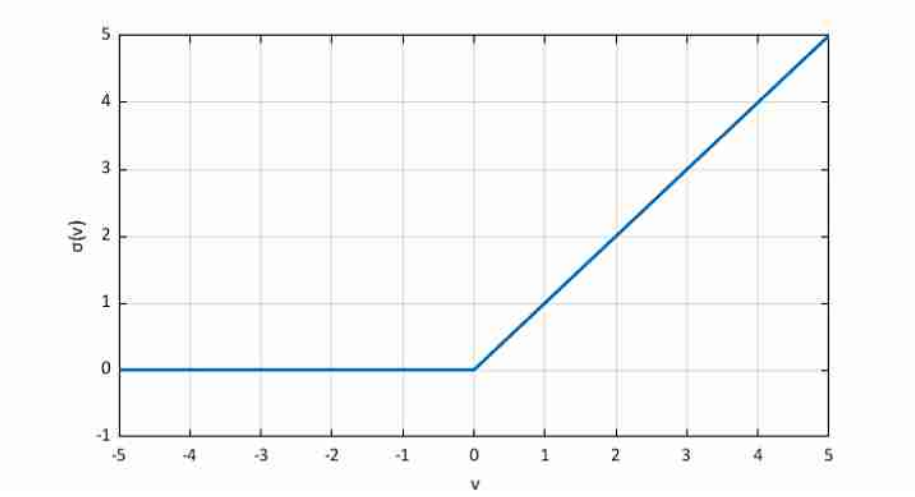


Abbildung 24: ReLU

Es existieren noch viele weitere Aktivierungsfunktionen wie bspw. der *Arkustangens*, *Sinus*, *Leaky ReLU* oder auch der *Einheitssprung* (Buscema 1998, S. 248; Hagan et al. 2014, S. 2–4; Maas et al. 2013). Diese stehen jedoch bei der Verwendung des *MLPRegressors* nicht zur Auswahl und werden daher auch hier nicht weiter behandelt.

2.3.2.2 Training

2.3.2.2.1 Vorbereitung der Daten

Grundsätzlich wird bei der Verwendung von MLPs (als auch bei normalen ANN) literaturübergreifend dringend dazu geraten, die Trainings-, Test- und Validierungsdaten vor dem Training des Modells gemeinsam zu transformieren. Durch eine Transformation der Daten konvergiert das MLP zum einen wesentlich schneller und zum anderen kann dadurch auch dessen Genauigkeit verbessert werden. Da die Gewichtungen zu Beginn des Trainings in der Regel im Intervall $[-0{,}7; +0{,}7]$ liegen, kann man, gerade bei der Verwendung der Sigmoid-Funktion ($a = [0; +1]$) bzw. Tanh-Funktion ($a = [-1; +1]$) als Aktivierungsfunktion, erkennen, dass betragsmäßig hohe Eingangsgrößen von den Nodes der ersten Hidden-Layer nicht vernünftig weitergegeben werden können, da sie stets voll aktiviert sind. Es bräuchte etliche Iterationsschritte, um die Gewichtungen zwischen der Input-Layer und der ersten Hidden-Layer soweit zu reduzieren, damit diese Nodes überhaupt einen Unterschied zwischen unterschiedlichen Werten der Eingänge feststellen können. Daher haben sich bis dato drei Transformationsmethoden etabliert, welche die Eingangsgrößen (und Ausgangsgrößen) in handhabbaren Spannweiten halten (Hastie et al. 2017, S. 398–400; LeCun et al. 1998, S. 17–18; Mendelsohn 2018; Pedregosa et al. 2011; Walde 2005, S. 12):

1. Lineare Transformation auf das Intervall $[0; +1]$ durch:

$$x_{i,j}^{transf\,[0;1]} = \frac{x_{i,j} - \min(\boldsymbol{x}_j)}{\max(\boldsymbol{x}_j) - \min(\boldsymbol{x}_j)}$$

2. Lineare Transformation auf das Intervall $[-1; +1]$ durch:

$$x_{i,j}^{transf\,[-1;1]} = 2 \cdot \frac{x_{i,j} - \min(\boldsymbol{x}_j)}{\max(\boldsymbol{x}_j) - \min(\boldsymbol{x}_j)} - 1$$

3. z-Standardisierung (engl. *z-Scores*), mit einer Standardabweichung von 1 und einem Durchschnitt von 0, durch:

$$x_{i,j}^{standard} = \frac{x_{ij} - \bar{x}_j}{\hat{\sigma}_j}$$

dabei gilt $\bar{x}_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$ und $\hat{\sigma}_j = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(x_{i,j} - \bar{x}_j\right)^2}$

Achtung: Es gilt hier unbedingt zu beachten, dass die gewählte Transformationsmethode mit den jeweiligen Parametern auch auf die Input-Vektoren im späteren Betrieb (wenn das Modell Schätzungen für neue, unbekannte Eingänge berechnen soll) angewendet werden muss.

2.3.2.2.2 Gradient Descent und Backpropagation

Nachdem die vorliegenden Daten transformiert bzw. standardisiert sind, kann das eigentliche Training des MLP beginnen.

Da die tieferliegende Funktion aus den Daten normalerweise nicht deterministisch berechnet werden kann, bleibt nur die Option: Durch geschicktes Justieren an den Gewichtungen und Bias-Werten des MLPs eine Funktion zu erzeugen, welche der echten Funktion möglichst nahekommt.

Dazu wird jedoch zunächst ein Maß benötigt, welches quantifiziert, wie nahe das MLP an der echten Funktion liegt bzw. wie weit es noch davon entfernt ist. In den meisten Fällen handelt es sich hierbei um die *Quadratische Fehlerfunktion*[17] J *(LeCun et al. 1998, S. 10; Pedregosa et al. 2011; Walde 2005, S. 18)*:

$$J_i = \frac{1}{2}\left(y_i - f(\boldsymbol{x}_i, \theta)\right)^2$$

Die Fehlerfunktion misst für ein Sample i aus den Trainingsdaten $\{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$ mit $i \in \{1, \dots, N\}$, die Diskrepanz zwischen der Schätzung des MLPs $f(\boldsymbol{x}_i, \theta)$ und der realen Zielgröße y_i, wobei θ alle aktuellen Gewichtungen und Bias-Werte des MLPs enthält. Erweitert man diese Funktion über die gesamten Trainingsdaten, ergibt sich daraus:

[17] Engl. „Squared-loss Costfunction“

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} J_i$$

Das Ziel sollte nun sein, ein θ zu finden, welches die Fehlerfunktion über die gesamten Trainingsdaten minimiert:

$$\theta^{(Opt)} = \arg\min_{\theta}[J(\theta)]$$

Die Fehlerfunktion ist allerdings in der Regel hoch nichtlinear und weist neben dem globalen Optimum auch eine Vielzahl von lokalen Optima auf. Des Weiteren kann ein Optimum nicht deterministisch berechnen werden. Man muss sich daher von einem Startpunkt aus schrittweise einem der (lokalen) Optima nähern. Dies geschieht in der Regel durch den sogenannten *Gradient Descent*[18].

Der Gradient einer Funktion zeigt an, in welche Richtung die Funktion am stärksten ansteigt (größte positive Steigung). Dementsprechend zeigt der negative Gradient an, in welcher Richtung der stärkste Abfall der jeweiligen Funktion liegt. Da das Ziel in der Minimierung der Fehlerfunktion $J(\theta)$ liegt, ist nur der negative Gradient der Funktion von Interesse: $-\nabla_\theta J(\theta)$ bzw. $-\frac{\partial J}{\partial \theta}$. Der negative Gradient zeigt an, welche Gewichtungen und Bias-Werte in welche Richtung und wie stark verändert werden müssen, damit die Fehlerfunktion sich am schnellsten einem lokalen Optimum nähert. Dabei wird der negative Gradient iterativ solange verfolgt, bis $\nabla J = 0$ ist, sich die Fehlerfunktion nicht mehr um einen gewissen Wert verbessert oder eine definierte Iterationsgrenze erreicht wurde (Pedregosa et al. 2011; Walde 2005, S. 19).

[18] Zu Deutsch: „Gradientenverfahren“

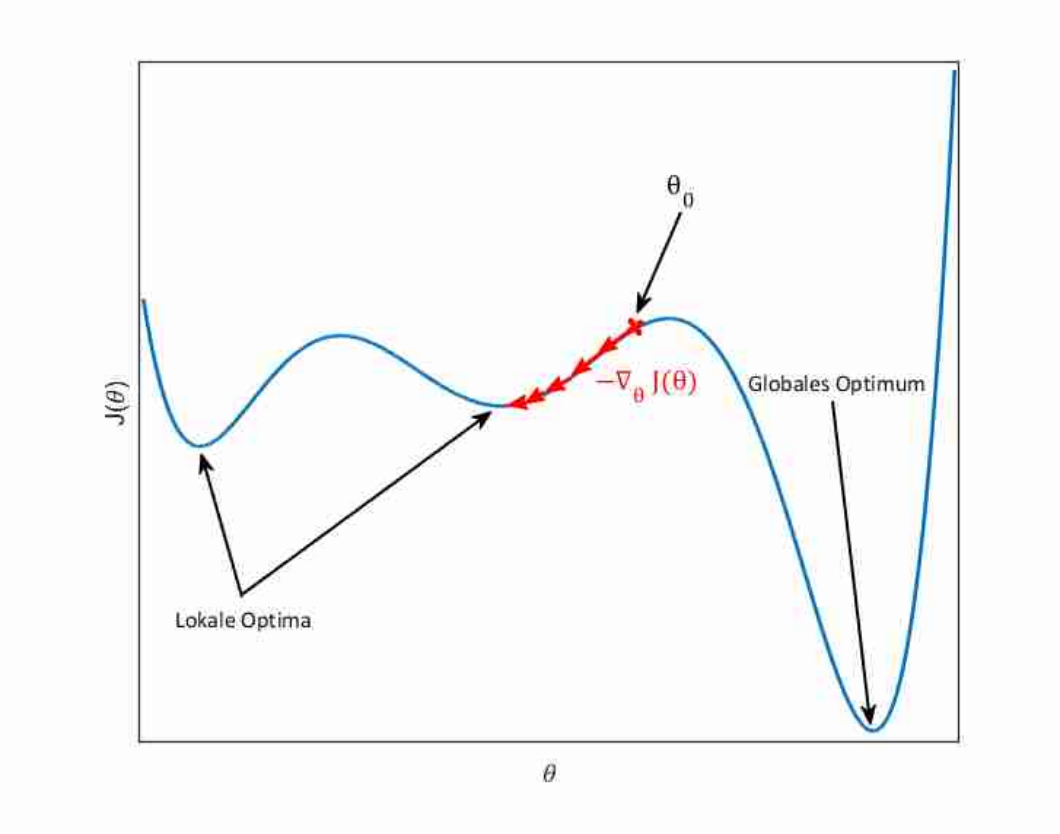


Abbildung 25: Gradient Descent

Abbildung 25 zeigt den Gradient Descent für ein MLP. Aus Anschauungszwecken mit lediglich einer Gewichtung θ. Die roten Pfeile kennzeichnen die negativen Gradienten bzw. die *Schrittweiten* aus jeder Iteration.

Wie man sieht, ist die Wahl des Startpunktes θ_0 dafür verantwortlich, welches der Optima überhaupt erreicht werden kann. Daher ist es meistens der Fall, dass man beim Training von MLPs (KNN im Allgemeinen) das Training von mehreren, unterschiedlichen Startpunkten aus durchführen muss, um dann das Modell mit dem kleinsten (lokalen) Optimum auswählen zu können.

Weiterhin kann man erkennen, dass die Schrittweiten proportional zu der Steigung (Gradient) der Funktion sind. Das bedeutet, dass je näher man einem (lokalen) Optimum kommt, die Anpassungen an θ immer kleiner werden. Den Verlauf der Anpassungen an θ kann man als Gleichung in Abhängigkeit des jeweiligen Iterationsschrittes[19] t schreiben:

[19] Engl. „cycle“ oder „epoch“

$$\theta(t) = \theta(t-1) - \eta \cdot \nabla_\theta J(\theta)$$

Dabei bezeichnet η die *Lernrate* (engl. Learning rate) des MLP. Sie nimmt einen Wert zwischen 0 und 1 an und beschreibt, wie stark die Werte in θ pro Iteration angepasst werden. Oftmals liegt die Lernrate im Bereich von $0{,}001$ bis $0{,}1$ (Goh 1995; LeCun et al. 1998, S. 12–14).

Die Implementierung des Gradient Descent zum Trainieren eines MLP (bzw. KNN) wird als *Backpropagation* (BP) bezeichnet. Die BP besteht aus zwei Phasen: In der ersten Phase wird ein Input-Vektor $\boldsymbol{x}_i$ aus den Trainingsdaten vorwärts durch das MLP geschickt, um einen Schätzwert $\hat{y}_i$ zu errechnen. Dieser Schätzwert wird dann mit der realen Zielgröße y_i verglichen, was ein Fehlersignal ergibt. In der zweiten Phase wird dann dieses Signal rückwärts durch das MLP „propagiert“ und die entsprechenden Gewichtungen und Bias-Werte abgeändert. Hierher hat die BP auch ihren Namen (LeCun et al. 1998, S. 13; Rumelhart et al. 1986a, S. 327; Walde 2005, S. 24).

Das Zurückpropagieren mit den entsprechenden Änderungen an den Gewichtungen und Bias-Werten wird von Rumelhart et al. als *Delta-Regel*[20] beschrieben. Die Delta-Regel besteht im Grunde genommen nur aus drei Gleichungen.

Kurz zur Notation: Da die BP von hinten (Output-Layer) nach vorne (Input-Layer) geht, stellt im Folgenden das Neuron k ein beliebiges Neuron in der nächsten Schicht (in Richtung Input-Layer) des MLPs dar, wohingegen das Neuron m ein beliebiges Neuron der vorigen Schicht (in Richtung Output-Layer) angehört. Siehe auch Abbildung 20: Aufbau eines Neurons.

Die erste Gleichung beschreibt die Gewichtungen[21] $w_{m,k}$ zwischen den Neuronen m und k im nächsten Iterationsschritt $(t+1)$, als Summe aus den aktuellen Gewichtungen und der Änderung $\Delta w_{m,k}$:

$$w_{m,k}(t+1) = w_{m,k}(t) + \Delta w_{m,k}$$

[20] Engl. „Delta Rule“
[21] Man beachte hier für die Bias-Werte: $b_m = 1 \cdot w_{0,m}$

Die Änderung an der Gewichtung $\Delta_i w_{m,k}$ für ein einziges Sample i sollte dann proportional zu dem Produkt aus dem erzeugten Fehlersignal $\delta_{i,m}$, der Aktivierung $a_{i,k}$ des Neurons in der nächsten Schicht und der Lernrate η sein:

$$\Delta_i w_{m,k} = \eta \cdot \delta_{i,m} \cdot a_{i,k}$$

Bei der Berechnung des Fehlersignals $\delta_{i,m}$ wird zwischen zwei Fällen unterschieden:

$$\delta_{i,m} = \begin{cases} \sigma'_m(v_{i,m})(y_{i,m} - a_{i,m}) & \text{falls Neuron in Output-Layer} \\ \sigma'_m(v_{i,m}) \sum_{l=1}^{L} \delta_l w_{l,i} & \text{falls Neuron in Hidden-Layer} \end{cases}$$

Wobei $l \in \{1, \dots, L\}$ die Verbindung zu allen Neuronen in der vorigen Schicht bezeichnet. Diese Berechnungen werden schließlich solange wiederholt, bis man an der Input-Layer angekommen ist (Rumelhart et al. 1986a, S. 326–327).

Eine detaillierte Herleitung der BP ist in Rumelhart et al. 1986a und Rumelhart et al. 1986b beschrieben.

2.3.2.2.3 Varianten des Gradient Descents

Bei der Verwendung des Gradient Descent gibt es unterschiedliche Definitionen davon, wann ein Iterationsschritt abgeschlossen ist und Änderungen an den Gewichtungen und Bias-Werten vorgenommen werden. Hierbei muss zwischen der Geschwindigkeit und der Genauigkeit des Verfahrens abgewogen werden.

Batch Gradient Descent

Beim *Batch Gradient Descent* durchläuft bei jeder Iteration der komplette Trainingsdatensatz das Netz bevor die Gewichtungen angepasst werden. Der Vorteil liegt hier darin, dass diese Variante garantiert bei konvexen J in ein globales Optimum und bei nicht-konvexen Fehleroberflächen in ein lokales Optimum konvergiert. Allerdings zu dem Preis, dass es sehr langsam ist und die gesamten Trainingsdaten in den Arbeitsspeicher geladen werden müssen, was bei größeren Datensätzen problematisch werden kann. Die allgemeine Gleichung für eine Iteration lautet (Ruder 2016):

$$\theta(t+1) = \theta(t) - \eta \cdot \nabla_\theta J(\theta)$$

Stochastic Gradient Descent

Der *Stochastic Gradient Descent* (SGD)[22] aktualisiert die Gewichtungen nach jedem einzelnen Sample der Trainingsdaten und ist dadurch wesentlich schneller als der Batch Gradient Descent:

$$\theta(t+1) = \theta(t) - \eta \cdot \nabla_\theta J(\theta, \boldsymbol{x}_i, y_i)$$

Als Folge dessen weisen die einzelnen Updates der Gewichtungen jedoch eine hohe Varianz auf, was dazu führt, dass die Fehlerfunktion J sehr große Sprünge macht. Zum einen kann es dadurch passieren, dass neue (und eventuell bessere) Minima gefunden werden. Zum anderen wird somit die Konvergenz behindert, da die Schritte über das Optimum hinausschießen. Abhilfe kann hier eine graduelle Verringerung der Lernrate schaffen (Ruder 2016).

Mini-batch Gradient Descent

Als Zwischenform der beiden bisher behandelten Varianten gilt der sogenannte *Mini-batch Gradient Descent*. Hierbei werden die Aktualisierungen jeweils nach Mini-batches[23] mit n Einträgen vorgenommen:

$$\theta(t+1) = \theta(t) - \eta \cdot \nabla_\theta J(\theta, \boldsymbol{x}_{i:i+n}, y_{i:i+n})$$

Diese Variante findet in der Regel eine sehr gute Balance zwischen Geschwindigkeit und Stabilität der Fehlerfunktion. Daher ist sie auch in den meisten heutigen MLP-Bibliotheken die erste Wahl. So auch beim MLPRegressor in Python. Typischerweise umfasst ein Mini-batch (je nach Anwendung) 50 bis 256 Einträge. Es sei zu erwähnen, dass auf Grund ihrer weiten Verbreitung das Mini-batch Verfahren oftmals begrifflich mit dem SGD gleichgesetzt wird (Pedregosa et al. 2011; Ruder 2016).

2.3.2.2.4 Optimierung des Gradient Descent

Bei heutigen Implementierungen des Gradient Descent wird in der Regel ein optimierter Algorithmus auf dessen Basis verwendet. Einer dieser Optimierungsansätze nutzt das sogenannte *Momentum*[24]. Dabei wird die Schrittweite der Änderungen an Gewichten und

[22] Zu Deutsch: „Stochastisches Gradientenverfahren“
[23] Kleine Datensätze aus den Trainingsdaten
[24] Zu Deutsch: „Schwung“ oder „Impuls“

Bias-Werten aus dem letzten Iterationsschritt v_{t-1} in die Berechnung der aktuellen Änderungen v_t miteinbezogen. Der *Momentum-Term* γ gibt an, wie stark der letzte Iterationsschritt berücksichtigt wird. Er liegt normalerweise im Bereich von 0,9 (Ruder 2016; Walde 2005, S. 28):

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla_\theta J(\theta)$$

$$\theta(t+1) = \theta(t) - v_t$$

Geht nun die Suche nach dem Optimum der Fehlerfunktion über mind. zwei Iterationen in die gleiche Richtung, wird die Schrittweite in diese Richtung vergrößert, geschieht Gegenteiliges, wird sie gedämpft. Dadurch kann die Suche wesentlich schneller konvergieren. Man kann sich hier auch einen Ball vorstellen, welcher in ein Tal hinabrollt. Je länger er rollt, desto schneller wird er. Hat er jedoch das Tal durchlaufen, wird er am gegenüberliegenden Berg zunächst ein Stück heraufrollen, umkehren und wieder zurückrollen. Dies wird so lange alternieren (oszillieren), bis der Ball die Talsohle erreicht hat und liegenbleibt. (Ruder 2016; Walde 2005, S. 28–29).

Der Algorithmus *AdaGrad* stellt wiederum eine Weiterentwicklung des Ansatzes mit Momentum dar. Siehe dazu Duchi et al. 2011 für Details.

Ein weiterer Optimierungsansatz ist die sogenannte *RProp* (Resilient Backpropagation) bzw. dessen Weiterentwicklung *RMSProp*. Dort werden die einzelnen Zahlenwerte des Gradienten nicht berücksichtigt, sondern nur deren Vorzeichen. Auch hier wird eine Art Momentum verwendet. Ändert sich das Vorzeichen eines Eintrages im Gradienten über mind. zwei Generationen nicht, wird die Schrittweite bzw. das *Update-value* Δ erhöht. Beim Vorzeichenwechsel eines Eintrages im Gradienten allerdings kann man davon ausgehen, dass der Algorithmus bei diesem Eintrag über das Optimum herausgeschossen ist. Daher wird in diesem Fall die Schrittweite verringert (Riedmiller 1994).

Eine detaillierte Beschreibung des RProp kann in Riedmiller 1994 bzw. des RMSProp in Tieleman und Hinton 2012 gefunden werden.

Abbildung 26 zeigt eine Visualisierung des normalen GD, des GD mit Momentum sowie der RProp. Die Länge der Pfeile stellt die Schrittweite pro Iterationsschritt dar. Man kann hier

gut erkennen bzw. eben nicht erkennen, wie klein die Schrittweiten der normalen GB auf Grund der geringen Steigung im Lösungsraum sind. Dementsprechend viele Iterationen werden auch zur Konvergenz benötigt. Die GD mit Momentum benötigt vergleichsweise rund 90% weniger Iterationen. Dafür schwingt sie, auf Grund des Momentums, mehrmals über den „idealen“ Pfad der normalen GD hinaus. Die RProp bewegt sich auf Grund der Vorzeichenwechsel im zweidimensionalen Raum recht zackig. Dafür findet sie das Optimum mit Abstand am schnellsten. Der Python-Code zur Anfertigung dieser Grafik stammt von Shevchuk 2015.

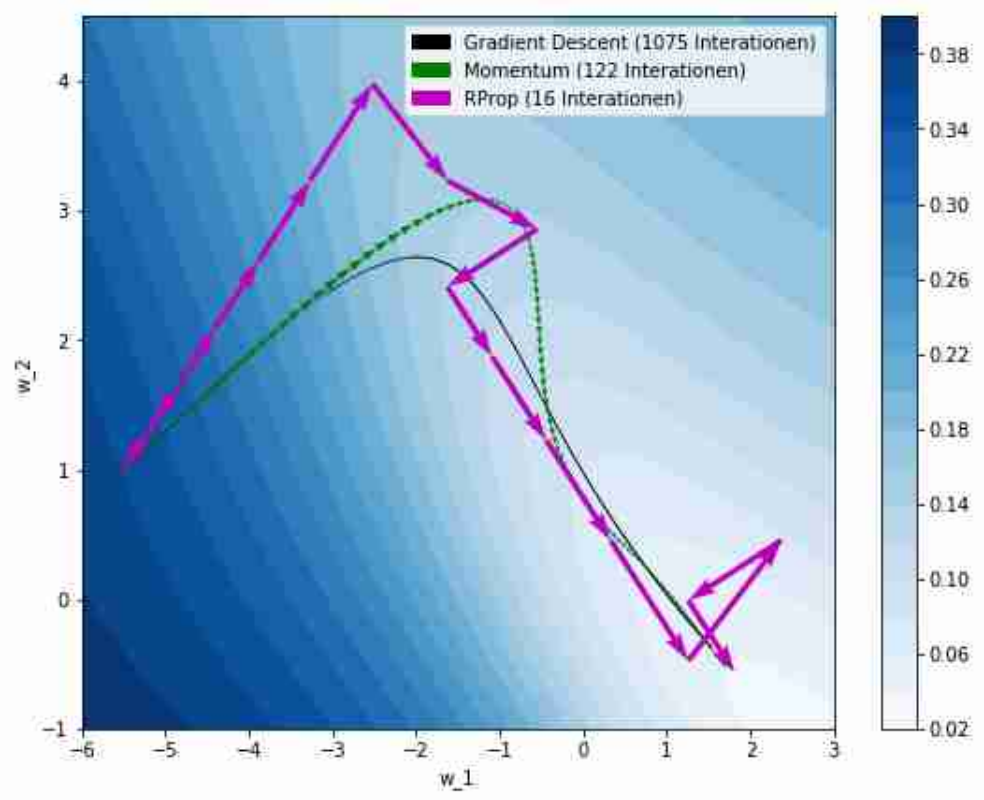


Abbildung 26: Visualisierung der GD-Varianten

Der MLPRegressor verwendet zum Training des MLPs den sogenannten *Adam* von Kingma und Ba. Dabei handelt es sich um eine verfeinerte Kombination aus RMSProp und AdaGrad. Auf eine detailliertere Betrachtung wird hier verzichtet, da dies den Umfang dieser Arbeit überschreiten würde. Er kann jedoch bei Bedarf in Kingma und Ba 2015 nachgeschlagen werden. Weitere Gradient Descent Optimierungen können darüberhinaus in Ruder 2016 nachgelesen werden.

2.3.2.2.5 Early Stopping

MLPs haben grundsätzlich das Problem, dass sie zum Overfitting neigen. Dabei ist nicht nur die offensichtliche Komplexität des MLPs in Form von der Anzahl der Hidden-Layer sowie die Neuronen pro Layer relevant, sondern auch, wie weit das MLP im Trainingsprozess vorangeschritten ist. In der Regel wird der Fehler des MLPs auf die Trainingsdaten mit jeder Iteration kleiner. Wenn man allerdings nach jeder Iteration das Modell zusätzlich mit ihm unbekannten Daten (Validierungsdaten) testet, erkennt man, dass der Fehler auf die Daten, welche nicht zum Training verwendet werden, zunächst zusammen mit den Trainingsdaten fällt und dann ab einem gewissen Punkt wieder ansteigt (siehe Abbildung 27). Dies lässt darauf schließen, dass das Overfitting des Modells anfängt einzusetzen (Geman et al. 1992; Prechelt 1998, S. 55).

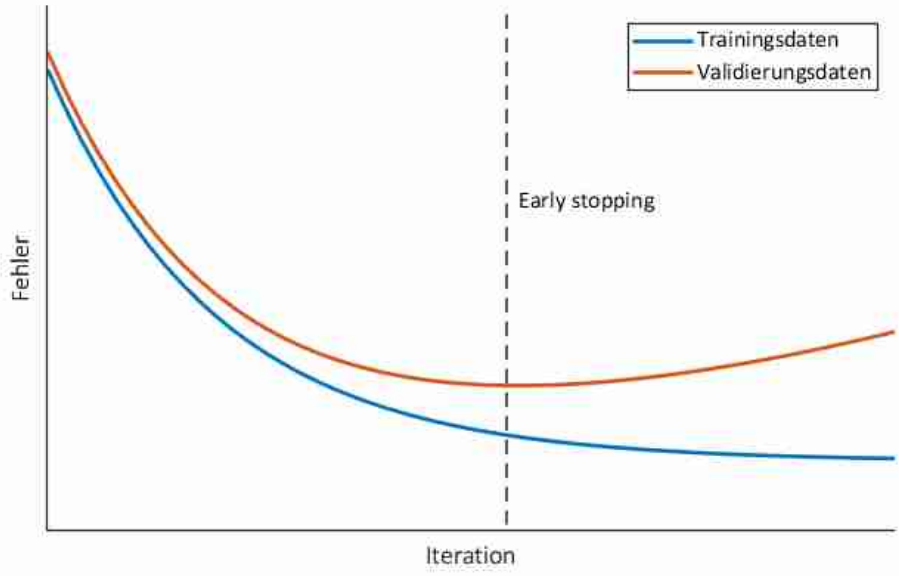


Abbildung 27: Early Stopping

Neben verschiedenen Regularisierungsansätzen hat sich das *Early Stopping (ES)* als eine sehr einfache und oftmals auch überlegene Strategie zur Vermeidung des Overfittings bei MLPs bewiesen.

Zur Implementierung des ES wird zunächst der Trainingsdatensatz in zwei Teile gesplittet: Die Daten, mit denen das Modell wirklich trainiert werden soll und ein zusätzlicher

Validierungsdatensatz[25], mit dessen Hilfe die Allgemeingültigkeit des Modells getestet werden soll. Während des Trainings wird nun in regelmäßigen Abständen (bspw. jede 5. Iteration) das Modell mit den Validierungsdaten getestet. Sobald der Fehler auf die Validierungsdaten beginnt wieder anzusteigen, wird das Training des MLPs gestoppt (Abbildung 27). Die letztendlichen Gewichtungen des MLPs sind dann diejenigen aus dem Iterationsschritt, an dem der Fehler auf die Validierungsdaten am niedrigsten war (Prechelt 1998, S. 55–56).

2.3.3 Modellauswahl

2.3.3.1 Maße für die Regressionsgüte

Für die Berechnung der Genauigkeit eines Regressionsmodells haben sich mehrere Maße etabliert. All diese Maße haben gemeinsam, dass sie anhand der Validierungs- bzw. Testdaten den Fehler des Modells durch einen Vergleich der durch das Modell geschätzten Ergebnisse $\hat{y}$ und der echten Ergebnisse y feststellen. Da hier sehr viele Datenpunkte auf einen einzelnen Zahlenwert zusammengepresst werden, beleuchten die Maße jeweils nur einen Teilaspekt des Problems. Für tiefergreifende und genauere Analysen müsste zusätzlich bspw. die Verteilung der Fehler betrachtet werden. Diese Maße sind jedoch schnell und einfach zu berechnen und liefern i.d.R. einen guten Eindruck von der Modellgüte (Chai und Draxler 2014).

R^2 *(R-squared)* - Coefficient of determination (Bestimmtheitsmaß)

Das Bestimmtheitsmaß bewegt sich nur im Intervall $[0,1]$ und ist somit dimensionslos. Bei einem Wert von $R^2 = 0$ schätzt das Modell die Zielgröße überhaupt nicht, wohingegen die Schätzungen bei $R^2 = 1$ exakt mit den realen Werten übereinstimmen. Es berechnet sich aus folgender Gleichung (Pedregosa et al. 2011; Ring et al. 2006):

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

[25] Es handelt sich hierbei nicht um die Validierungsdaten aus 2.3.1.1

Mean Absolute Error (MAE) (Mittlere absolute Abweichung)

Der MAE hat die Eigenschaft, dass er sämtliche Fehler gleich stark in die Bewertung einbezieht; dadurch ist er für gleichverteilte Fehlerverteilungen geeignet. Zum anderen wird er in der Einheit der Zielgröße angegeben. Wie bei allen folgenden Maßen sollte der Wert für ein gutes Modell möglichst niedrig sein (Chai und Draxler 2014).

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$$

Mean-Square Error (MSE) (Mittlere quadratische Abweichung)

Beim MSE werden auf Grund der Quadrierung hohe absolute Abweichungen stärker gewichtet als kleine. Das Problem hierbei ist, dass das Maß nicht sehr intuitiv ist, da hier sehr hohe Werte entstehen können. Auf der anderen Seite hat es den Vorteil, dass es keine absoluten Werte verwendet und auf Grund der Beschaffenheit sehr gut für Ableitungen und damit verbundene Berechnungen von bspw. Gradienten geeignet ist (Chai und Draxler 2014).

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Root-Mean-Square Error (RMSE) (Wurzel der mittleren quadratischen Abweichung)

Für den RMSE gilt bzgl. der Gewichtung der Abweichungen das Gleiche wie für den MSE. Durch das Ziehen der Quadratwurzel wird der Wert des MSE jedoch wieder „runterskaliert“, was ihn intuitiver macht. Für die Praxis empfehlen Chai und Draxler 2014 sehr hohe Ausreißer in der Fehlerverteilung von der Bewertung auszuschließen, insbesondere dann, wenn die Anzahl der Testsamples begrenzt ist. Des Weiteren ist die Aussagekraft des RMSE am besten, wenn die Fehlerverteilung normalverteilt ist und einen Mittelwert von 0 aufweist. Systematische Fehler des Modells in der Fehlerverteilung (bspw. Fehlerwerte fast alle positiv) sollten, wenn möglich, vor der Berechnung des RMSE herausgerechnet werden (Chai und Draxler 2014).

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2} = \sqrt{\mathrm{MSE}}$$

Normalized-Root-Mean-Square Error (NRMSE) (Normalisierter RMSE)

Der NRMSE ist wiederum diemensionslos, allerdings ist dessen Definition in der Literatur sehr uneinheitlich. Im Folgenden sind zwei mögliche Berechnungsarten dargestellt, wobei die Zweite in dieser Arbeit (der Vollständigkeit halber) verwendet wird (CIRP 2014):

$$\mathrm{NRMSE} = \frac{\mathrm{RMSE}}{\bar{y}} \text{ bzw. } \mathrm{NRMSE} = \frac{\mathrm{RMSE}}{\max(\boldsymbol{y}) - \min(\boldsymbol{y})}$$

2.3.3.2 Risidual-Plots

Risidual-Plots zeigen die Abweichungen zwischen den Vorhersagen des Metamodells $\hat{y}_i$ und den tatsächlichen Testergebnissen y_i grafisch an. Auf der x-Achse sind die realen Testergebnisse abgebtragen. Zusammen mit der dazugehörigen Schätzung des Modells auf der y-Achse ergeben sie einen gemeinsamen Punkt im Graphen (Abbildung 28). Im Idealfall liegen alle diese Punkte auf der Diagonalen in der Mitte. Liegt ein Punkt auf dieser Geraden, gilt für ihn $\hat{y}_i = y_i$ (Siebertz et al. 2017, S. 78–81).

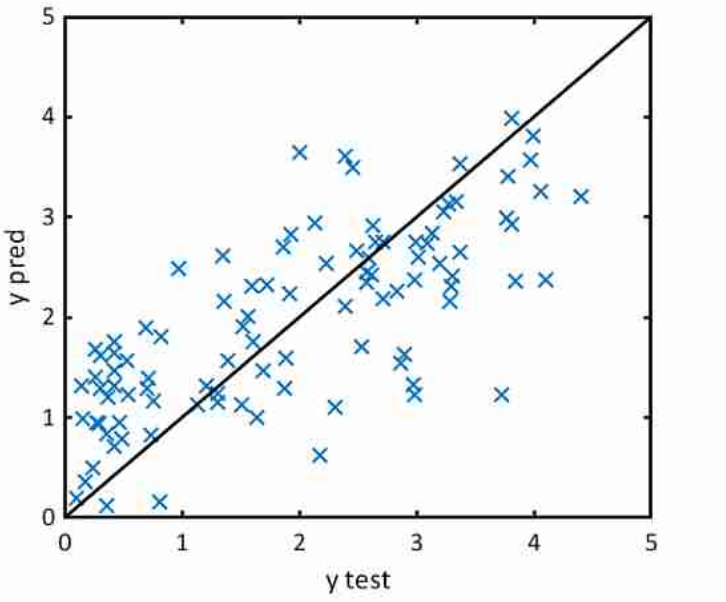


Abbildung 28: Risidual-Plot

2.3.3.3 Hyperparameter-Optimierung

Hyperparameter bieten für Machine-Learning-Algorithmen eine ganze Reihe von Einstellmöglichkeiten, wie das Modell aufgebaut und wie genau sein Training ablaufen soll. Dabei beeinflussen die gewählten Hyperparameter maßgeblich die Güte bzw. Genauigkeit

des Modells. Bei einem MLP gehören unter anderem die Anzahl der Hidden-Layer, Neuronen pro Layer sowie die Lernrate zu den einstellbaren Hyperparametern.

Das Ziel eines Algorithmus $\mathcal{A}$, mit den Hyperparametern λ und den Trainingsdaten $\boldsymbol{X}^{(Train)}$, ist es, eine Funktion bzw. ein Modell f zu finden, welche eine innere Fehlerfunktion J auf die Trainingsdaten $\boldsymbol{X}^{(Train)}$ möglichst minimiert. Dabei gilt $f = \mathcal{A}\left(\boldsymbol{X}^{(Train)}; \lambda\right)$. Die *Hyperparameter-Optimierung*[26] beschäftigt sich nun damit, diejenigen Hyperparameter $\lambda^{(Opt)}$ zu identifizieren, welche eine äußere, vorher festgelegte, Fehlerfunktion $\mathcal{L}$ (über die Validierungsdaten $\boldsymbol{X}^{(Val)}$) minimieren, um somit ein Modell zu finden, welches sich für das aktuelle Problem am besten eignet. Dabei löst der Algorithmus $\mathcal{A}$ für jedes λ das innere Optimierungsproblem (bspw. beim MLP durch den Gradient Descent). Bei der Fehlerfunktion $\mathcal{L}$ kann es sich bspw. um eines der in 2.3.3.1 beschriebenen Maße handeln. Formal kann diese Suche durch die folgende Gleichung beschrieben werden (Bergstra und Bengio 2012; Claesen et al. 2014; Claesen und Moor 2015):

$$\lambda^{(Opt)} = \arg\min_{\lambda} \left[\mathcal{L}\left(\boldsymbol{X}^{(Val)}; \mathcal{A}\left(\boldsymbol{X}^{(Train)}; \lambda\right)\right)\right]$$

Das Problem hierbei liegt darin, dass es bis dato keine effizienten Algorithmen gibt, welche Optimierungen auf Basis dieser Gleichung durchführen können. Zum anderen verbirgt sich eine stochastische Komponente in ihr, die insbesondere bei der Verwendung von Modellen, welche in sich selbst gewisse Zufälligkeiten tragen, zum Vorschein kommt. Dies kommt gerade bei MLPs zum Tragen, bei denen die Initialisierung der Startgewichtungen zufällig ist. Es gibt Ansätze, welche diese Einflüsse versuchen zu handhaben, allerdings führt dies in den meisten Fällen dazu, dass die Rechenzeiten unverhältnismäßig stark ansteigen (Bergstra und Bengio 2012; Claesen und Moor 2015).

Ein Ansatz, welcher vom Grundgedanken her der einfachste ist, liegt darin, alle möglichen Kombinationen von Hyperparametern in einem definierten Lösungsraum zu testen und anschließend das beste Modell auszuwählen. Diese Herangehensweise wird als *Grid*

[26] Engl. „Hyperparameter optimization" bzw. „Hyperparameter search"

Search[27] bezeichnet und ist praktisch identisch zu vollfaktoriellen Versuchsplänen. Bei der Variation über nur sehr wenige Hyperparameter ist dies problemlos machbar, jedoch bei zunehmender Anzahl Parameter setzt der *Fluch der Dimensionalität*[28] ein, da die Anzahl der benötigten Versuche exponentiell mit der Anzahl der Hyperparameter ansteigt. Ein weiteres Problem liegt darin, dass in den meisten Fällen nicht alle Hyperparameter gleich wichtig für die Modellgüte sind. Da jedoch über alle Kombinationen variiert wird, werden viele unnötige Versuche durchgeführt, welche keine nennenswerten neuen Erkenntnisse liefern und nur die Rechendauer in die Länge ziehen (Bellman 1961; Bergstra und Bengio 2012; Claesen et al. 2014).

Eine Möglichkeit, dem zu entgegnen, ist die Kombination aus Grid Search und einer manuellen Suche. Dies ist auch bislang der in der Praxis am weitesten verbreitete Ansatz zur Optimierung der Hyperparameter. Dazu wird durch die manuelle Suche der Lösungsraum zunächst eingeschränkt und anschließend in ihm eine Grid Search durchgeführt. Problematisch bleibt auch hier der Fluch der Dimensionalität und zusätzlich benötigt die manuelle Suche ein sehr hohes Maß an Expertenwissen, welches oftmals nicht verfügbar ist und auch die Reproduzierbarkeit der Optimierung erschwert (Bergstra und Bengio 2012; Claesen et al. 2014; Hinton 1998).

Bergstra und Bengio 2012 empfehlen daher die sogenannte *Random Search*[29]. Die Random Search nutzt den gleichen definierten Lösungsraum wie die Grid Search, allerdings werden aus ihm gleichverteilt zufällige Hyperparameter-Konfigurationen generiert und diese anschließend getestet. Dies macht es bei wenigen Hyperparametern unwesentlich schlechter als die Grid Search. Allerdings erweist sie sich als umso effizienter bei einer hohen Anzahl von Parametern. Dies liegt vornehmlich an dem erwähnten Fluch der Dimensionalität sowie der unterschiedlichen Relevanz der einzelnen Hyperparameter für ein Problem.

Die Random Search erweist sich, genauso wie die Grid Search, als sehr leicht zu implementieren und reproduzieren. Der Vorteil liegt hier darin, dass die Random Search

[27] Zu Deutsch: „Rastersuche“
[28] Engl. „Curse of Dimensionality“
[29] Zu Deutsch „Zufallssuche“

nicht adaptiv ist, d.h. jede Konfiguration, die generiert und getestet wird, ist komplett unabhängig von den getesteten Konfigurationen zuvor und danach. Dadurch kann die Versuchsreihe auf mehrere Rechner aufgeteilt werden, wobei einzelne Rechner ausfallen können, ohne, dass das komplette Experiment wiederholt werden müsste. Daher kann das Experiment auch zu jedem Zeitpunkt angehalten und bei Bedarf fortgesetzt werden (Bergstra und Bengio 2012).

Die Random Search bietet sich bei den Versuchen in dieser Arbeit an, da kein leistungsstarkes Rechenzentrum zur Verfügung steht. Stattdessen können die Evaluationen auf mehrere kleinere Rechner verteilt werden und parallel laufen. Da sie nicht adaptiv ist, bedarf es außerdem keiner Abstimmung der beteiligten Rechner untereinander.

Neben den angesprochenen Verfahren existieren noch weitere Ansätze zur Hyperparameter-Optimierung wie z.B. *Genetische Algorithmen*, *Particle Swarm Optimization*, *Coupled Simulated Annealing*, *Racing Algorithmen* oder auch *Reversible Learning* (Claesen und Moor 2015; Maclaurin et al. 2015).

2.4 Auswertungsmethoden

Oftmals ist es nicht ausreichend, nur zu wissen, ob und gegebenenfalls wie gut ein trainierter Algorithmus funktioniert. Um ein besseres Verständnis über ein System zu erlangen, muss man die Möglichkeit haben, in die Black-Box des Algorithmus hineinzuschauen, um dort die treibenden Faktoren ausfindig zu machen. Dazu wurden im Laufe der Zeit verschiedene Verfahren entwickelt. Zum einen existieren Methoden, die aufzeigen, welche Prädiktoren wichtig für die Berechnung einer Zielgröße sind. Zum anderen bieten verschiedene Methoden Informationen darüber, wie sich eine Zielgröße verhält, wenn man den Wert eines oder mehrerer Prädiktoren verstellt. Dabei sind einige dieser Methoden für verschiedene Machine-Learning-Algorithmen universal einsetzbar und andere wiederum nur für eine bestimmte Art verwendbar.

Detaillierte Rechenbeispiele und Anwendungen zu den einzelnen Auswertungsmethoden können in Dickel 2017 gefunden werden.

2.4.1 Relative Importance

In den meisten Fällen sind nicht alle Prädiktoren gleichwichtig für die Prognose eines Modells. Es handelt sich dabei in der Regel um einige wenige, welche sozusagen „den Ton angeben“. Andere Prädiktoren tragen hingegen praktisch nichts zum Ergebnis bei und können somit im Hinblick auf eine Optimierung bzgl. der betrachteten Zielgröße vernachlässigt werden. Um ein System besser zu verstehen, ist es deshalb von hoher Wichtigkeit, die treibenden und vernachlässigbaren Faktoren zu identifizieren und gegebenenfalls deren Wichtigkeit zu quantifizieren. Diese Aufgabe kann über die *Relative Importance (RI)* der Variablen erfüllt werden.

Zur Berechnung der RI von Prädiktoren für ein MLP stehen mehrere Möglichkeiten zur Auswahl. Olden et al. 2004 haben die unterschiedlichen Methoden auf ihre Performance und Verlässlichkeit hin untersucht und anschließend miteinander verglichen. Dabei hat sich herausgestellt, dass der sogenannte *Connection Weight Approach* (auch *Olden Algorithmus* genannt) die besten Ergebnisse liefert.

Der *Olden Algorithmus* wurde von Olden und Jackson entwickelt und bildet bei der Berechnung der Relative Importance für jeden Prädiktoren eine Summe über die Produkte der ein- und ausgehenden rohen Gewichtungen der Neuronen innerhalb der Hidden-Layers. Die Bias-Werte der einzelnen Neuronen werden dabei ignoriert. Je größer die betragsmäßige Summe eines Prädiktors ist, desto wichtiger ist er für den Wert der Zielgröße des MLP. Dabei sagt das Vorzeichen der Summe aus, in welche Richtung der Prädiktor die Zielgröße beeinflusst. Der Vektor $\boldsymbol{r}$ beinhaltet die RI der einzelnen Prädiktoren, L ist die Anzahl der Layer im MLP (inklusive Input- und Output-Layer) und die Matrix $\boldsymbol{W}$ umfasst alle Gewichtungen zwischen zwei benachbarten Schichten (Olden und Jackson 2002; Olden et al. 2004; Paliwal und Kumar 2011):

$$\boldsymbol{r} = \prod_{l=1}^{L-1} \boldsymbol{W}_{l,l+1}$$

2.4.2 Partial Dependence Plot

Die wohl bekannteste und am weitesten verbreitete Variante zur Visualisierung des geschätzten Ergebnisses eines Modells in Abhängigkeit von bestimmten Prädiktoren-

Werten ist der *Partial Dependence Plot (PDP)*. Er wurde von Friedman im Jahre 2001 erstmals vorgestellt.

Auf Grund der beschränkten menschlichen Wahrnehmung kann der PDP nur die Abhängigkeit des Modells auf einige wenige Prädiktoren in einer einzigen Grafik darstellen. Um bessere Einblicke in das Innenleben des Modells zu bekommen, müssten entweder entsprechend viele unterschiedliche PDPs erstellt werden oder man beschränkt sich auf die im Vorhinein ermittelten wichtigsten Prädiktoren des Modells.

Die allgemeine Berechnung gilt für alle sogenannten Black-Box-Algorithmen, bei denen das genau Modelle nicht ohne Weiteres betrachtet werden kann, wie z.B. MLP, Support Vector Machines, Nearest Neighbors oder auch die Radiale Basisfunktion. Grundsätzlich benötigt man für die allgemeine Berechnung des PDP nur ein Modell und die Trainingsdaten des Modells (Friedman 2001; Goldstein et al. 2014; Hastie et al. 2017, S. 369–370).

Soll die partielle Abhängigkeit der geschätzten Zielgröße $\hat{y}$ aus einem Modell $\hat{f}(\boldsymbol{x})$ auf eine bestimmte Prädiktoren-Teilmenge (engl. *subset*) $\boldsymbol{x}_S$, mit der Größe $l < p$, aus der Gesamtheit an Prädiktoren $\boldsymbol{x} = (x_1, x_2, \dots, x_p)$, untersucht werden, ergibt sich für $\boldsymbol{x}_S$ das Komplement (engl. *complement*) $\boldsymbol{x}_C$, wobei

$$\boldsymbol{x}_C \cup \boldsymbol{x}_S = \boldsymbol{x}$$

gilt. Durch die Beschränkung der menschlichen Wahrnehmung sollte jedoch die zu untersuchende Teilmenge $\boldsymbol{x}_S$ recht klein gehalten werden ($l \in 1, 2, 3$).

Grundsätzlich ist das Ergebnis des Modells $\hat{f}(\boldsymbol{x})$ von allen Prädiktoren abhängig:

$$\hat{f}(\boldsymbol{x}) = \hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_C)$$

Formal ist die Funktion der partiellen Abhängigkeit des Modells $\hat{f}(\boldsymbol{x})$ auf die Teilmenge $\boldsymbol{x}_S$ durch

$$\bar{f}_S(\boldsymbol{x}_S) = E_{\boldsymbol{x}_C}\hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_C) = \int \hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_C)\, dP(\boldsymbol{x}_C)$$

definiert. Jede Teilmenge (konkrete Werte der Prädiktoren) in $\boldsymbol{x}_S$ verfügt dabei wiederum über eine eigene Funktion $\hat{f}(\boldsymbol{x}_S)$ über die partielle Abhängigkeit. In dieser Funktion sind die Werte für $\boldsymbol{x}_S$ fixiert, wohingegen die $\boldsymbol{x}_C$-Werte über die Randverteilung $dP(\boldsymbol{x}_C)$ variieren. Anschließend wird aus den einzelnen Funktionswerten ein Mittelwert gebildet.

Da jedoch weder $\hat{f}(\boldsymbol{x})$ noch $dP(\boldsymbol{x}_C)$ exakt bekannt sind, wird in der Praxis die Funktion durch

$$\overline{f_S}(\boldsymbol{x}_S) = \frac{1}{N}\sum_{i=1}^{N} \hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_{iC})$$

annähernd geschätzt. Bei $\{\boldsymbol{x}_{1C}, \boldsymbol{x}_{2C}, \dots, \boldsymbol{x}_{NC}\}$ handelt es sich um konkrete Datenpunkte für $\boldsymbol{x}_C$ aus den Trainingsdaten, welche zur Erstellung des Modells $\hat{f}(\boldsymbol{x})$ dienten. Auch hier wird für jede Kombination an $\boldsymbol{x}_S$-Werten ein gemittelter Funktionswert $\overline{f_S}(\boldsymbol{x}_S)$ gebildet. Dabei bleiben die Werte für $\boldsymbol{x}_S$ fix, wohingegen $\boldsymbol{x}_{iC}$ über alle N Beobachtungen in den Trainingsdaten variiert.

Um Lücken in den Datenpunkten von $\boldsymbol{x}_S$ schließen zu können und dadurch die Kurve des PDP exakter zu zeichnen, können neben den $\boldsymbol{x}_S$-Werten aus den Trainingsdaten auch frei gewählte reelle Werte für $\boldsymbol{x}_S$ genutzt werden. Generell kann dabei für $\boldsymbol{x}_S$ eine beliebige reelle Zahl gewählt werden. Im PDP selbst werden jedoch in der Regel nur die tatsächlich in den Trainingsdaten vorhandenen $\boldsymbol{x}_S$-Werten als Punkte abgetragen. Die Schrittweite zwischen den einzelnen $\boldsymbol{x}_S$-Werten wird hauptsächlich durch die vorhandene Rechenleistung begrenzt, da die Berechnung aller für einen PDP nötigen Datenpunkte schon bei mittelgroßen Datenmengen einen enormen Rechenaufwand darstellen kann.

2.4.3 Individual Conditional Expectation Plot

Der *Individual Conditional Expectation Plot (ICE)* wurde 2014 in einem Paper von Alex Goldstein et al. vorgestellt und ist sozusagen als Weiterentwicklung des Partial Dependence Plot anzusehen (Goldstein et al. 2014).

Wie auch der PDP dient der ICE zu Visualisierung der partiellen Abhängigkeit eines Modells $\hat{f}(\boldsymbol{x})$ von einer bestimmte Prädiktoren-Teilmenge $\boldsymbol{x}_S$. Wobei auch hier wieder gilt: $\boldsymbol{x}_C \cup \boldsymbol{x}_S = \boldsymbol{x}$. Zur besseren Darstellung soll von nun an nur die partielle Abhängigkeit des Modells $\hat{f}(\boldsymbol{x})$ von einem Prädiktor untersucht werden $|x_S| = 1$.

Im Vergleich zum PDP werden bei dem ICE jedoch nicht die gemittelten Werte der partiellen Abhängigkeit als eine einzelne Kurve dargestellt, sondern alle N Kurven über die bedingten Erwartungswerte des Modells.

Mit Hilfe des Modells $f(\boldsymbol{x})$ und der Beobachtungen $\{(x_{iS}, \boldsymbol{x}_{iC})\}_{i=1}^{N}$ aus den Trainingsdaten wird für alle N beobachteten und fixierten Werte aus $\boldsymbol{x}_{iC}$ eine eigene Kurve der bedingten Erwartung $\hat{f}_S^{(i)}$ über die korrespondierenden x_S-Werte gebildet. Dementsprechend bestehen zu jedem x_S-Wert auf der x-Achse genau N bedingte Erwartungswerte.

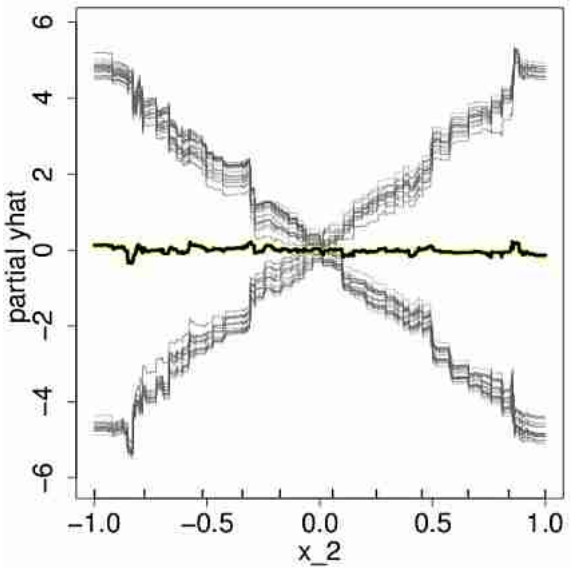


Abbildung 29: Beispielhafter ICE für ein x_2 (Goldstein et al. 2014)

In Abbildung 29 wurde ein Modell auf den Prädiktor x_2 hin untersucht. Die dicke, gelbumrandete Kurve in der Mitte stellt den klassischen PDP dar. Da dieser, abgesehen von kleineren Ausschlägen, annähernd eine Gerade ist, deutet er daraufhin, dass das Modell nicht direkt von dem Wert des Prädiktors x_2 abhängig ist. Die grauen Kurven werden im ICE erstellt und deuten hingegen sehr wohl daraufhin, dass im Modell sehr starke Abhängigkeiten von x_2 bestehen. Jede einzelne Kurve stellt eine bedingte Erwartung $\hat{f}_S^{(i)}$ dar. Dadurch, dass der PDP nur den Durchschnitt der bedingten Erwartungen abbildet, kann es passieren, dass unter Umständen wichtige Informationen herausgefiltert werden. Dies ist beim ICE nicht der Fall.

Des Weiteren werden oftmals nicht alle N Trainingsdatenpunkte als Verläufe im ICE abgetragen, sondern nur 1-20% der Trainingsdaten. Dies hat den Vorteil, dass zum einen wesentlich weniger Rechenleistung benötigt wird, und zu anderen ist die Nachvollziehbarkeit von Effekten und Entwicklungen im ICE ab einer gewissen Anzahl von Verläufen doch erheblich eingeschränkt, da der Plot praktisch nur noch aus sich überlagernden Verläufen bestehen würde.

Centered Individual Conditional Expectation Plot

Für den Fall, dass sich die einzelnen Kurven eines ICE zu stark überlagern und dadurch mögliche Effekte verdeckt werden, kann es sinnvoll sein, einen sogenannten *Centered-ICE-Plot* (c-ICE) zu verwenden. Sollten alle bzw. sehr viele Kurven quasi-parallel zueinander verlaufen, suggeriert der ICE, dass lediglich der Wert der jeweiligen $\boldsymbol{x}_{iC}$ die Höhe der einzelnen Kurven $\hat{f}_S^{(i)}$ vorgibt. Hier kann der c-ICE helfen, um mögliche Trends und Interaktionen zwischen x_S und $\boldsymbol{x}_{iC}$ aufzudecken (Goldstein et al. 2014).

Die Basis ist die gleiche wie bei dem Standard-ICE, allerdings wird hier ein x^* aus der Wertemenge von x_S ausgewählt und alle Bedingte-Erwartungskurven an diesem Punkt zusammengeführt.

Jede Kurve $f_S^{(i)}$des Standard-ICE wird durch die Gleichung

$$\hat{f}_{S,cent}^{(i)} = \hat{f}_S^{(i)} - \hat{f}(x^*, \boldsymbol{x}_{iC}) \tag{1}$$

im Punkt x^* zentriert. Der Term $\hat{f}(x^*, \boldsymbol{x}_{iC})$ bezeichnet für jede Kurve $\hat{f}_S^{(i)}$ einen konstanten Wert an der Stelle $x_S = x^*$. Alle Punkte der Kurve $\hat{f}_S^{(i)}$ werden durch diesen Wert nach oben oder unten verschoben, so dass der Punkt $\left(x^*, \hat{f}(x^*, \boldsymbol{x}_{iC})\right)$ als Basis für alle Kurven dient. Es hat sich gezeigt, dass bei der Wahl von x^* das Minimum oder Maximum von x_S am besten geeignet ist.

Derivative Individual Conditional Expectation Plot

Eine weitere Möglichkeit, Interaktionen zwischen dem Ergebnis eines Modells $\hat{f}(\boldsymbol{x})$ und einem bestimmten Prädiktor x_S darzustellen, besteht darin, $\hat{f}(\boldsymbol{x})$ nach x_S abzuleiten. Dadurch wird die Änderungsrate des Modellergebnisses in Bezug auf einen konkreten x_S-Wert ermittelt. Es wird also die Steigung der Hyperebene des Modells am Punkt $(x_S, \boldsymbol{x}_{iC})$ berechnet. Da die Funktion des Modells $\hat{f}(\boldsymbol{x})$ nicht direkt bekannt ist bzw. nicht analytisch differenziert werden kann, wird sie hier **numerisch** differenziert.

Wie auch beim ICE und c-ICE wird für jede x_S-Wert und $\boldsymbol{x}_{iC}$-Kombination eine eigene Kurve abgetragen. Der so entstandene Graph, mit N Kurven, wird als *Derivative ICE Plot* bezeichnet (d-ICE) (Goldstein et al. 2014).

Wenn in einem Modell zwischen x_S und $\boldsymbol{x}_C$ keinerlei Interaktionen vorhanden sind, kann $\hat{f}(\boldsymbol{x})$ wie folgt beschrieben werden:

$$\hat{f}(\boldsymbol{x}) = \hat{f}(x_S, \boldsymbol{x}_C) = g(x_S) + h(\boldsymbol{x}_C) \rightarrow \frac{\partial \hat{f}(\boldsymbol{x})}{\partial x_S} = g'(x_S)$$

Da keine Interaktionen zwischen x_S und $\boldsymbol{x}_C$ bestehen, fällt $h(\boldsymbol{x}_C)$ nach dem Ableiten des Modells nach x_S weg. Dementsprechend würde der d-ICE aus N Kurven bestehen, welche exakt übereinanderliegen. Bei dem normalen ICE wäre es ähnlich: er würde aus N Kurven bestehen, welche die gleiche Form haben, aber sich in der Höhe durch die unterschiedlichen Werte von $\boldsymbol{x}_C$, unterscheiden.

Wenn es jedoch zu Interaktionen kommt, würden diese durch eine Vielzahl unterschiedlicher Kurven im d-ICE aufgezeigt werden.

2.5 Genetische Algorithmen

Die Optimierung beschäftigt sich damit, zu einer gegebenen Problemstellung, sei es eine Minimierung oder Maximierung, eine Lösung zu finden, welche besser ist als alle anderen möglichen Lösungen. Die Suche nach dem (globalen) Optimum wird durch viele Faktoren erschwert, wie bspw. eine hohe Anzahl von lokalen Optima, zu erfüllende Nebenbedingungen, nicht ableitbare Funktionen, Unstetigkeiten, Rauschen usw. An diesen einzelnen oder Kombinationen von Faktoren scheitern jedoch viele klassische Optimierungsverfahren. Die sogenannten *Genetischen Algorithmen*[30] *(GA)* haben sich jedoch als sehr robust und zuverlässig bei Optimierungsproblemen mit den o.g. Restriktionen erwiesen (Kramer 2017, S. 3–4).

Kurze Bemerkung: Die einschlägige Literatur zu diesem Thema ist so gut wie ausschließlich in englischer Sprache gehalten. Der Einheitlichkeit und Wiedererkennbarkeit halber werden hier vornehmlich die englischen Fachbegriffe verwendet, wobei eine deutsche Übersetzung, zur Einordnung der Begriffe, als Fußnote vermerkt ist.

[30] Engl. „Genetic Algorithms“

GAs nutzen die Prinzipien der Genetik und der natürlichen Auslese[31] der Natur zur Optimierung von Problemstellungen. Sie nutzen dabei nicht nur eine einzige Lösung, um das Optimum zu finden, sondern eine ganze Population von möglichen Lösungen. Jede mögliche Lösung wird dabei durch die DNS[32] bzw. das Chromosom eines der Individuen aus der Population repräsentiert. Das Chromosom eines Individuums gibt demnach auch vor, wie gut es an die Umwelt bzw. an das jeweilige Optimierungsproblem angepasst ist. Dies geschieht über die sogenannte *Fitness*[33]. Im Laufe der Optimierung durchschreitet die Population mehrere Generationen. In jeder Generation wird ein Teil der besten Individuen über ein gewisses Auswahlverfahren zur Fortpflanzung zugelassen, bei der i.d.R. zwei bestehende Individuen Informationen ihrer Chromosomen austauschen und so neue Individuen zeugen. Weiterhin kann es bei einem kleinen Teil der Population gelegentlich dazu kommen, dass deren Chromosom durch Mutation verändert wird, um neue Impulse zu setzen. Dadurch, dass nur die guten Individuen eine Generation „überleben", erhöht sich der Anteil an guten Lösungen von Generation zu Generation und die Population tendiert zunehmend zum (globalen) Optimum hin (Goldberg 1989, S. 1; Kramer 2017, S. 5; Sarker et al. 2001).

Der Unterschied zu herkömmlichen Optimierungsverfahren liegt nun darin, dass GAs oftmals nicht direkt die zu optimierenden Parameter benutzen, sondern eine Verschlüsselung über das Chromosom. Weiterhin wird über eine Vielzahl von Punkten im Lösungsraum nach dem Optimum gesucht, wobei für jeden dieser Punkte ein Fitness-Wert berechnet werden kann, was den Ansatz probabilistisch und nicht deterministisch macht (Goldberg 1989, S. 7).

[31] Engl. „Survival of the Fittest"; in Anlehnung an die Evolutionstheorie von Charles Darwin
[32] Desoxyribonukleinsäure
[33] Zu Deutsch „Tauglichkeit"

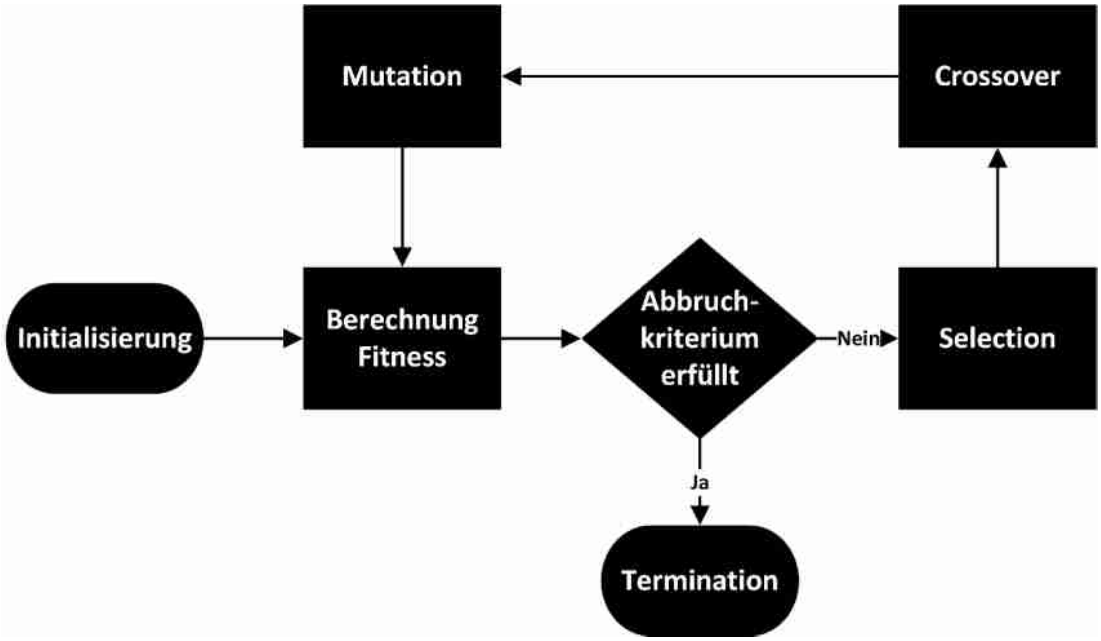


Abbildung 30: Schritte eines Genetischen Algorithmus

Dabei durchläuft jeder GA unterschiedliche Schritte, siehe dazu Abbildung 30. Im ersten Schritt wird eine Population aus zufälligen Individuen erstellt. Im Anschluss wird für jedes dieser Individuen eine Fitness berechnet, welche angibt, wie gut sich das Individuum für das jeweilige Problem eignet. Die Berechnung der Fitness ist problemspezifisch, d.h. sie muss für jede Problemstellung individuell erstellt werden. Im nächsten Schritt werden über die *Selection,* oder auch *Recombination* genannt, diejenigen Individuen aus der aktuellen Population ausgewählt, welche überleben sollen. Dabei gibt es verschiedene Methoden, wie die Auswahl der Überlebenden genau abläuft. Nach der Selection pflanzt sich ein Teil der Population fort. Im sogenannten *Crossover*[34] tauschen zwei (oder mehr) Eltern ihre Informationen untereinander aus und erzeugen daraus deren *Offspring*[35]. Auch hier gibt es wiederum verschiedene Ansätze, wie genau die Auswahl der Eltern abläuft bzw. wie aus ihnen der Nachwuchs zusammengesetzt wird. Im vorletzten Schritt, der *Mutation*, werden Teile der DNS einiger weniger Individuen zufällig innerhalb eines auswählbaren Schemas verändert. Letztlich werden noch für die neuen Individuen der Population die Fitness-Werte berechnet und der Kreislauf beginnt in der nächsten Generation von vorne. Dieser

[34] Zu Deutsch „Kreuzung“
[35] Zu Deutsch „Nachwuchs“

Kreislauf wird solange wiederholt, bis ein vorher definiertes Abbruchkriterium erreicht wird. Ist dies der Fall, endet der GA mit der *Termination*[36] und das Optimierungsergebnis wird präsentiert (Schwefel 1995, S. 152–153).

2.5.1 Kodierung

Wie bereits angesprochen, werden bei einem GA nicht die Variablen eines Optimierungsproblems direkt optimiert, sondern deren Repräsentation über die DNS bzw. das Chromosom der Individuen einer Population. Dabei wird die Kodierung einer möglichen Lösung im Zusammenhang mit GAs auch als *Genotyp* bezeichnet. Die einzelnen Elemente eines Genotyps sind die *Gene*. Die tatsächlich zur Berechnung einer Fitness verwendeten Werte, welche durch den Genotyp verschlüsselt werden, bezeichnet man als *Phänotyp*. Wie genau ein Genotyp aufgebaut ist, bestimmt die Wahl der Operatoren für die einzelnen Schritte eines GAs. In den allermeisten Fällen handelt es sich bei einem Genotyp um eine Liste von Einträgen, wobei diese Liste drei unterschiedliche Formen annehmen kann: Sind die Einträge nur Bits (0 und 1), wird sie als *Bit-String* bezeichnet, bei Fließkommazahlen als *Vektoren* und letztlich beinhaltet der Genotyp bei kombinatorischen Problemen[37] eine Liste von *Symbolen* (Kramer 2017, 11–12, 15).

Die klassische Herangehensweise bei der Kodierung von Variablen ist die über Bit-Strings. Dabei wird jedes Gen durch dieselbe Anzahl an Bits repräsentiert. Bei einem Phänotypen mit drei Einträgen, wobei jeder Eintrag einen Wert zwischen 0 und 15 einnehmen kann (mit 4 Bits pro Gen), würde der Bit-String des Genotypen wie folgt aussehen (Herrera et al. 1998; Michalewicz 1996, S. 100):

$$(15\ 1\ 9)_{10} \leftrightarrow (1111\ 0001\ 1001)_2$$

Die Verwendung von Bit-Strings hat sich im Laufe der Zeit jedoch als recht unpraktisch im Umgang herausgestellt. Um den GA näher an die eigentliche Problemstellung heranzurücken, werden anstelle der Bit-Strings-Genotypen Vektoren mit Gleitkommazahlen (engl. *Floating Point Numbers*) bzw. ganzen Zahlen (engl. *Integer*) direkt als Genotypen verwendet. Die Kodierung über Integer kann zum einen für Variablen verwendet werden,

[36] Zu Deutsch „Beendigung“
[37] Wie bspw. das „Problem des Handlungsreisenden“ (engl. Traveling Salesperson Problem)

welche nur ganzzahlig auftreten oder auch für kategorische[38] und ordinale[39] Variablen. Gleitkommazahlen werden dann verwendet, wenn die Variablen kontinuierlicher Natur sind (Arumugam und Rao 2007; Eiben und Smith 2010, S. 41).

Diese sogenannten *Real-Coded-GAs*[40] *(RCGA)* bieten mehrere Vorteile gegenüber den klassischen binären GAs (engl. *Binary-Coded-GA [BCGA]*) (Deb 2000; Herrera et al. 1998; Michalewicz 1996, 101, 105-106):

Zum einen können sie einen sehr großen Lösungsraum darstellen, und da sie nicht auf eine gewisse Anzahl von Bits beschränkt sind, können diese Räume auch im Nachhinein leicht erweitert werden. Dadurch, dass hier Gleitkommazahlen verwendet werden, sind auch wesentlich präzisere Ergebnisse möglich. Man könnte zwar auch einen BCGA um zusätzliche Bits erweitern, allerdings würde dies die Rechenzeiten des GA erhöhen. Um zu verdeutlichen, wo diese Rechenzeiten herkommen, kann man das vorige Beispiel nun mit Gleitkommazahlen wiederholen und dazu den aktuellen IEEE-754-Standard zur binären Kodierung verwenden. Man bräuchte nun pro Zahl einen String mit 32 Bits, also insgesamt 96 Bits (IEEE 754-2008):

$$(15{,}567\ldots\ 1{,}070\ldots\ 9{,}436\ldots)_{10} \leftrightarrow \begin{pmatrix} 01000001011110010001001001101111 \\ 00111111100010001111010111000011 \\ 01000001000101101111100111011011 \end{pmatrix}_2^T$$

Für jedes dieser 96 Bits müsste man nun Crossover- und Mutation-Operationen durchführen und zur Evaluation der Fitness diesen String in letztlich drei Gleitkommazahlen zurückentschlüsseln, wohingegen man bei RCGAs direkt auf die Zahlen zugreifen kann. Ein weiterer Grund für die Verwendung von RCGAs liegt darin, dass sie wesentlich intuitiver im Umgang sind und somit das Design eines GAs vereinfachen. Dies macht sich vor allem bei der Berücksichtigung von restriktiven Nebenbedingungen und beim Einbau von problemspezifischem Expertenwissen bemerkbar.

[38] Variablen, welche keiner Ordnung unterliegen (bspw. männlich – weiblich)
[39] Variablen, welche einer inneren Rangfolge entsprechen (bspw. klein – mittel – groß)
[40] Zu Deutsch: „echt verschlüsselt“

2.5.2 Initialisierung

Wie bei allen Optimierungsalgorithmen hat der Startpunkt der Berechnung sehr großen Einfluss auf den Verlauf der Optimierung. Zum einen wird durch ihn festgelegt, ob ein globales Optimum überhaupt erreicht werden kann. Zum anderen wird dadurch bestimmt, wie weit man vom globalen oder lokalen Optimum entfernt ist und damit auch, wie viele Iterationsschritte benötigt werden, um dies zu erreichen. Damit der GA die Möglichkeit hat, das globale oder wenigstens ein gutes lokales Optimum zu finden, sollte die Anfangspopulation möglichst gleichmäßig im gesamten Lösungsraum verteilt sein. Dabei wird die Spannweite für jedes Gen durch dessen obere und untere Schranke festgelegt. Um das Optimierungsergebnis zu verbessern bzw. zu beschleunigen, kann es hilfreich sein, bei der Platzierung der Anfangspopulation vorhandenes Expertenwissen mit einfließen zu lassen (Kramer 2017, S. 12; Schwefel 1995, S. 158–159).

2.5.3 Selection

Die Selection wählt Individuen aus der Population der alten Generation für die neue Generation aus, wobei die Individuen mit guter Fitness präferiert ausgewählt werden. Dadurch werden auch gleichzeitig die (guten) Genotypen der einzelnen Individuen mit in die nächste Generation genommen, was auch ein erwünschter Zustand ist, da ein Teil dieser Individuen sich im nächsten Schritt (dem Crossover) paaren wird (Arumugam und Rao 2007; Miller und Goldberg 1995).

Bei der Selection gibt es verschiedene Verfahren, nach denen die Auswahl der Individuen ablaufen kann. Bei Ihnen ist jedoch der Faktor der *Selection Pressure*[41] zu beachten. Sie gibt an, zu welchem Grad die „guten“ Individuen bei der Selection bevorzugt werden. Ist die Selection Pressure hoch, werden nur die besten Individuen für die nächste Generation ausgewählt, ist sie niedrig, haben auch schwächere Individuen die Chance weiterzukommen. Die Selection Pressure bestimmt somit, wie stark sich die durchschnittliche Fitness der Individuen pro Generation verbessert und somit auch, wie schnell der GA in ein Optimum konvergiert. Ist die Selection Pressure zu niedrig angesetzt, konvergiert der GA nur sehr langsam und braucht u.U. unnötig lange, um zu einer optimalen Lösung zu

[41] Zu Deutsch: „Auswahldruck“

kommen. Ist die Selection Pressure jedoch zu hoch, konvergiert der GA wesentlich schneller, allerdings zu dem Preis, dass er möglicherweise nur ein lokales und nicht das globale Optimum gefunden hat (Miller und Goldberg 1995).

Tournament-Selection

Eine sehr bekannte und wahrscheinlich auch die am weitesten verbreitete Selection-Methode ist die *Tournament-Selection* (TS)[42]. Dabei ist die Funktion der TS recht simpel: Aus der Menge an Individuen aus einer Population werden s Individuen zufällig ausgewählt. Mit s wird die Größe des Tournament beschrieben (engl. *Tournament Size*). Diese s Individuen treten nun gegen einander an, wobei nur das Beste (das mit der höchsten Fitness) der s Individuen in die nächste Generation übernommen wird. Dieser Vorgang wird solange wiederholt, bis die ursprüngliche Populationsgröße wieder erreicht ist. Es gilt hier jedoch zu beachten, dass die Individuen (egal ob Gewinner oder Verlierer) nach dem Tournament wieder in den Pool die möglichen Kandidaten für das nächste Tournament zurückgelegt werden. Das bedeutet, manche Individuen können mehrfach in der nächsten Generation vertreten sein. Daneben hat durch dieses Verfahren jedes Individuum (auch die relativ schlechten) die Chance, in die nächste Generation zu kommen (Kramer 2017, S. 17; Miller und Goldberg 1995).

Dadurch, dass hier nur die Individuen mit der höchsten Fitness aus jedem Tournament in die nächste Generation kommen, wird die durchschnittliche Fitness ansteigen. Die Selection Pressure kann bei der TS durch die Tournament-Size s eingestellt werden. Mit steigendem s steigt auch die Selection Pressure, da der Gewinner aus einem großen Turnier im Schnitt eine höhere Fitness aufweisen wird als Gewinner aus kleineren Turnieren. Da jedes Tournament unabhängig von den anderen ist, können die einzelnen Turniere parallel ablaufen, was die TS, neben der sehr einfachen Implementierung, außerdem zu einem zeiteffizienten Algorithmus macht (Goldberg und Deb 1991; Miller und Goldberg 1995).

Neben der TS existieren noch weitere Verfahren zur Auswahl von Individuen. Hierzu zählen u.a. die *Ranking Selection* oder die *Fitness Proportional Selection*. Ein Vergleich einiger

[42] Zu Deutsch: „Turnierauswahl“

Verfahren wird in Goldberg und Deb 1991 sowie Eiben und Smith 2010, S. 59–66 beschrieben.

2.5.4 Crossover

Der Crossover-Operator verkörpert die grundsätzliche Strategie zur Verbesserung von möglichen Lösungen in der Population, indem er dafür sorgt, dass das „gute“ Genmaterial von zwei oder mehr Eltern-Individuen (welche die Selection überstanden haben) untereinander getauscht bzw. vermischt wird. Die Idee hier hinter sieht vor, dass die Genotypen der Eltern jeweils Teile einer guten Lösung (hohe Fitness) beinhalten. Durch eine Kombination der beiden Teile soll nun ein Individuum geschaffen werden, dessen Fitness noch wesentlich besser ist als die der Eltern. Dabei wird zwischen zwei Ansätzen unterschieden: Der Parent-Centric-Approach generiert Nachwuchs mit Genotypen, welche nah bei jeweils einem der Eltern sind, wohingegen die einzelnen Gene des Nachwuchs-Genotyps durch den Mean-Centric-Approach eher in der Nähe des Mittelwertes der beiden Eltern liegen. Abbildung 31 zeigt einen schematischen Parent-Centric-Crossover von zwei Eltern (engl. Parents) und den daraus resultierenden Offspring mit einem Genotyp, welcher nur zwei Gene enthält. Die Farben kennzeichnen dabei in ihrer Intensität unterschiedliche Informationen der Gene. (Arumugam und Rao 2007; Kramer 2017, S. 12–13; Lim et al. 2017; Umbarkar und Sheth 2015).

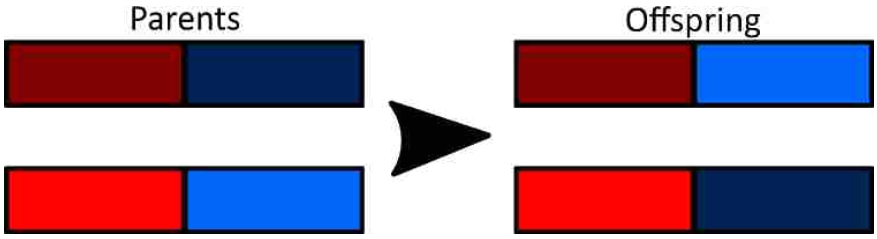


Abbildung 31: Crossover in Anlehnung an Kramer 2017, S. 13

Bei BCGAs (z.T. auch bei ganzzahligen RCGAs) werden vorrangig Crossover-Operatoren eingesetzt, welche die beiden Genotype in zwei oder mehr Teile splitten und diese anschließend untereinander tauschen. Dabei kann jeder der Teile mehrere benachbarte Gene beinhalten. Dazu zählen der *One-Point Crossover* sowie der *N-Point Crossover* (Eiben und Smith 2010, S. 47–48).

Es wird allerdings nicht jedes Individuum, welches die Selection überstanden hat, zum Crossover zugelassen. Bei den meisten GAs bestimmt die Crossover-Wahrscheinlichkeit (CXPB) für jedes Individuum einzeln, wie wahrscheinlich es ist, dass es ausgewählt wird. Dabei sollte die CXPB doch recht hoch angesiedelt werden, um gute Ergebnisse erzielen zu können. Typischerweise liegt sie im Bereich von 60 bis 80 Prozent (Arumugam und Rao 2007; Kramer 2017, S. 13).

Neben der im Folgenden erläuterten Crossover-Variante wird von Umbarkar und Sheth 2015 eine Vielzahl weiterer Methoden beschrieben.

Uniform Crossover

Der *Uniform Crossover* ist eine recht einfache Crossover-Variante (siehe Abbildung 32). Zur Erzeugung des einen der beiden Offsprings wird positionsweise (Gen für Gen) mit einer gewissen Wahrscheinlichkeit (i.d.R. $p_c = 0{,}5$) das Gen des ersten Parents ausgewählt. Die nichtbesetzten Gene werden anschließend mit den Genen des zweiten Parents aufgefüllt. Der zweite Offspring wird dementsprechend invers zum Ersten zusammengesetzt (Eiben und Smith 2010, S. 48–49).

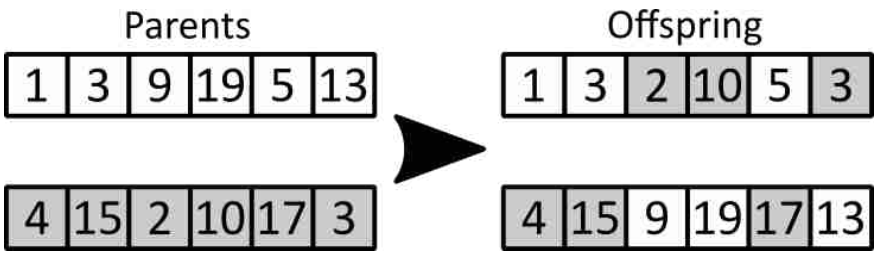


Abbildung 32: Uniform Crossover

2.5.5 Mutation

Durch die Mutation werden in den Teilen der Genotypen zufällige Veränderungen (Mutationen) vorgenommen. Durch diese Mutationen soll eine größere Anzahl unterschiedlicher Individuen in der Population erzeugt werden und dafür sorgen, dass der GA den Lösungsraum besser erforschen kann und nicht in lokalen Optima stecken bleibt (Lim et al. 2017).

Genauso wie beim Crossover werden auch hier nicht alle Individuen der Population zur Mutation zugelassen. Durch die Mutation-Wahrscheinlichkeit (MUTPB) wird geregelt, wie

viele Individuen mutiert werden. Dabei liegt die MUTPB weit unter der CXPB und sollte nur mit großer Sorgfalt eingestellt werden, da die Performance des GAs hierauf recht empfindlich reagieren kann. Typische Werte für die MUTPB liegen im Bereich zwischen 1 und 10 Prozent (Arumugam und Rao 2007; Kramer 2017, S. 13).

Auch hier stehen wiederum eine Vielzahl von unterschiedlichen Mutation-Varianten zur Verfügung, wobei die ausgewählte Variante drei Bedingungen erfüllen sollte (Kramer 2017, S. 13–14):

1. Erreichbarkeit (engl. Reachability): Es muss eine minimale Chance bestehen, dass jeder Punkt im Lösungsraum von einem beliebigen Punkt aus erreichbar ist. Ist dies nicht der Fall, kann das globale Optimum vom GA u.U. nicht gefunden werden.
2. Erwartungstreue (engl. Unbiasedness): Die Mutation eines Gens sollte nicht nur vornehmlich in eine bestimmte Richtung gehen, sondern möglichst gleichverteilt sein.
3. Skalierung (engl. Scalability): Die Stärke einer Mutation sollte mit der Spannweite des Lösungsraums skalierbar sein.

Random Resetting

Bei dem *Random Resetting* (engl. „zufällige Neueinstellung“) wird mit einer definierten Wahrscheinlichkeit p_m, für jedes Gen ein neuer zufälliger Wert aus der Menge der zulässigen Werte bestimmt. Dabei können alle zulässigen Werte mit derselben Wahrscheinlichkeit gezogen werden. Diese Mutationsart wird bei ganzzahlig kodierten GAs verwendet und bietet sich insbesondere bei kategorischen Variablen an, da diese keinem natürlichen Ordnungsschema unterliegen (Eiben und Smith 2010, S. 43).

Creep Mutation

Die *Creep Mutation* (engl. „schleichende Mutation“) wird ebenfalls bei ganzzahlig kodierten GAs verwendet, wobei die Gene hier geordnet sein sollten (ordinale Variablen). Hierbei wird wiederum mit einer Wahrscheinlichkeit p_m für jedes Gen ein kleiner positiver oder negativer Wert hinzuaddiert. Dieser Wert wird ebenfalls zufällig erzeugt, wobei kleinere Werte eher gezogen werden als größere. Man könnte hier beispielsweise die *Gauß'sche Normalverteilung* verwenden. Allerdings gilt es zu beachten, dass hier zusätzliche

Parameter, wie z.B. die Standardabweichung, eingestellt werden müssen. Wenn sich die Gene in ihrer Spannweite stark unterscheiden, kann es von Vorteil sein, mehrere Mutations-Einstellungen parallel zu verwenden: Bspw. eine Creep Mutation mit kleiner Standardabweichung und eine mit einer großen (Eiben und Smith 2010, S. 43–44).

Weitere Mutations-Operatoren können z.B. in Herrera et al. 1998 nachgeschlagen werden.

2.5.6 Fitness-Funktion

Die Evaluation der Fitness eines Individuums ist wohl das Herzstück eines jeden GAs. Dabei stellt die Fitnessfunktion die Verbindung zwischen Genotyp, Phänotyp und Fitness her, indem sie eine Bewertung abgibt, wie gut sich eine jede Lösung des GA für das jeweilige Problem eignet. Dazu muss, je nach Kodierungsart, der Genotyp zunächst in den Phänotyp entschlüsselt werden. Anschließend wird für die Phänotypen, mit Hilfe der Fitnessfunktion, die Fitness berechnet und den Genotypen zugeordnet. Handelt es sich um ein Optimierungsproblem, welches nach mehreren Kriterien hin optimiert werden soll, braucht man auch dementsprechend mehrere Fitness-Funktionen. Die Ergebnisse dieser Funktionen können dann bspw. über gewichtete Mittelwerte in einer einzigen Fitness vereinigt werden. Die Fitnessfunktion ist, genauso wie die Repräsentierung der einzelnen Variablen in den Genen, problemspezifisch und muss individuell entworfen werden (Kramer 2017, S. 15–16).

Bei den meisten Optimierungsproblemen ist der mögliche Lösungsraum, auf Grund von Nebenbedingungen (engl. *Constraints*), jedoch eingeschränkt. Dabei kann es sich z.B. um logische Restriktionen, Materialeigenschaften oder physische Beschaffenheiten handeln. Für Optimierungen im Allgemeinen werden solche Nebenbedingungen als Gleichungen und Ungleichungen mathematisch formuliert. Dies hat den Vorteil, dass nicht nur zwischen gültiger und ungültiger Lösung unterschieden wird, sondern auch ein Maß vorliegt, wie (stark) ungültig eine Lösung ist (Kramer 2017, S. 39–40).

Für den Umgang mit ungültigen Lösungen stehen unter anderen die folgenden Optionen zur Verfügung:

Die gedanklich einfachste Lösung besteht darin, einfach keine ungültigen Lösungen in die Population zu lassen. Dieses Verfahren wird als *Death Penalty*[43] bezeichnet. Verletzt eine Lösung mindestens eine Nebenbedingung, werden solange neue Lösungen (durch Crossover und Mutation) erzeugt, bis alle Lösungen in der Population zulässig sind. Dies klingt zunächst nach einer sehr einfachen Vorgehensweise. Allerdings ist sie sehr ineffizient für den Fall, dass nur sehr wenige gültige Lösungen vorhanden sind. Da das Optimum in den meisten Fällen an der Grenze einer oder mehrerer Nebenbedingungen liegt, ist dies insbesondere gegen Ende der Optimierung der Fall, da hier viele Lösungen über den zulässigen Bereich hinausschießen und dadurch ungültig werden (Kramer 2017, S. 41).

Penalty-Funktionen verändern die Fitness von ungültigen Lösungen und machen sie dadurch für die Selection unattraktiver. Bei Maximierungsproblemen wird die Fitness reduziert, wohingegen sie bei Minimierungsproblemen erhöht wird. Dabei skaliert die Stärke dieser Veränderung mit der des Verstoßes bzw. der Verstöße. Der neue Fitness-Wert $f'(\boldsymbol{x})$ des Phänotyps $\boldsymbol{x}$ setzt sich dann aus seiner eigentlichen Fitness $f(\boldsymbol{x})$ und einem Penalty-Wert $g(\boldsymbol{x})$, welcher mit einem Penalty-Faktor α weiter angepasst werden kann, zusammen:

$$f'(\boldsymbol{x}) = f(\boldsymbol{x}) + \alpha \cdot g(\boldsymbol{x})$$

Dabei misst die Penalty-Funktion $g(\boldsymbol{x})$, wie sehr die Nebenbedingungen des Optimierungsproblems verletzt wurden. Je größer der Verstoß, desto höher ist auch der Penalty-Wert (Kramer 2017, S. 41).

Die Verwendung einer Penalty-Funktion erlaubt die Suche im unzulässigen Raum, was von Vorteil ist, da das Optimum (wie schon erwähnt) zumeist an der Grenze zwischen zulässigem und unzulässigem Raum liegt. Über den Penalty-Faktor α kann zudem eingestellt werden, wie das Verhältnis zwischen gültigen und ungültigen Lösungen in der Population aussehen soll. Des Weiteren sollte beachtet werden, dass das Design der Penalty-Funktion, wie auch bei der Fitness-Funktion, problemspezifisch ist und keine allgemeinen Empfehlungen existieren. Insbesondere das Anpassen von Faktoren innerhalb

[43] Zu Deutsch „Todesstrafe“

der Penalty-Funktion ist eine Herausforderung und bedarf einer Vielzahl von Experimenten während des Designs eines GAs (Deb 2000; Kramer 2017, S. 42; Kramer et al. 2013).

Für den Fall, dass mehrere Nebenbedingungen vorhanden sind, sollte beachtet werden, dass diese normalisiert werden. Dies ist insbesondere dann wichtig, wenn die Nebenbedingungen in unterschiedlichen Einheiten vorliegen, wie bspw. in Metern, Liter oder Watt (Deb 2001, S. 127; Deep et al. 2009).

Eine Formel zur Normalisierung der Nebenbedingungen wird in Carlson 1995 beschrieben:

Die Schwere des Verstoßes, für eine Maximierung der Nebenbedingung[44] $m_j(\boldsymbol{x}) \leq h_j$, wird durch normalisierte Distanz zum zulässigen Lösungsraum d_j wie folgt beschrieben:

$$d_j = \frac{h_j - m_j(\boldsymbol{x})}{\frac{m_j(\boldsymbol{x}) + h_j}{2}}$$

Die Fitness von $\boldsymbol{x}$ wird dann über die Summe aller normalisierten Distanzen der Nebenbedingungen berechnet:

$$f'(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) & \text{falls } \boldsymbol{x} \text{ zulässig} \\ 0 - \sum_{j=1}^{m} d_j & \text{falls } \boldsymbol{x} \text{ unzulässig} \end{cases}$$

In einigen Fällen kann es sehr schwer sein, eine Penalty-Funktion zu Parametrisieren; insbesondere dann, wenn die optimale Fitness bei unterschiedlichen Konfigurationen[45] des Problems starken Schwankungen unterliegt. Deb 2000 hat hierzu einen Ansatz vorgestellt, bei dem eine Parametrisierung nicht notwendig ist. Die Individuen werden zunächst in zulässige und unzulässige unterteilt. Anschließend wird als Selection-Operator die bereits beschriebene Tournament Selection verwendet. Bei einer Tournament Size von 2 können drei Fälle auftreten:

1. Jede zulässige Lösung wird einer unzulässigen bevorzugt
2. Bei zwei zulässigen Lösungen wird die mit der besseren Fitness bevorzugt
3. Bei zwei unzulässigen Lösungen wird die mit dem kleineren Penalty-Wert bevorzugt

[44] Damit $\boldsymbol{x}$ zulässig ist, soll $m_j(\boldsymbol{x})$ größer als h_j sein.
[45] Bspw. durch geänderte Nebenbedingungen

Dadurch, dass zulässige Lösungen immer den unzulässigen bevorzugt werden, ist es nicht notwendig, eine Penalty-Funktion aufzustellen, welche künstlich die Fitness einer unzulässigen Lösung verschlechtert. Des Weiteren wird in dem Ansatz die Fitness wie folgt beschrieben:

$$f'(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) & \text{falls } \boldsymbol{x} \text{ zulässig} \\ f_{max} + \sum_{j=1}^{m} \langle g(\boldsymbol{x})_j \rangle & \text{falls } \boldsymbol{x} \text{ unzulässig} \end{cases}$$

Wobei f_{max} die schlechteste Fitness der zulässigen Lösungen in der aktuellen Population darstellt. Dadurch, dass unzulässige Lösungen immer mindestens auf dem Wert von f_{max} sind, können die drei genannten Fälle erfüllt werden.

Es gilt auch hier wieder der Vollständigkeit halber zu erwähnen, dass noch weitere Verfahren für den Umgang mit Nebenbedingungen existieren, wie z.B. das *Repairing* oder *Decoders*.

2.5.7 Termination

Das Abbruchkriterium (engl. *Termination Condition*) bestimmt bei einem GA, wann die innere Schleife von:

Fitness → Selection → Crossover → Mutation → Fitness

beendet und das Optimierungsergebnis präsentiert wird. Dabei werden in der Praxis für gewöhnlich zwei verschiedene Ansätze für die Definition des Abbruchkriteriums verfolgt: Die einfachste Variante sieht eine im Vorhinein definierte Anzahl von Generationen oder Fitness-Evaluationen vor. Ist diese Anzahl erreicht, beendet der GA die Optimierung. Dies hat jedoch den Nachteil, dass der GA eventuell schon lange in das Optimum konvergiert ist oder vielleicht noch einige Generationen benötigt, um überhaupt das Optimum zu erreichen und dennoch wird die Optimierung beendet. Um dem entgegenzuwirken, kann der GA auch auf Basis seiner Konvergenz in ein Optimum beendet werden. Dies ist bspw. dann der Fall, wenn kein wesentlicher Fortschritt in der Fitness der Individuen festzustellen ist (der GA stagniert). Eine Herangehensweise, dies zu prüfen, ist, die Fitness der Individuen aus aufeinander folgenden Generationen miteinander zu vergleichen. Verbessert sich deren Fitness nicht mehr um einen gewissen (statischen) Wert, gilt der GA als konvergiert und wird abgebrochen (Arumugam und Rao 2007; Kramer 2017, S. 17–18; Safe et al. 2004).

Darüber hinaus kann eine durchaus beachtliche Anzahl weiterer Abbruchkriterien in Ghoreishi et al. 2017 nachgeschlagen werden.

2.5.8 Multimodale Probleme und Restart-Strategien

Die meisten Optimierungsprobleme weisen eine Reihe von Punkten auf, welche eine bessere Lösung (Fitness) als all ihre Nachbarpunkte haben (lokale Optima[46]), wobei jede dieser Lösungen nicht so gut ist wie das globale Optimum. Diese Art von Problemen wird als *multimodales Optimierungsproblem* bezeichnet. Abbildung 33 zeigt beispielhaft die Fitnessoberfläche eines solchen Problems (Eiben und Smith 2010, S. 154).

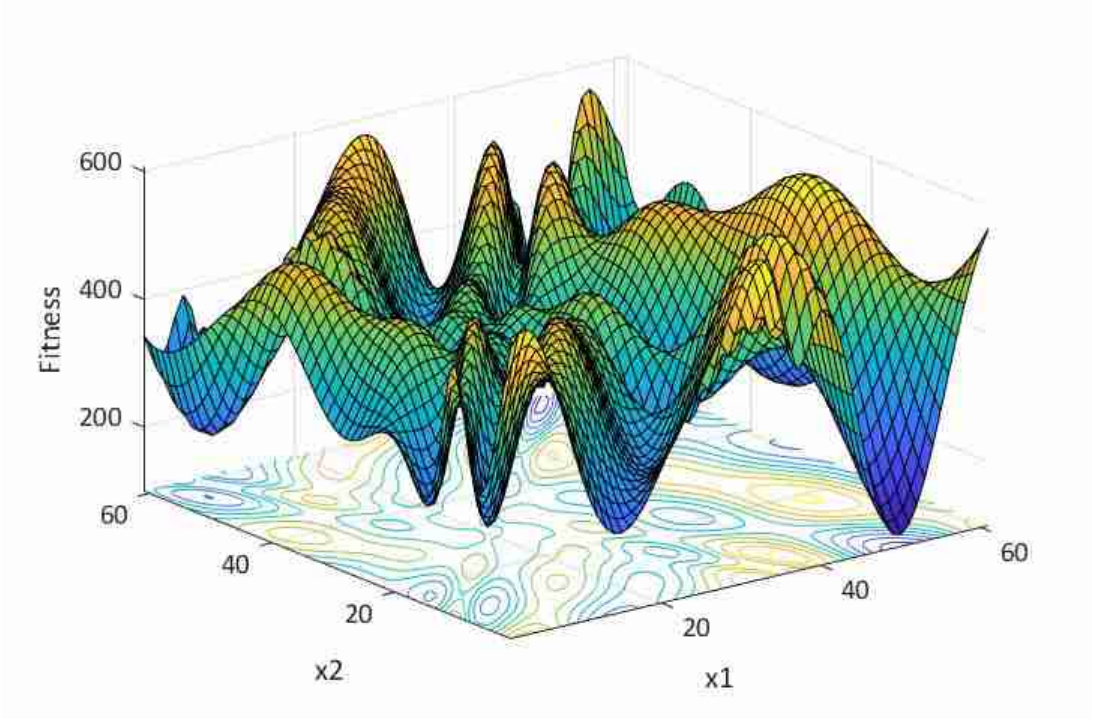


Abbildung 33: Multimodales Optimierungsproblem

Das Problem liegt darin, dass lokale Optima (genauso wie das globale Optimum) den GA „anziehen“. Es kann daher oftmals dazu kommen, dass der GA in einem lokalen Optimum endet und nicht wie gewünscht in das globale Optimum konvergiert. Man könnte natürlich durch eine entsprechend große Startpopulation versuchen, den Lösungsraum möglichst komplett abzudecken oder mit einer höheren MUTPB bzw. anderem Mutations-Operator, dem GA zu unterstützen, sich von den lokalen Optima wieder zu lösen, allerdings ist dies

[46] Werden im Kontext von GAs im Englischen auch als „*Niches*“ bezeichnet

auch kein Garant dafür, dass der GA nicht trotzdem in ein lokales Optimum konvergiert (Eiben und Smith 2010, S. 154; Kramer 2017, S. 31–32).

Eine grundsätzlich recht einfache Methode, dennoch das globale Optimum zu finden, besteht darin, den GA mit einer anderen, zufälligen Startpopulation neu zu starten (engl. *Restarts*). Diese Restarts sorgen dafür, dass der GA aus anderen Teilen des Lösungsraumes die Optimierung beginnt und nun (hoffentlich) in das globale Optimum konvergiert (Kramer 2017, S. 32; Loshchilov 2013).

Neben dem simplen mehrfachen Neustarten des GAs wurden auch wesentlich elaboriertere Strategien entwickelt, jeden Restart besser zu nutzen. Bspw. haben Das und Pratihar 2018 eine Restart-Strategie für RCGAs mit komplexen multimodalen Problemen entwickelt, welche den GA nach vier unterschiedlicher Kriterien immer wieder neustartet. Weitere Restart-Strategien können unter anderem in Dao et al. 2016 und Ghannadian et al. 1996 nachgeschlagen werden.

Restarts sind außerdem nicht das einzige Mittel, der Mutlimodalität von Problemen Herr zu werden: *Fitness Sharing*, *Niching* oder auch *Novelty Search* werden ebenfalls hierzu verwendet.

2.5.9 Parameter-Optimierung

Bei der Optimierung von GAs gilt das sogenannte *No-free-Lunch-Theorem*[47]. Es besagt, dass keine optimale Parameter-Konfiguration für alle Problemtypen existiert. Eine Konfiguration, die sich als optimal für ein Problem X bewährt hat, kann für ein Problem Y nutzlos sein (Kramer 2017, 22, 61).

Es wird zudem unterschieden, welche Art von Parametern optimiert werden sollen: Die *Parameterkontrolle* (engl. Parameter Control) beschäftigt sich mit Parametern, welche sich während des Durchlaufs eines GA ändern (bspw. von Generation zu Generation). Da hier viele Einstellungen und Verläufe beachtet werden müssen, stellt sich diese Aufgabe als wesentlich schwieriger heraus als die *Parametereinstellung* (engl. Parameter Tuning).

[47] Sinngemäß zu Deutsch: „Nichts ist umsonst“

Hierbei bleiben die Parameter des GA über den gesamten Durchlauf konstant (Eiben und Smit 2012).

Da viele Funktionen eines GAs auf Zufälligkeiten beruhen, ist es notwendig, das gleiche Problem mehrmals mit demselben GA zu lösen. Es haben sich hier einige stochastische Maße etabliert, welche die Leistung eines GAs jeweils aus verschiedenen Blickwinkeln betrachten (Eiben und Smit 2012):

- Durchschnittliche Fitness des besten Individuums eines Durchlaufs
- Durchschnittliche Anzahl an Fitness-Evaluationen
- Erfolgsrate
- Anzahl globals Optimum gefunden

Als Anhaltspunkt für die benötigte Anzahl der GA-Durchläufe, um ein aussagekräftiges stochastisches Maß zu gewinnen, hat sich die Zahl 50 bewährt (Deb 2000; Loshchilov 2013).

Die Parameter des GAs können entweder iterativ oder nicht-iterativ optimiert werden. Zu den iterativen Methoden gehört das manuelle Einstellen der Parameter durch einen Experten (*Expertenwissen*) oder die Verwendung eines *Meta-GAs*. Dabei handelt es sich um einen GA, welcher als zusätzliche Schleife außen um den eigentlichen GA platziert wird. Dabei repräsentieren die Gene des Meta-GA die Parameter-Einstellungen des zu optimierenden GA. Als nicht iterative Methoden können hier sowohl die *Grid Search* als auch die *Random Search* (siehe Abschnitt 2.3.3.3, S. 53) zum Einsatz kommen (Clune et al. 2005; Eiben und Smit 2012; Kramer 2017, S. 22).

Weiterhin muss bei Optimierung der Parameter eines GAs zwischen zwei Problem-Typen unterschieden werden: Zum einen die GAs, welche sehr oft zum Einsatz kommen und denen meist nur wenig Zeit zum Lösen des Optimierungsproblems zur Verfügung steht. Zum anderen die sogenannten *one-off Probleme.* Bei ihnen ist die Laufzeit eher zweitrangig, da sie dafür verwendet werden, für ein komplexes Problem einmalig möglichst das globale Optimum zu finden. Da dieses Problem nicht noch einmal gelöst werden muss, sind die optimalen GA-Parameter hier nicht wirklich von Relevanz (Eiben und Smit 2012).

3 Analyse von Wirkzusammenhängen

3.1 Vorgehensweise

Das Ziel dieses Abschnitts soll zum einen die Analyse von Wirkzusammenhängen und die Identifikation von maßgeblichen Kostentreibern innerhalb der betrachteten Kommissioniersysteme sein. Zum anderen sollen mögliche Gemeinsamkeiten und Unterschiede zwischen den einzelnen Kommissioniersystemen erforscht und aufgezeigt werden. Hierbei baut der zweite Punkt auf den ersten auf.

Die Methoden bzw. Werkzeuge, um dies leisten zu können, wurden ausführlich in Abschnitt 2.4 (S. 56) vorgestellt. Mit Hilfe der RI sollen zunächst die wichtigsten Kostentreiber der Kommissioniersysteme identifiziert und anschließend mit PDP und ICE tiefergehend untersucht werden. Der Vorteil dieser Methoden liegt darin, dass sie sehr viele Informationen komplexer Systeme bündeln und relativ leichtverständlich grafisch darstellen können.

Um diese Methoden jedoch (effizient) anwenden zu können, bedarf es einer Abbildung der Struktur der Kommissioniersysteme in Metamodelle. Für die Berechnung der RI ist ein Metamodell ohnehin zwingend notwendig, wohingegen PDP und ICE auch mit dem Excel-Tool direkt berechnet werden könnten. Allerdings sind hierzu sehr viele einzelne Berechnungen notwendig; dies würde im Excel-Tool schlicht zu viel Zeit in Anspruch nehmen. MLPs (Abschnitt 2.3.2 auf S. 35) sind sehr universell einsetzbar und recht schnell trainierbar. Zudem existieren für sie überhaupt Methoden, die bewerten können, wie wichtig ein Faktor für den Verlauf einer Zielgröße ist und ob dieser sie eher verstärkt oder sie verringert (RI).

Um Metamodelle überhaupt trainieren zu können, müssen Trainingsdaten vorhanden sein – am besten so viele Daten wie möglich, um ein möglichst genaues Modell trainieren zu können. Da es sich hier um ein Computer-Experiment handelt, stellt dies kein Problem dar. Bei Computer-Experimenten kommen für gewöhnlich gleichverteilte Versuchspläne (Abschnitt 2.2.2 auf S. 25) zum Einsatz, da sie den gesamten Lösungsraum abdecken und so die Struktur des Systems sehr gut darstellen können. Als gleichverteilter Versuchsplan

kommt hier die Sobol'-Sequenz zum Einsatz. Man könnte theoretisch hier auch LHD benutzen, allerdings ist das Testfeld hochdimensional und beinhaltet mehrere zehntausend Datenpunkte. Die Berechnung eines LHD ist hier von der Rechenzeit einfach nicht darstellbar. Außerdem ist es äußerst fraglich, ob dies überhaupt -im Vergleich zur Sobol'-Sequenz- einen wirklichen Mehrwehrt liefert. Dementsprechend sollen hier die Testfelder mit Hilfe der Sobol'-Sequenz erstellt werden (Ebert et al. 2015; Garud et al. 2017, S. 37; Siebertz et al. 2017, S. 207).

Um einen Versuchsplan überhaupt aufstellen zu können, müssen die Faktoren identifiziert und ausgewählt werden, welche die Zielgröße des Systems beeinflussen. Unter Umständen kann es notwendig sein, dass auch Faktoren erst modifiziert werden müssen, damit sie von dem Metamodell überhaupt verarbeitet werden können. Weiterhin kann es sein, dass Faktoren auf bestimmten Stufen festgelegt werden müssen, für den Fall, dass sie nicht modifiziert werden können.

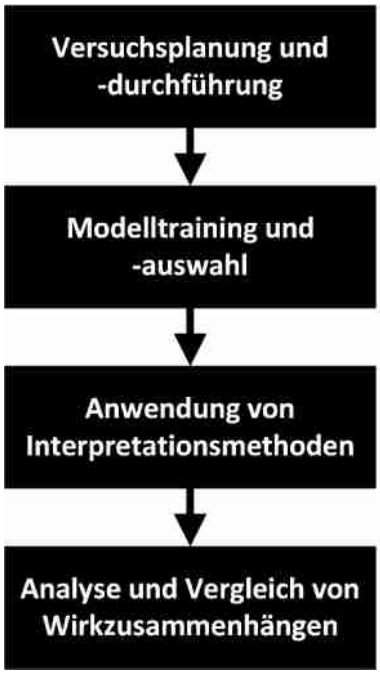


Abbildung 34: Vorgehensweise bei der Analyse von Wirkzusammenhängen

Die hier beschriebenen, notwendigen Schritte sollen rückwärts, gemäß Abbildung 34, durchschritten werden. Dabei werden sie für jedes Kommissioniersystem einzeln angewandt. Es sollte jedoch insbesondere bei der Auswahl der Faktoren darauf geachtet werden, dass eine Vergleichbarkeit zwischen den Systemen möglich ist.

Eine detaillierte Beschreibung der einzelnen Schritte soll für das gassenungebundene konventionelle Kommissionieren in Abschnitt 3.2 geschehen. Da sich viele dieser Schritte inhaltlich weitestgehend wiederholen, wird bei der gassengebundenen Betrachtung lediglich auf die Unterschiede eingegangen.

Zunächst jedoch ein kurzer Überblick über die Funktion des vorhandenen Excel-Tools:

Abbildung 35 zeigt das Tabellenblatt (TB) „Kommissioniersystem“ des Tools. Es stellt die Eingabemaske für den Benutzer dar, in dem das zu berechnende Kommissioniersystem konfiguriert werden kann. Hinter den orangenen Feldern befinden sich Dropdown-Einträge, aus denen Komponenten oder Einstellungen für die Funktion des Kommissioniersystems ausgewählt werden können. Einige dieser Einträge können nur in Kombination mit gewissen Einträgen anderer Felder ausgewählt werden. Die grünen Felder zeigen an, ob die ausgewählte Kombination zulässig ist. Die gelb und weiß hinterlegten Felder beinhalten numerische Einträge, die vom Nutzer angepasst werden können.

Die Berechnungsergebnisse des Excel-Tools für ein Kommissioniersystem sind in TB „Ergebnis-Zusammenfassung“ zu finden (Abbildung 36). Sie sind allesamt kontinuierliche Größen.

Es sei an dieser Stelle erwähnt, dass Abbildung 35 und Abbildung 36 jeweils nur die für diese Arbeit relevanten Bereiche der TB zeigen. Die kompletten TB können im Datenanhang gefunden werden: „03_Analyse_von_Wirkzusammenhängen \ 01_Konventionelles_ Kommissionieren_gassenungebunden \ 01_Versuchsplanung \ Tool_Output \ 2018-09-13 Berechnungstool_Konv_gassenungebunden.xlsm“

Kategorie	Parameter	Einheit	Wert
Systemtyp und Auftragsdaten			
Kommissionier-Systemtyp	PzW-Sytemtyp		Konventionelles Kommissionieren
Auftragsdaten	ø Anzahl Aufträge pro Tag	Aufträge/d	4025
	ø Anzahl Positionen pro Auftrag	Pos/Auftrag	
	ø Anzahl Picks pro Position	Picks/Pos	8
	Prüfung: Anzahl Positionen pro Auftrag möglich mit FM		möglich
	Verwendete durchschnittliche Anzahl Positionen pro Auftrag	Pos/Auftrag	5,5
Lager 1			
	Lager-Layout		Kopfganglayout
	Art der Betrachtung		gassenungebunden
	Verteilung der Artikel im Lager		exponentialverteilt
	Art der Ansteuerung		ungeordnet
	Geschwindigkeitsveränderung bei Gassenwechsel		nein
	Wegstrategie		Schleifenstrategie ohne Überspringen
	Kompatibilität Wegstrategie mit Art der Betrachtung		Kombination möglich
	ø Artikelgewicht	kg	1
	ø Anzahl Artikel pro LHM	Artikel/LHM	10
	ø Anzahl LHM pro Position	LHM/Position	1
	Anzahl Kommissionierer im System	MA	1
	Anzahl Gassen im Lager	Gassen	20
	Breite einer Gasse	m	2,35
	Konstantes Anfahrmaß	m	1,00
	Faktor Hin-/Rückfahrt		0,124
	Faktor Zwischenfahrten		0,08
	Parameter der Exponentialverteilung		8,038
	Kommissionieren mit/ohne Basis		mit Basis
	Lage der Basis		dezentral
	Zonendurchgang		beidseitig
	Weg Auftragsannahme - Behälterbereitstellung	m	2,00
	Weg Behälterbereitstellung - Gassenanfang	m	3,00
	Weg Gassenanfang - Abgabeort	m	3,00
	Weg Abgabeort - Auftragsannahme	m	2,00
	Weg Gassenanfang - Auftragsannahme	m	3,00
	Weg Basis - Gassenanfang	m	5,00
LM und LHM	LM 1		Fachbodenregal
	LHM 1		Großer KLT
	Prüfung Kombination von LM 1 mit LHM 1		Kombination möglich
FM	FM 1		Niederhubwagen
	Anzahl Fördermittel im System	Stk	43,9
	Prüfung Kombination FM 1 mit LM 1		Kombination möglich
	Prüfung Kombination von FM 1 mit LHM 1		Kombination möglich
ITK	ITK 1		Pickliste
	Prüfung Kombination von ITK mit PzW-Systemtyp		Kombination möglich

Abbildung 35: Ausschnitt aus TB "Kommissioniersystem"

Kategorie	Parameter	Einheit	Wert
Leistungsdaten System 1	Pickleistung	Pick/h	251,91
	Kommissionierleistung	Pos/h	31,49
	Kommissionierzeit pro Pos	s/Pos	114,33
	Zusammenführungszeit pro Pos	s/Pos	92,83
	Wegzeit pro Auftrag	s/Auftrag	474,00
	Wegstrecke pro Auftrag	m/Auftrag	577,30
	Gassenweg pro Auftrag	m/Auftrag	510,54
	Systemauslastung	%	4.393,91
	Anzahl Kommissionierer im System		1
	Empfohlene Anzahl Kommissionierer		**43,94**
	Bewegtes Gewicht pro Stunde	kg/h	393,61
Investitions-kosten	Lagermittel (LM)	€	775.000,00
	Fördermittel (FM) unstetig	€	87.878,17
	Fördermittel (FM) stetig	€	0,00
	Ladehilfsmittel (LHM)	€	170.500,00
	Informationstechnische Kommissioniererführung (ITK)	€	879.100,00
	Gesamt	**€**	**1.912.478,17**
Betriebskosten	Abschreibung	€/Jahr	142.775,21
	Kalkulatorische Zinsen	€/Jahr	57.374,35
	Wartung	€/Jahr	95.623,91
	Energie	€/Jahr	688.492,14
	Kalkulatorische Miete	€/Jahr	57.027,60
	Lohnkosten	€/Jahr	4.393.910,00
	Gesamt	**€/Jahr**	**5.435.203,21**
Flächenbedarf	Gesamt	m^2	1.584,10
TCO	über 10 Jahre	**€/10 Jahre**	**56.264.494,42**

Abbildung 36: Ausschnitt aus TB "Ergebnis-Zusammenfassung"

3.2 Gassenungebundenes konventionelles Kommissionieren

Zunächst soll nur die gassenungebundene (gug.) Betrachtung des konventionellen Kommissionierens untersucht werden. Der in diesem Abschnitt verwendete Python-Code kann im Datenanhang unter „03_Analyse_von_Wirkzusammenhängen\01_Konventionelles_Kommissionieren_gassenungebunden\02_Modell\MA_Model_Interpret_gug_FINAL.py“ in voller Länge gefunden werden.

3.2.1 Versuchsplanung

3.2.1.1 Auswahl von Zielgrößen

Im ersten Schritt der Versuchsplanung soll gesichtet werden, welche potentiellen Zielgrößen das Excel-Tool überhaupt liefern kann. Anschließend sollen die Relevanten für die weitere Bearbeitung ausgewählt werden.

Alle möglichen Zielgrößen sind in TB „Ergebnis-Zusammenfassung“ (siehe Abbildung 36) zu finden. Die Leistungsdaten (abgesehen von der Anzahl der Kommissionierer) sind hier als zu untersuchende Zielgröße uninteressant, da sie nur die durch die Anzahl der Aufträge und Positionen/Auftrag geforderten Leistungsdaten aus TB „Kommissioniersystem“ (Abbildung 35) wiederspiegeln.

Interessant sind hier jedoch die in Abbildung 36 orange hinterlegten Felder. Namentlich: *Investitionskosten*, *jährliche Betriebskosten*, *Anzahl der benötigten Kommissionierer* und die *Total Cost of Ownership* über 10 Jahre (TCO). Die TCO umfasst zum einen die Investitionskosten des Kommissioniersystems sowie sämtliche anfallenden Betriebskosten über den Lebenszyklus des Systems. Dieser wird hier mit 10 Jahren angenommen. Weiterhin können zudem noch mögliche Entsorgungskosten des Systems mit in den TCO einberechnet werden, was hier jedoch vernachlässigt wird (Fraunhofer-Gesellschaft 2018).

Demnach berechnet sich der TCO im Excel-Tool aus der folgenden Gleichung:

$$\mathrm{TCO} = \mathrm{Investitionskosten} + 10 \cdot \mathrm{Betriebskosten}$$

Die vier hier betrachteten Zielgrößen (sowie deren Kürzel) lauten:

- Investitionskosten (y1_invest), in Euro (€)
- Betriebskosten (y2_betrieb), in Euro (€)
- Anzahl der benötigten Kommissionierer (y3_anz_kommi)
- TCO (y4_tco), in Euro (€)

3.2.1.2 Auswahl und Modifizierung von Faktoren

Die Auswahl der Faktoren gestaltet sich im Gegensatz zu den Zielgrößen schon schwieriger: Dazu werden zunächst die entscheidenden Einflussgrößen nach deren Typ (quantitativ / qualitativ) eingeteilt (siehe Tabelle 2). Wie man dort sieht, sind nicht alle möglichen

Faktoren des TB „Kommissioniersystem“ abgetragen. Bei den vernachlässigten Faktoren handelt es sich um Konstanten und Größen, welche Berechnungen im Inneren des Tools beeinflussen, allerdings auf Standardwerte aus der Literatur festgelegt sind (bspw. „Parameter der Exponentialverteilung“ ist nach Hompel et al. 2011, S. 150 auf den Wert $8{,}038$ festlegt). Andere Faktoren, wie das Artikelgewicht, haben einfach keinen Einfluss auf die ausgewählten Zielgrößen und können deshalb ignoriert werden.

Tabelle 2: Faktoren nach Typ

	Quantitativ (kontinuierlich)	**Quantitativ (ganzzahlig)**	**Qualitativ (kategorisch)**
1	ø Anzahl Aufträge pro Tag	Anzahl Gassen	Lager-Layout
2	ø Anzahl Picks pro Position	-	Art der Betrachtung
3	ø Anzahl Positionen pro Auftrag	-	Verteilung der Artikel im Lager
4	Breite einer Gasse	-	Art der Ansteuerung
5	-	-	Geschwindigkeitsveränderung bei Gassenwechsel
6	-	-	Wegstrategie
7	-	-	Kommissionieren mit/ohne Basis
8	-	-	Lage der Basis
9	-	-	Zonendurchgang
10	-	-	LM
11	-	-	LHM
12	-	-	FM
13	-	-	ITK

Bei der Betrachtung von Tabelle 2 fällt auf, dass die Anzahl der kategorischen Variablen sehr hoch ist. Dies ist problematisch für den Einsatz von Metamodellen, da diese auf die Verarbeitung und Vorhersage von kontinuierlichen Größen ausgelegt sind. Kategorische Variablen lassen sich jedoch weder zählen noch messen. Es existieren zwar Methoden, die versuchen, kategorische Variablen zu ordnen, um ihnen anschließend auf Basis dieser Ordnung Zahlenwerte zuzuordnen, wie bspw. *1-to-N Binary Coding*, *Frequency Based Encoding (FBE)* oder *Thermometer Encoding*, allerdings ist insbesondere bei Wegstrategien und Lager-Layouts davon auszugehen, dass diese maßgeblich interne Berechnungen des

Excel-Tools verändern. Ein Metamodell würde diese massiven Änderungen der Rechenwege, von der einen Faktorstufe zur anderen, nicht richtig deuten können und somit sehr schlechte Vorhersageergebnisse liefern und zu sehr starkem Overfitting neigen (Hastie et al. 2017, S. 310; Siebertz et al. 2017, S. 156; Wang et al. 2008).

Möchte man jedoch alle Kombinationen der kategorischen Variablen durch Metamodelle abbilden, müsste für jede dieser Kombinationen ein eigenes Metamodell trainiert werden. Im vorliegenden Fall müssten somit 154.560 unterschiedliche Metamodelle[48], für die verbleibenden kontinuierlichen Faktoren, trainiert werden (Siebertz et al. 2017, S. 158).

Es kann jedoch sein, dass sich hinter diesen kategorischen Faktoren in Wirklichkeit kontinuierliche Faktoren befinden. Mit dem sogenannten *Morphing* werden kategorische Faktoren in numerische Faktoren „umgewandelt“. Dazu muss das betrachtete System genauer untersucht werden. Gelingt das Morphing für alle kategorischen Variablen, können alle Versuchspläne und Metamodelle für das Problem eingesetzt werden. Zudem steht so der gesamte Faktorraum zur möglichen Optimierung zur Verfügung (Siebertz et al. 2017, S. 156–157).

Betrachtet man die Faktoren LM, LHM, FM und ITK genauer, stellt man fest, dass diese lediglich auf Voreinstellungen verschiedener numerischer Faktoren im TB „Technik“ verweisen.

Alle Lagermittel (LM) sind durch dieselben numerischen Faktoren beschrieben. Als Beispiel hierfür soll das Fachbodenregal in Abbildung 37 dienen. Eine Unterscheidung zwischen den einzelnen Lagermitteln findet lediglich auf Grund von Änderungen in den fett und rot geschriebenen Faktoren statt. Dementsprechend werden diese als relevante numerische Faktoren in den Versuchsplan mit aufgenommen. Alle anderen Faktoren des LM werden aus eben diesen Faktoren berechnet oder haben sich im Vorfeld dieser Arbeit für die Berechnung der gewählten Zielgrößen als irrelevant herausgestellt.

[48] Das ist die Anzahl der hier möglichen Faktorstufenkombinationen aus den kategorischen Faktoren

Kategorie	Parameter	Einheit	Wert	Kommentar	Zusätzliche	Kommentar
Lagermittel (LM)						
Fachbodenregal						
	LM-Länge (= Länge einer Gasse)	mm	28.000	[Anz. Regaleleinheiten pro Gassenarm (einseitig)] * [Länge Regaleinheit]	Anz. Regaleleinheiten pro Gasse (einseitig)	20
	LM-Breite = Fachtiefe	mm	650		Anz. Regaleleinheiten pro Gassenarm (einseitig)	20 [Anz. Regaleleinheiten pro Gasse (einseitig)] * [2 (Kopfgang) bzw. 1 (Zentralgang)]
	LM-Höhe	mm	2.000		Länge Regaleinheit [mm]	1.400
	Fachbreite	mm	467		Anz. LHM pro Fachbodeneinheit	3
	Fachhöhe	mm	400		Kosten pro Regaleinheit	200,00 €
	Regalbau Anzahl Ebenen pro Regaleinheit	Ebenen/LM	5,00			
	Kosten pro LM	€/LM	4.000,00	[Kosten pro Regaleinheit] * [Anz. Regaleleinheiten pro Gassenarm (einseitig)]		
	Anzahl Fächer pro LM	Fächer/LM	300	[Anz. LHM pro Fachbodeneinheit] * [Ebenen/LM] * [Anz. Regaleleinheiten pro Gassenarm (einseitig)]		
	Anzahl LM/Regaleinheiten im Lager	Anz. LM ges.	24	[Anzahl Gassen] * [2 (Kopfgang) bzw. 4 (Zentralgang)]		
Ladehilfsmittel (LHM)						
	Großer KLT Eignung	LHM	ja	[Fächer/LM] * [Anzahl LM/Regaleinheiten im Lager]		
	Kapazität		7.200			

Abbildung 37: Voreinstellungen für LM „Fachbodenregal“ im TB „Technik“

Excel bietet die Funktionen „Spur zum Nachfolger“ bzw. „Spur zum Vorgänger“ an. Diese sind hier sehr hilfreich und sollten definitiv verwendet werden, um eventuell unnötige Faktoren außenvorzulassen. Ein Beispiel dafür ist hier der Faktor „Gassenbreite“ in TB „Kommissioniersystem“ (Abbildung 35) und der Faktor „Fachtiefe“ in TB „Technik“ (Abbildung 37). Bei Tests hat sich gezeigt, dass beide Faktoren das Ergebnis des Tools beeinflussen. Wenn man hier jedoch die Funktion „Spur zum Nachfolger“ verwendet, zeigt sich, dass der entscheidende Faktor für die Ergebnisänderung in TB „Systemkonfiguration“ zu finden ist: „Breite eines Moduls“ (Abbildung 38).

Kategorie	Parameter	Einheit	Wert	Kommentar
System 1				
Lagerdaten 1	Gassenbreite	m	1,50	Eingabewert aus TB "Kommissioniersystem"
	Konstantes Anfahrmaß	m	1,00	Eingabewert aus TB "Kommissioniersystem"
	Abstand zwischen erster und letzter Gasse	m	64,40	(Gassen im Lager - 1) * Gassenbreite
	Gasseneinfahrbreite	m	1,50	Gassenbreite - 2 * LM-Breite
	Gassenlänge	m	14,00	LM-Länge = Gassenlänge --> Eingabewert aus TB "Technik"
	Breite eines Moduls (2 Regale + Gasse)	m	**2,80**	2 * LM-Tiefe + Gassenbreite
	Breite des Lagers	m	67,20	
	Länge des Lagers	m	14,00	
	Flächenbedarf	m2	940,80	

Abbildung 38: TB "Systemkonfiguration"

Wie man erkennen kann, berechnet sich dieser Faktor aus einer Addition der zwei genannten Faktoren durch:

$$\text{Modulbreite} = 2 \cdot \text{LM-Fachtiefe} + \text{Gassenbreite}$$

Um Faktoren einzusparen, kann man hier auch die Fachtiefe und die Gassenbreite durch den Faktor Modulbreite substituieren. Es muss allerdings darauf geachtet werden, dass die substituierten Faktoren keinen Einfluss auf andere Parameter des Systems haben, welche berücksichtigt werden sollten. Dies ist hier der Fall. Der einzige Zweck der LM-Fachtiefe und Gassenbreite besteht darin, die Modulbreite zu berechnen. Zur späteren Schätzung durch ein Metamodell müsste man die hier substituierte Rechnung vor den Faktor der Modulbreite schalten.

Bei den Fördermitteln (FM) sieht es ähnlich aus (siehe Abbildung 39). Allerdings beschreiben die Faktoren das jeweilige FM wesentlich direkter. Die hier relevanten Faktoren sind ebenfalls in fettem Rot markiert. Die Kapazität des FM (für ein gewisses LHM)

beeinflusst ebenfalls die Ergebnisse des Excel-Tools, allerdings wird diese nicht direkt hier betrachtet, sondern (auf Grund einiger Rechenvorschriften im Tool) ist in dem Faktor „Verwendete durchschnittliche Anzahl Positionen pro Auftrag“ im TB „Kommissioniersystem“ enthalten. Es wird dort das Minimum aus der Kapazität des FM und der „Anzahl Positionen pro Auftrag“ verwendet. Dieser Vergleich wurde ebenfalls entfernt, da ein solcher Wenn-Dann-Vergleich ein Metamodell irritieren würde und zu schlechten Schätzungen führen kann. Bei der späteren Verwendung des Modells müsste diese Abfrage wieder vorgeschaltet werden, wie auch bei der Modulbreite des Lagers.

Kategorie	Parameter	Einheit	Wert
Fördermittel (FM)			
Niederhubwagen			
	Förderebene		eindimensional
	Bedienung		automatisch
	Förderbewegung		unstetig
	FM-Breite	mm	550
	Max. horizontale Beschleunigung	m/s²	**0,80**
	Max. horizontale Geschwindigkeit	m/s	**1,30**
	Max. vertikale Beschleunigung	m/s²	
	Max. vertikale Geschwindigkeit	m/s	
	FM-Kosten	€/Fzg	**4.000,00**
Ladehilfsmittel (LHM)			
Großer KLT	Eignung		ja
	Kapazität	LHM	**12**

Abbildung 39: Voreinstellungen für FM „Niederhubwagen“ im TB „Technik“

Der einzig relevante Faktor der Ladehilfsmittel (LHM) sind die Stückkosten (siehe Abbildung 40). Sie unterscheiden sich ebenfalls durch ihre Maße und Gewicht, allerdings spielt auch dieses hier keine Rolle für die untersuchten Zielgrößen.

Kategorie	Parameter	Einheit	Wert
Ladehilfsmittel (LHM)			
Großer KLT			
Maße	Länge	mm	600
	Breite	mm	400
	Höhe	mm	320
	Eigengewicht	kg/LHM	2,50
	Stückkosten	€/LHM	20,00

Abbildung 40: Voreinstellungen für LHM „Großer KLT“ im TB „Technik“

Letztlich noch die Informationstechnische Kommissioniererführung (ITK) (Abbildung 41): Die Gesamtbearbeitungszeit ist die Summe der Basis-, Greif- und Totzeit, wobei diese drei Zeiten außer zur Bildung der Gesamtzeit des jeweiligen ITK keine Verwendung finden. Folglich muss nur die Gesamtbearbeitungszeit als Faktor in den Versuchsplan einfließen. Weiterhin sind die drei markierten Kostenstellen für das Excel-Tool von Relevanz.

Kategorie	Parameter	Einheit	Wert
Informationstechnische Kommissioniererführung (ITK)			
Pickliste			
Bearbeitungszeit	Basiszeit	s/Pos	20,0
	Greifzeit	s/Pos	7,0
	Totzeit	s/Pos	10,0
	Bearbeitungszeit gesamt	**s/Pos**	**37,0**
Ausrüstungskosten	Kosten pro Kommissionierer	€/MA	200,00
	Kosten pro Lagerfach	€/Lagerfach	0,00
	Sonstiges	€	3.000,00

Abbildung 41: Voreinstellungen für ITK „Pickliste“ im TB „Technik“

Viele der hier markierten Faktoren sind im normalen Gebrauch des Tools ganzzahlig. Allerdings ist das Ziel des Versuchsplans, möglichst viele Datenpunkte im Lösungsraum zu verteilen. Um dies zu bewerkstelligen, müssen die ganzzahligen Faktoren auch als kontinuierliche Faktoren behandeln werden. Dies stellt hier auch kein Problem dar, da Vorversuche ergeben haben, dass das Tool auch mit kontinuierlichen Größen, wie bspw. 3,59 Gassen, rechnen und ein Ergebnis produzieren kann. Dies macht im normalen Gebrauch des Tools natürlich keinen Sinn, allerdings können so mehr Datenpunkte an unterschiedlichen Orten der jeweiligen Dimension untergebracht werden. Somit liefert

jeder Datenpunkt des Versuchsplans mehr Informationen über die Struktur innerhalb des Excel-Tools als eine Häufung von Punkten an nur wenigen ganzzahligen Werten.

Es wurden nun durch Morphing 18 kontinuierliche (bzw. ganzzahlige, als kontinuierliche Faktoren gehandhabte) Faktoren als relevant identifiziert. Im folgenden Schritt müssen für diese sinnvolle Spannweiten, also minimale und maximale Werte, für den Versuchsplan definiert werden. Die Spannweiten sollten dabei so gewählt werden, dass die meisten realen Kommissioniersysteme von dem Metamodell abgebildet werden können. Eine Extrapolation des Metamodells über die Grenzen des Versuchsplans hinaus ist nicht zulässig, da es passieren kann, dass sich dort Unstetigkeiten im Systemverhalten hervortun könnten. Je weiter man allerdings die Spannweite wählt, desto ungenauer wird das Modell bzgl. der absoluten Schätzungsfehler. Dementsprechend muss zwischen der Genauigkeit des Metamodells und der Größe des vom Modell erfassten Lösungsraums abgewogen werden (Siebertz et al. 2017, S. 24).

Um die Spannweiten der Faktoren festzulegen, wurde auf Expertenwissen von Prof. Ulrich Stache zurückgegriffen. Siehe dazu Tabelle 3.

Für die übrigen kategorischen Variablen hat sich keine Möglichkeit ergeben, diese zu modifizieren. Um lediglich ein Metamodell pro Zielgröße trainieren zu müssen, sollen diese nun auf sinnvolle Einstellungen festgelegt werden. Es soll geprüft werden, ob es Einstellungen dieser Faktoren gibt, die generell gegenüber den anderen zu bevorzugen sind. Faktoreinstellungen, die Kommissioniersysteme liefern, welche bei gleicher Leistung teurer sind als andere, werden in der Realität sehr wahrscheinlich nicht genutzt. Dementsprechend sollte die kostengünstigste Variante bzgl. des TCO die beste Aussagekraft über die Wirkzusammenhänge innerhalb der konventionellen Kommissioniersysteme liefern können. Der TCO wird deshalb hier als Maßstab verwendet, da er die übrigen drei Zielgrößen beinhaltet.

Tabelle 3: Spannweiten der quantitativen Faktoren

Kategorie	Allgemeines			Gassen			LM			
Bezeichnung	Durchschnittliche Aufträge pro Tag	verwendete Durchschnittliche Positionen pro Auftrag	Picks pro Position	Anzahl Gassen im Lager	Breite eines Moduls [m]	Anzahl LM nebeneinander	LM Fachbreite [mm]	Anzahl Ebenen pro LM [Ebenen/LM]	Kosten pro LM/Regalei nheit [€/LM]	LM Kapazität [LHM/LM Fach]
Variable	alg_auftrag_tag x1	alg_max_pos_auftrag x2	alg_pick_pos x3	alg_anz_gassen x4	alg_breite_modul x5	lm_anz_reihe x6	lm_fachbreite x7	lm_ebenen x8	lm_kosten x9	lm_fach_kapazitaet x10
Min	50,00	1,00	1,00	1,00	1,60	1,00	800,00	1,00	0,00	1,00
Max	8.000,00	10,00	15,00	40,00	6,00	30,00	2.000,00	6,00	2.500,00	10,00
Max - Min	7.950,00	9,00	14,00	39,00	4,40	29,00	1.200,00	5,00	2.500,00	9,00

Kategorie	FM		LHM	ITK				
Bezeichnung	Max horizontale Beschleunigung [m/s^2]	Max horizontale Geschwindigkeit [m/s]	FM-Kosten [€]	Stückkosten [€/LHM]	Bearbeitungszeit gesamt [s/Pos]	Kosten pro Kommissionierer [€/MA]	Kosten pro Lagerfach [€/LF]	Sonstiges [€]
Variable	fm_max_a x11	fm_max_v x12	fm_kosten x13	lhm_kosten x14	itk_bearb_zeit x15	itk_kosten_kommi x16	itk_kosten_fach x17	itk_sonst x18
Min	0,25	0,50	0,00	0,00	3,00	0,00	0,00	3.000,00
Max	1,50	1,70	4.000,00	20,00	40,00	200,00	100,00	50.000,00
Max - Min	1,25	1,20	4.000,00	20,00	37,00	200,00	100,00	47.000,00

Hier können die in Abschnitt 2.2.1 (S. 22 ff.) beschriebenen Effektdiagramme zum Einsatz kommen. Dazu soll ein Vollfaktorplan über die übrigen 10 kategorischen Variablen angefertigt werden. Ein Vollfaktorplan ist deshalb hier sinnvoll, da bei der gassenungebundenen Betrachtung lediglich 320 Varianten zu testen sind, was in einem Computer-Experiment nicht allzu lange dauert. Zum anderen sollen alle Kombinationen der Faktorstufen getestet werden, da davon auszugehen ist, dass sich die Ergebnisse, auf Grund von Änderungen in den Rechenvorschriften, voneinander stark unterscheiden (Siebertz et al. 2017, S. 30).

Die bereits identifizierten 18 kontinuierlichen Größen sollen dazu, der Einfachheit halber, auf die Mitte ihrer Spannweite gesetzt werden. Es könnte natürlich sein, dass manche Faktorstufen bei sehr kleinen oder sehr großen Kommissioniersystemen kostengünstiger sind als bei mittleren Systemen. Würde man hier allerdings zusätzlich noch über jeden der 18 kontinuierlichen Faktoren auf jeweils nur zwei Faktorstufen Versuche fahren, umfasst der Versuchsplan $320 \cdot 2^{18} \approx 8{,}4 \cdot 10^{7}$ Einträge. Da es sich hierbei nur um einen Vorversuch zum eigentlichen Versuchsplan handelt, wäre dieser Aufwand viel zu groß. Ein mittel großes Kommissioniersystem sollte hier eine gute Basis zur Einstellung der kategorischen Faktoren bieten.

Die Faktoren „Lager-Layout“, „Verteilung der Artikel im Lager“ und die „Art der Ansteuerung“ sind jeweils nur auf zwei Faktorstufen zu testen.

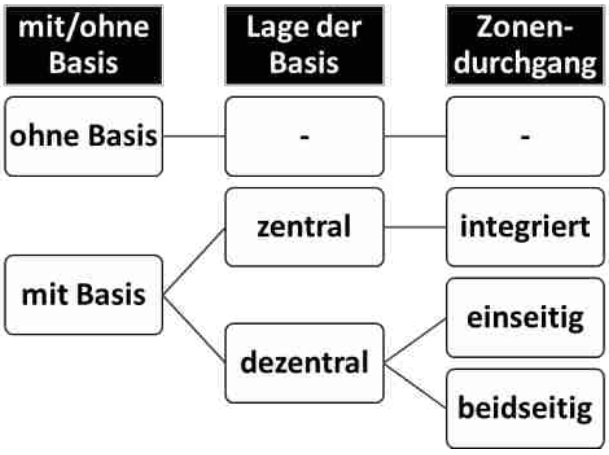


Abbildung 42: Abhängigkeiten bei Basis

Beim Kommissionieren mit bzw. ohne Basis sind allerdings einige Abhängigkeiten zwischen den beteiligten Faktoren zu beachten (siehe Abbildung 42). Man könnte nun für jeden Faktor mit den jeweiligen Faktorstufen ein zusätzliches Effektdiagramm anfertigen, allerdings könnte man auch die vier vorhandenen Kombinationsmöglichkeiten mit den drei beteiligten Faktorstufen zu einer Gesamtfaktorstufe zusammenfassen, z.B. „mit Basis / dezentral / einseitig“. All diese Gesamtfaktorstufen werden gleich oft im Vollfaktorplan getestet, was eine Vergleichbarkeit sichert. Allerdings spart man so Effektdiagramme ein, die auf Grund der vorhandenen Abhängigkeiten keine wirkliche Aussagekraft hätten.

Bei der Betrachtung der Wegstrategie kann dieses Verfahren ebenfalls angewendet werden. Hier sind jedoch für die gug. Betrachtung zehn Gesamtfaktorstufen zu beachten (siehe Abbildung 43), bei der gg. sind es zwei.

Es werden demnach genau drei Effektdiagramme benötigt, die aus einem Vollfaktorplan mit $2 \cdot 2 \cdot 2 \cdot 4 \cdot 10 = 320$ Versuchen berechnet werden. Da hier z.T. mehr als zwei Faktorstufen betrachtet werden, müsste zur Berechnung der Effekte (Wechsel von der einen Faktorstufe zur anderen) eine Effektmatrix aufgestellt werden, wobei hier eigentlich nur die Unterschiede im Mittelwert des TCO über die einzelnen Faktorstufen als Maß interessant sind.

Wie oben beschrieben werden die übrigen 18 numerischen Faktoren auf ihre mittleren Werte festgesetzt. Der Vollfaktorplan sowie die Ergebnisse können im Detail im Datenanhang gefunden werden: „03_Analyse_von_Wirkzusammenhängen\01_Konventionelles_Kommissionieren_gassenungebunden\01_Versuchsplanung\Tool_Input\Vorversuch“.

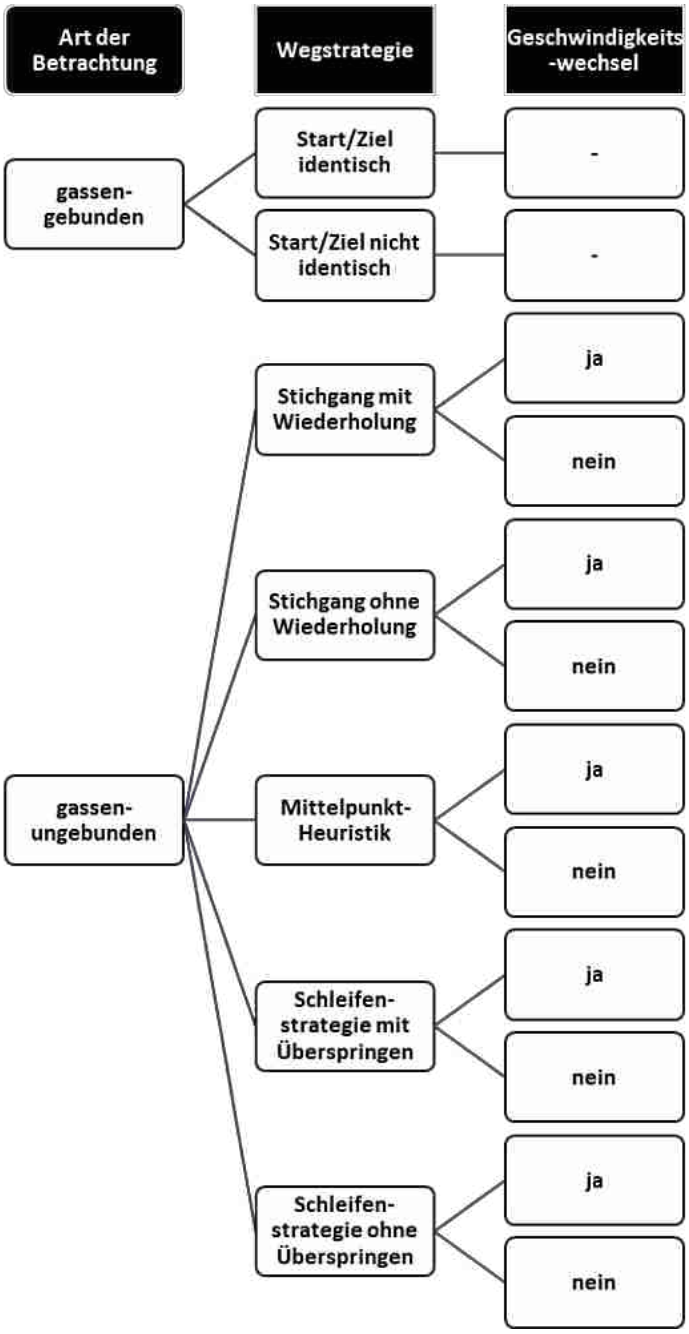


Abbildung 43: Abhängigkeiten bei Wegstrategie

Tabelle 4 zeigt die durchschnittlichen Werte der vier Zielgrößen aus den Versuchen des Vollfaktorplans für die gassenungebundene Betrachtung. Interessant ist zunächst nur der Mittelwert des TCO mit $\bar{y}_4 = 3{,}72 \cdot 10^7$.

Tabelle 4: Durchschnittswerte der Zielgrößen des Vollfaktorplans (gassenungebunden)

Investitionskosten	Betriebskosten	Anzahl Kommissionierer	TCO
$\bar{y}_1 = 1{,}57 \cdot 10^6$	$\bar{y}_2 = 3{,}57 \cdot 10^6$	$\bar{y}_3 = 27{,}0$	$\bar{y}_4 = 3{,}72 \cdot 10^7$

Abbildung 44, Abbildung 45 und Abbildung 46 zeigen die berechneten Effektdiagramme für die betrachteten kategorischen Faktoren. Sie wurden in MATLAB mit der Funktion `maineffectsplot()` erstellt[49].

Zur Erinnerung: Auf der y-Achse ist der Durchschnittliche TCO abgetragen, welcher sich über die Faktorstufen auf der x-Achse einstellt. Er wird über alle Versuche des Vollfaktorplans berechnet, in denen die jeweilige Faktorstufe enthalten ist. In allen Versuchen des Vollfaktorplans ist immer eine der Faktorstufen enthalten, wobei alle Faktorstufen eines Diagramms gleich oft vorkommen. Die gestrichelte, schwarze Linie zeigt den durchschnittlichen TCO über alle Versuche an und ist somit ein Anhaltspunkt dafür, wie gut die jeweilige Faktorstufe ist.

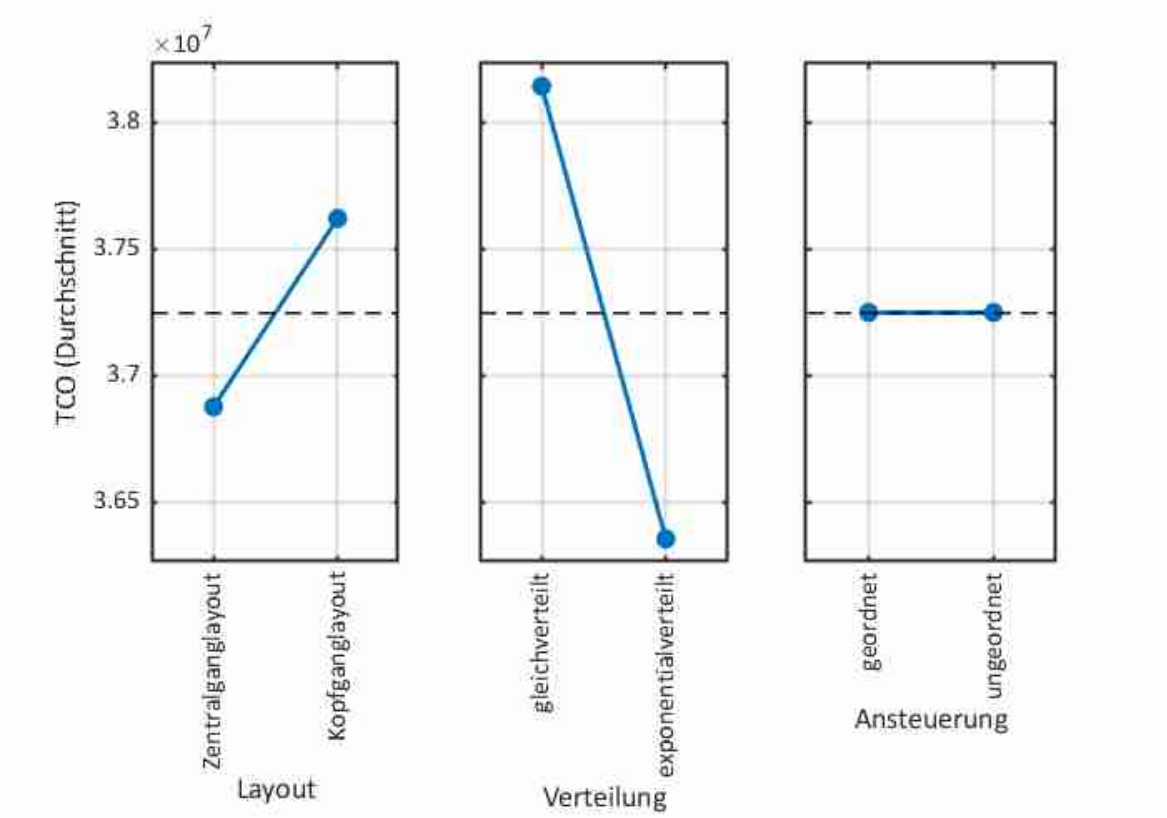


Abbildung 44: Effektdiagramme für Layout, Verteilung und Ansteuerung

Die Effektdiagramme für die Faktoren „Lager-Layout“, „Verteilung der Artikel im Lager“ und „Art der Ansteuerung“ sind in Abbildung 44 zu sehen. Es zeigt sich hier, dass das

[49] Eine Dokumentation zu dieser Funktion kann online unter https://www.mathworks.com/help/stats/maineffectsplot.html gefunden werden.

Zentralganglayout im Schnitt um ca. 800.000 Euro günstiger ist als das Kopfganglayout. Dementsprechend sollte hier die Faktorstufe „Zentralganglayout“ in den folgenden Versuchen verwendet werden. Bei der „Verteilung der Artikel im Lager“ zeigt sich, dass „exponentialverteilt“ wesentlich kostengünstiger ist als „gleichverteilt“. Bei Art der Ansteuerung ist offensichtlich gleichgültig, welche der beiden Faktorstufen eingestellt wird.

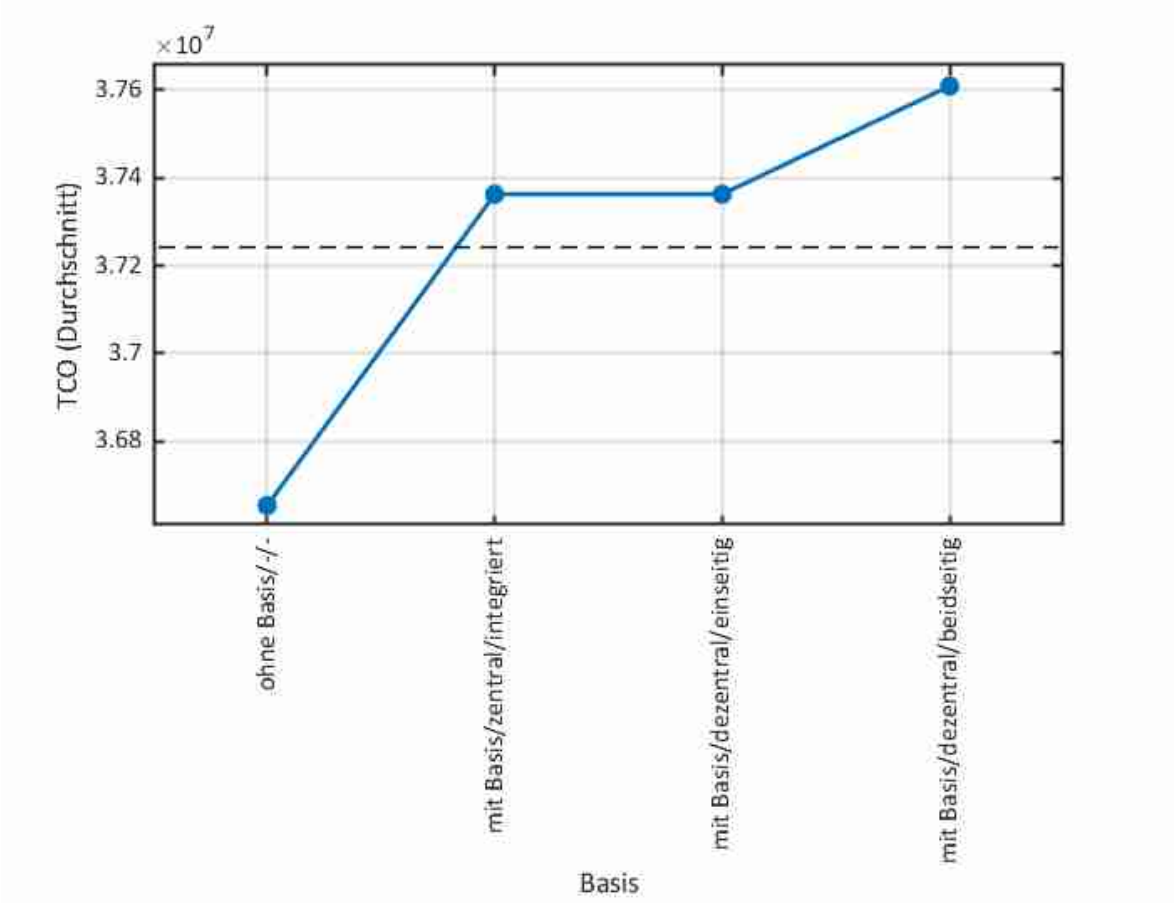


Abbildung 45: Effektdiagramm für Basis

Das betrachtete Kommissioniersystem sollte außerdem möglichst keine Basis aufweisen, wie man anhand von Abbildung 45 erkennen kann.

Bei den Wegstrategien (Abbildung 46) zeigt sich, dass die „Schleifenstrategie ohne Überspringen“ durchschnittlich einen fast doppelt so hohen TCO erzeugt als alle anderen Wegstrategien. Die übrigen Wegstrategien sind in etwa auf gleichem Niveau, wobei der „Stichgang ohne Wiederholung“ (und ohne Geschwindigkeitsveränderung) den niedrigsten TCO produziert.

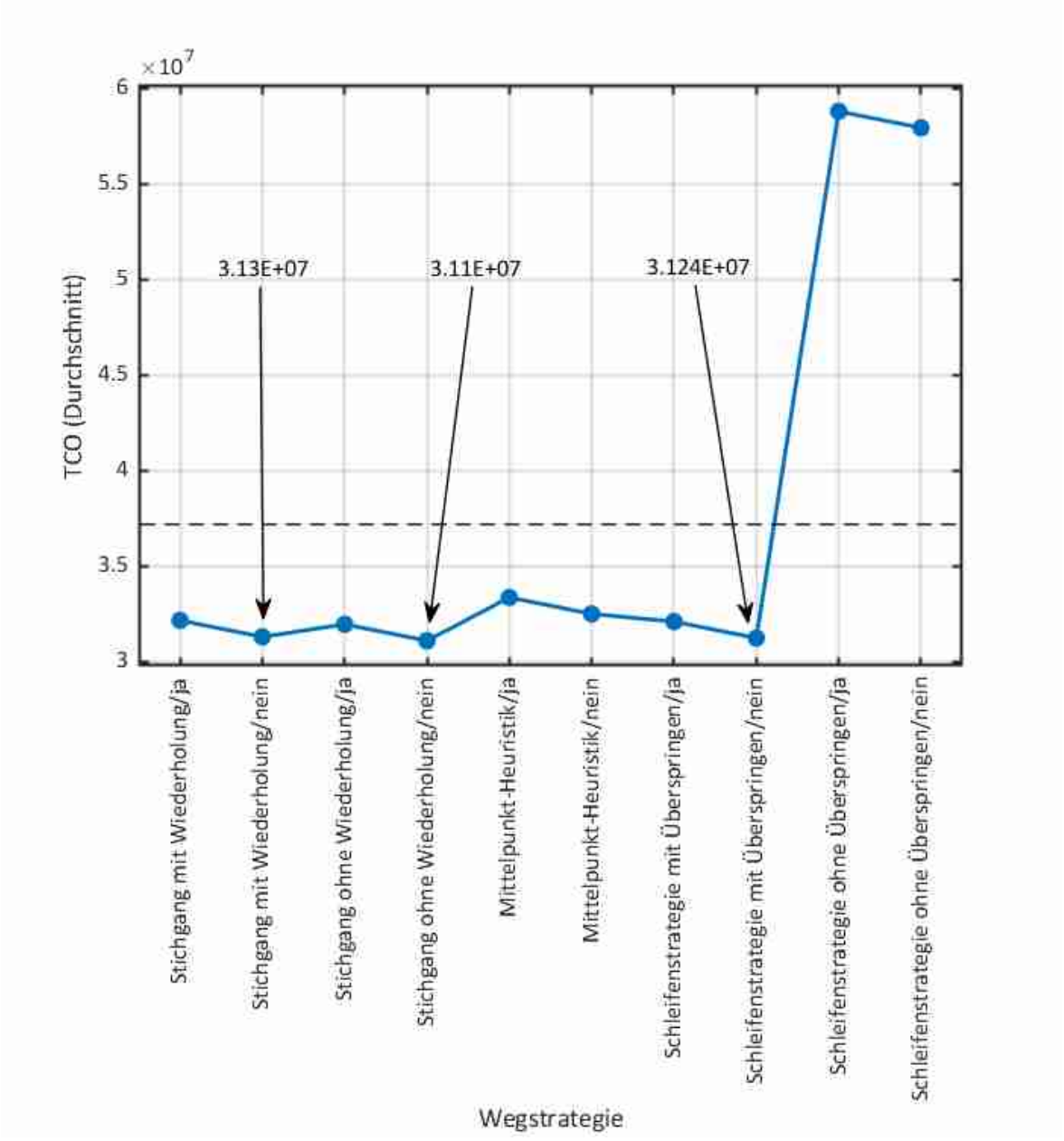


Abbildung 46: Effektdiagramm für Wegstrategie

Im folgenden Schritt wurden zudem noch die Wechselwirkungen der Faktoren untereinander berechnet (Abschnitt 2.2.1, S. 22). Siehe dazu Abbildung 48. Dies geschieht wiederum in MATLAB mit der Funktion `interactionplot()`[50]. Der MATLAB-Code zu den Effekt- und Wechselwirkungsdiagrammen kann im Datenanhang unter „03_Analyse_von_Wirkzusammenhängen\01_Konventionelles_Kommissionieren_gassenungebunden\01_Versuchsplanung\Tool_Input\Vorversuch\Maineffects_01.m“ gefunden werden.

[50] Online Dokumentation unter: https://www.mathworks.com/help/stats/interactionplot.html

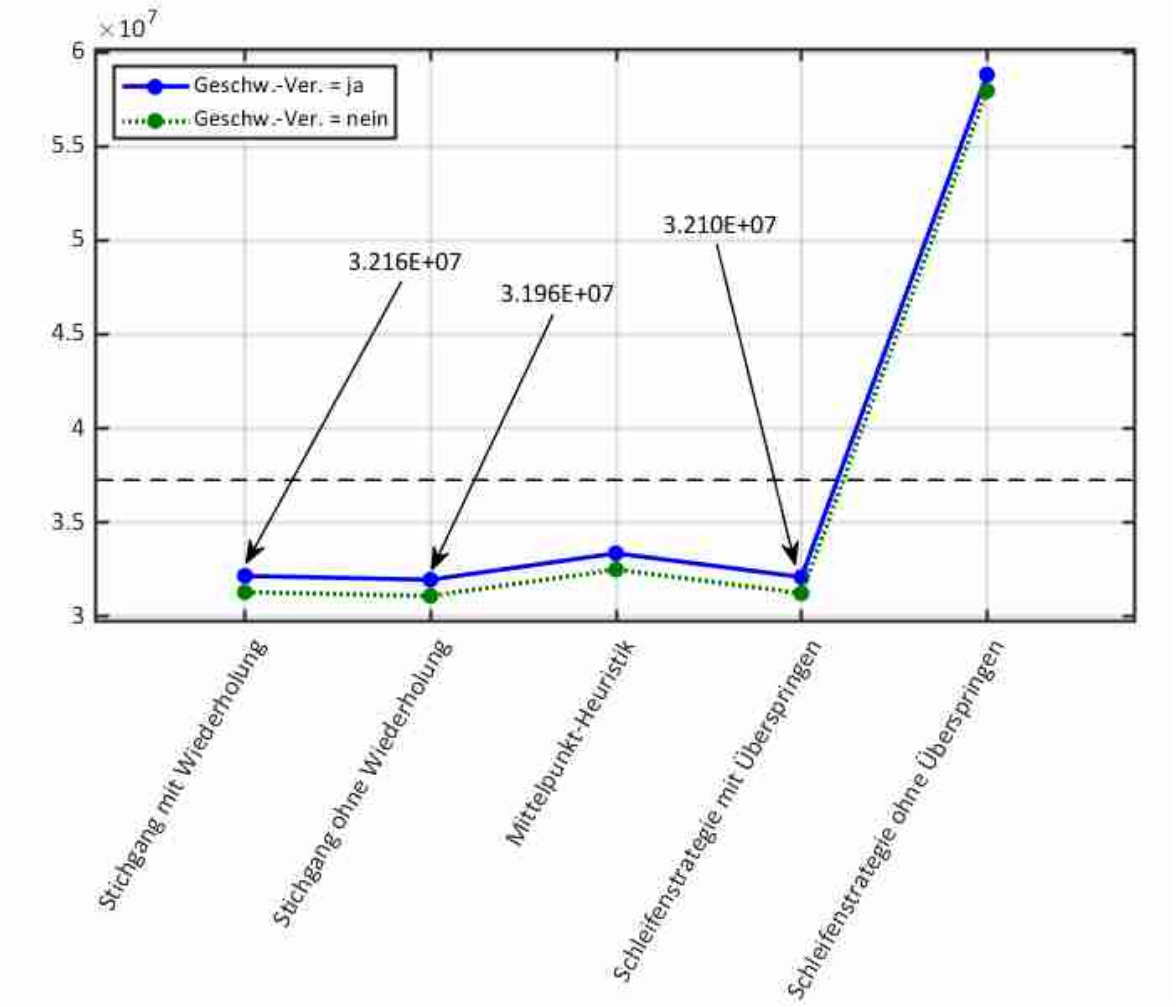


Abbildung 47: Wechselwirkung zwischen Wegstrategie und Geschwindigkeitsveränderung

In Abbildung 47 wurde das Effektdiagramm aus Abbildung 46 in die beiden Komponenten „Wegstrategie“ und „Geschwindigkeitsveränderung bei Gassenwechsel“ aufgeschlüsselt. Interessanterweise scheint eine vorhandene Geschwindigkeitsänderung auf alle Westrategien eine gleich hohe durchschnittliche Verschlechterung des TCO zu bewirken. Es stellt sich nun die Frage, welche Wegstrategie ausgewählt werden sollte und ob hier eine eher pessimistische Sicht angenommen wird: Generell ist es wahrscheinlicher, dass von den vorhandenen Komponenten eines Kommissioniersystems (hier insbesondere die FM) nicht alle ohne Geschwindigkeitsverlust die Gassenwechsel vollziehen können. Demnach ist hier eine pessimistische Sicht eher angebracht; also eine Betrachtung mit Geschwindigkeitswechsel. Bei den Wegstrategien (mit Geschwindigkeitsveränderung) zeigt sich, dass der „Stichgang ohne Wiederholung“ am besten ist.

L	Layout
L01	Zentralganglayout
L01	Kopfganglayout
V	**Verteilung**
V01	gleichverteilt
V02	exponentialverteilt
A	**Ansteuerung**
A01	geordnet
A02	ungeordnet
B	**Basis**
B01	ohne Basis/-/-
B02	mit Basis/zentral/integriert
B03	mit Basis/dezentral/einseitig
B04	mit Basis/dezentral/beidseitig
W	**Wegstrategie**
W01	Stichgang mit Wiederholung/ja
W02	Stichgang mit Wiederholung/nein
W03	Stichgang ohne Wiederholung/ja
W04	Stichgang ohne Wiederholung/nein
W05	Mittelpunkt-Heuristik/ja
W06	Mittelpunkt-Heuristik/nein
W07	Schleifenstrategie mit Überspringen/ja
W08	Schleifenstrategie mit Überspringen/nein
W09	Schleifenstrategie ohne Überspringen/ja
W10	Schleifenstrategie ohne Überspringen/nein

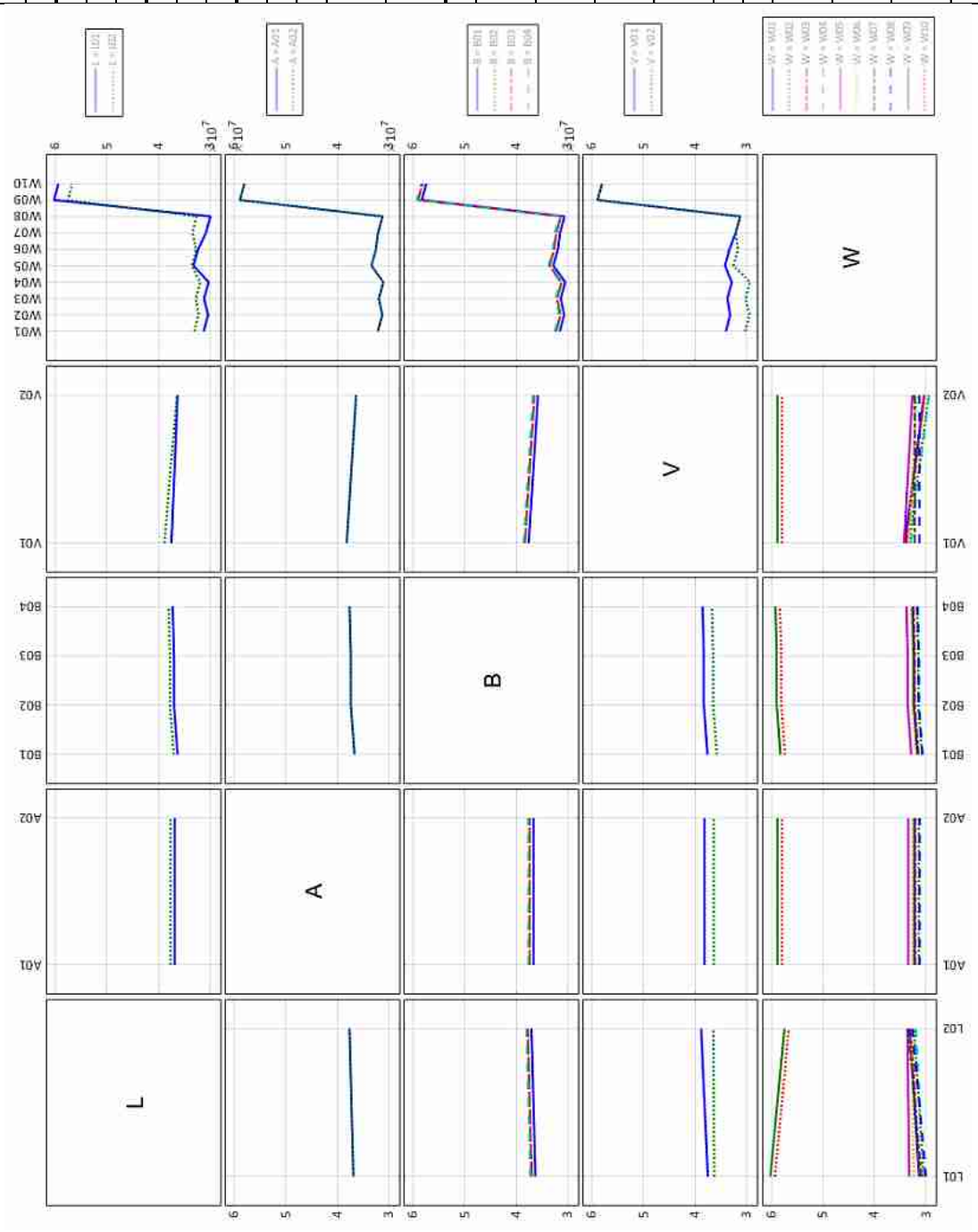


Abbildung 48: Wechselwirkungen gassenungebundenes konventionelles Kommissionieren

Es bildet sich heraus, dass die meisten Faktorkombinationen entweder überhaupt keine Wechselwirkungen untereinander haben oder den TOC lediglich nach oben/unten verschieben, wobei die Effektlinien weitestgehend parallel bleiben. Die Wahl der Basis sowie der Ansteuerung ist dem ersten Fall zuzurechnen, wohingegen die „Verteilung der Artikel im Lager“, das Lager-Layout sowie die Wegstrategie zum letzten Fall angehören. Zudem hat die „Verteilung im Lager“ offensichtlich auf beiden Schleifenstrategien (im Gegensatz zu den anderen Wegstrategien) keinen Einfluss, was auch Sinn macht, da dort ohnehin die gesamte Ganglänge durchschritten wird. Bei der „Schleifenstrategie ohne Überspringen“ ist das Kopfganglayout offensichtlich besser als das Zentralganglayout. Auch dies ergibt wiederum Sinn, da die Wegstrecke durch den Zentralgang etwas länger sein sollte als bei einem Kopfganglayout. Abgesehen davon bestätigen die Wechselwirkungen lediglich die Eindrücke aus den Effektdiagrammen.

Im letzten Schritt können noch die einzelnen Versuchsergebnisse des Vollfaktorplans betrachtet werden. Tabelle 5 zeigt die besten 5 Versuche aufsteigend nach der Höhe des TCO (mit „Geschwindigkeitsveränderung beim Gassenwechsel“). Es zeigt sich auch hier, dass die Ansteuerung überhaupt keinen Einfluss auf den TCO hat. Zudem ist der „Stichgang ohne Wiederholung“ auch die beste Wahl, wenn auch nur knapp vor dem „Stichgang mit Wiederholung“. Eine Basis sollte möglichst vermieden werden.

Tabelle 5: Top 5 des Vollfaktorplans (gassenungebunden)

Lager-Layout	Verteilung der Artikel im Lager	Art der Ansteuer-ung	Art der Betracht-ung	Wegstrategie	Geschwindig-keitsverände rung bei Gassenwech sel	Kommiss-ionieren mit/ohne Basis	Lage der Basis	Zonen-durch-gang	TCO
Zentral-ganglayout	exponential-verteilt	geordnet	gassen-ungebunden	Stichgang ohne Wiederholung	ja	ohne Basis	-	-	29.363.745,18
Zentral-ganglayout	exponential-verteilt	ungeordnet	gassen-ungebunden	Stichgang ohne Wiederholung	ja	ohne Basis	-	-	29.363.745,18
Zentral-ganglayout	exponential-verteilt	geordnet	gassen-ungebunden	Stichgang mit Wiederholung	ja	ohne Basis	-	-	29.414.513,83
Zentral-ganglayout	exponential-verteilt	ungeordnet	gassen-ungebunden	Stichgang mit Wiederholung	ja	ohne Basis	-	-	29.414.513,83
Kopfgang-layout	exponential-verteilt	geordnet	gassen-ungebunden	Stichgang ohne Wiederholung	ja	ohne Basis	-	-	29.828.860,87

Zusammen mit den Spannweiten der quantitativen Faktoren aus Tabelle 3 (S. 92) wird das hier betrachtete konventionelle Kommissioniersystem durch die Konfiguration der kategorischen Faktoren aus Tabelle 6 definiert.

Tabelle 6: Konfiguration für gassenungebundenes konventionelles Kommissionieren

Faktor	Faktorstufe
Lager-Layout	Zentralganglayout
Verteilung der Artikel im Lager	exponentialverteilt
Art der Ansteuerung	ungeordnet
Art der Betrachtung	gassenungebunden
Wegstrategie	Stichgang ohne Wiederholung
Geschwindigkeitsveränderung bei Gassenwechsel	ja
Kommissionieren mit/ohne Basis	Ohne Basis
Lage der Basis	-
Zonendurchgang	-

3.2.1.3 Erstellung des Testfeldes

Da nun alle relevanten Faktoren ausgewählt und entweder auf Stufen festgesetzt oder ihnen Spannweiten zugeordnet wurden, kann das Testfeld für die Versuchsdurchführung erstellt werden.

Es sollen insgesamt 18 quantitative Faktoren hinsichtlich deren Einfluss auf die Zielgrößen untersucht werden. Dementsprechend muss ein 18-dimensionales $(p = 18)$ Testfeld generiert werden, in dem die Testpunkte möglichst gleichverteilt sind. Eine Schätzung darüber, wie viele Testpunkte (Samples) benötigt werden, ist sehr schwer und voll Fall zu Fall unterschiedlich. Nach einer Schätzung von Herrn Prof. Oliver Nelles sollten etwa $N = 30.000$ Samples für ein 18-dimensionales Testfeld ausreichen.

Aus den bereits genannten Gründen bietet sich bei einer so hohen Anzahl von Testpunkten die Sobol'-Sequenz zur Erstellung des Testfeldes an (siehe dazu Abschnitt 2.2.2 ab S. 26).

Da die Sobol'-Sequenz jedoch nur Punkte im Intervall $[0; 1]$ produziert, müssen diese für jeden Faktor bzgl. der Spannweites des Faktors skaliert werden. Tabelle 3 zeigt pro Faktor neben dem Minimum und Maximum zudem die Differenz zwischen den beiden Werten an. Diese Differenz wird mit dem Wert der Sobol'-Sequenz multipliziert, anschließend wird

dazu noch das Minimum des Faktors aufaddiert. Nachdem alle Samples skaliert wurden, erhält man das fertige Testfeld $\boldsymbol{T}$ in Form einer $N \times p$ Matrix. Der hier verwendete MATLAB-Code ist in Quellcode 1 inklusive Kommentierung der einzelnen Schritte zu sehen. Die beteiligten Tabellen können im Datenanhang[51] eingesehen werden. Bei der Multiplikationsmatrix $\boldsymbol{M}$ handelt es sich um eine $p \times p$ Matrix, welche auf der Diagonalen die Differenz zwischen Maximum und Minimum für jeden Faktor enthält. Die Additionsmatrix $\boldsymbol{A}$ ist hingehen eine $N \times p$ Matrix und enthält sämtliche Minima der beteiligten Faktoren.

```
%Erzeugen einer 18D Sobol-Sequenz S
%Und Abtrennen der ersten 30.000 Samples
p = sobolset(18);
S = net(p,30000);

%Import der Multiplikations- M und Additionsmatrix A
source1 = "Tool-Input-Variables-Multiplication-Matrix.csv";
source2 = "Tool-Input-Variables-Addition-Matrix.xlsx";
M = xlsread(source1);
A = xlsread(source2);

%Multiplikation der Sobol-Sequenz mit Multiplikationsmatrix M
%Anschließende Addition der Min-Werte durch die Additionsmatrix A
T = (S*F)+A;

%Export des Testfeldes T in eine Excel-Datei
filename = "Tool-Input-Konventionelles-Kommissionieren-18D-N30k.xlsx";
xlswrite(filename,T);
```

Quellcode 1: Erzeugung des Testfeldes in MATLAB

3.2.1.4 Versuchsdurchführung

Das Testfeld $\boldsymbol{T}$ kann nun in das TB „Versuchsplan“ im Excel-Tool eingefügt werden. Mit dem VBA-Makro[52] `Run_Test()` werden nacheinander die Samples in die entsprechenden Felder des Excel-Tools geschrieben und eine Berechnung durchgeführt. Anschließend werden die vier Zielgrößen in die entsprechende Spalte des Versuchsfeldes eingefügt. Das Makro zeichnet zudem die verstrichene Zeit des gesamten Versuches auf, was für einen Vergleich zwischen Excel-Tool und Metamodell bzgl. des Zeitaufwandes pro Rechnung nützlich ist.

[51] Siehe dazu: „03_Analyse_von_Wirkzusammenhängen\01_Konventionelles_Kommissionieren_gassen ungebunden\01_Versuchsplanung\Tool_Input“

[52] Visual Basic for Applications (VBA)

3.2.2 Metamodelltraining und -auswahl

Im diesem Abschnitt soll ein Metamodell, in Form eines MLPs (siehe Abschnitt 2.3.2, ab S. 35), mit den Ergebnissen der Versuche aus dem vorigen Abschnitt trainiert werden. Die Einheit aus Versuchsplan und der daraus resultierenden Ergebnisse, wird im Folgenden als „Datensatz“ bezeichnet. Die Wahl des Metamodells fällt hier auf ein MLP, da diese recht universell einsetzbar und vergleichsweise schnell trainierbar sind. Zudem existiert für sie eine Methode zur Bestimmung der Relative Importance einzelner Prädiktoren, die außerdem angibt, in welche Richtung (positiv/negativ) der jeweilige Prädiktor die Zielgröße beeinflusst. Dies ist bei anderen Metamodelltypen nicht der Fall (bspw. *Random Forests*).

Wie bereits angesprochen, soll hier der *MLPRegressor* aus dem Machine-Learning-Paket *scikit-learn* in der Programmiersprache Python verwendet werden. Python ist eine leicht verständliche Open-Source-Programmiersprache, welche durch den Einsatz verschiedener, schon vorhandener Pakete einfach erweitert werden kann. Eine Auflistung aller verwendeter Python-Pakete ist in A-3 (S. 203) zu finden.

Der MLPRegressor kann mit insgesamt 21 unterschiedlichen Hyperparametern konfiguriert werden[53]. Einige davon stehen jeweils in Abhängigkeiten zu anderen gewählten Hyperparametern bzw. sollten nicht verändert und daher in den Standardeinstellungen belassen werden.

[53] Eine komplette Dokumentation der MLPRegressor-Hyperparameter kann online unter http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html gefunden werden

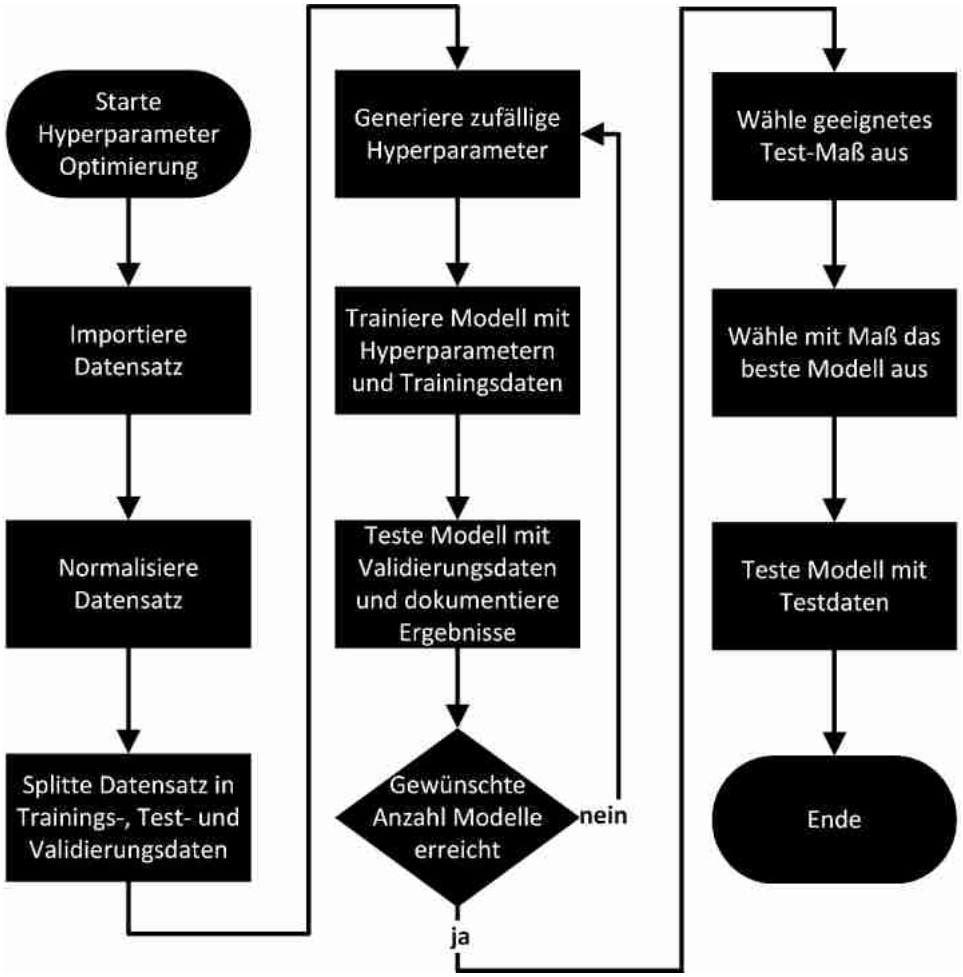


Abbildung 49: Vorgehensweise der Hyperparameter-Optimierung

Um die Güte des MLP zu verbessern, sollen die übrigen Hyperparameter jedoch optimiert werden. Hierzu bietet sich die in Abschnitt 2.3.3.3 (S. 53 ff.) beschriebene *Random Search*, aus den dort genannten Gründen, an. Abbildung 49 zeigt schematisch die hier verwendete Vorgehensweise der Hyperparameter-Optimierung mit Hilfe der Random Search (Schleife im Inneren). Da hier für jede der vier Zielgrößen ein eigenes MLP optimiert werden soll, muss diese Prozedur dementsprechend viermal durchlaufen werden.

```
518.    def random_search(n_models):
519.        #Anzahl Durchläufe pro Zielgröße
520.        n = n_models
521.
522.        #Schleife über alle vier Zielgrößen
523.        for y_select in range(1,5):
524.            #Versuchsergebnisse des Excel-Tools werden importiert
525.            filename = "Excel-Tool-Output-GUG_Konv_Kommis-18D-N30k.csv"
526.            data = import_data(filename)
527.
528.            #Jede Dimension (Spalte) wird in da Intervall [0;1] normalisiert
529.            data = scale_data(data)
530.            #Datensatz wird in Prädiktoren-Matrix X
531.            #und Zielgrößen-Vektor y für die betrachtete Zielgröße y_select
532.            #aufgeteilt
533.            X, y = split_data_xy(data, y_select)
534.
535.            #Split der Daten in Trainings- (60%),
536.            #sowie Test- und Validierungsdaten mit je 20%
537.            X_train, y_train, X_test, y_test, X_val, y_val = split_data_ttv(X, y,
538.                                                                            0.6,
539.                                                                            0.2,
540.                                                                            0.2)
541.
542.            #Daten und Parameter für Random-Search werden übergeben
543.            #Resultate der Random-Search für die aktuelle Zielgröße werden
544.            #anschließend empfangen
545.            mlp_validation_results = rand_search_mlp(n, X_train, y_train, X_val,
546.                                                     y_val, y, y_select)
547.
548.            #Resultate der Random-Search werden in Excel-Tabelle exportiert
549.            title = ("Models\GUG_Konv_Kommis_y" + str(y_select) +
550.                     "_MLP_Validation_"  + "Results_N=" + str(len(y_train)) +
551.                     "_n=" + str(n) + "_" + str(len(X_train.columns)) + "D_" +
552.                     str(random.randint(1000,9999)) + ".xlsx")
553.            #Ergebnisse werden als Excel-Datei exportiert
554.            df_to_excel(title, mlp_validation_results)
```

Quellcode 2: Grundfunktion der Hyperparameter-Optimierung

Quellcode 2 zeigt die Implementierung der Grundfunktion der Hyperparameter-Optimierung im Python-Code. Die Zeilennummerierung hier stimmt mit der des Python-Codes im Datenanhang überein. Die Variable `y_select` ist ganzzahlig und bezeichnet die aktuell betrachtete Zielgröße. Mit der Größe n wird die Anzahl der zu testenden Modelle beschrieben.

Im ersten Schritt wird der komplette Datensatz importiert. Anschließend müssen die Daten für die Verwendung im MLP vorbereitet werden (siehe Abschnitt 2.3.2.2.1, S. 41 f.). Dazu wird der gesamte Datensatz spaltenweise mit einer linearen Transformation in das Intervall $[0; 1]$ transformiert. Da hier keine Datenpunkte mit negativem Vorzeichen vorhanden sind, bietet sich diese Transformationsart an. Zudem ist sie, im Vergleich zu einer z-

Transformation, noch recht leicht zu interpretieren, wenn man ungefähr weiß, in welchem Rahmen sich die Zielgrößen befinden.

Anschließend werden die Daten in Trainings-, Test- und Validierungsdaten aufgesplittet (Abschnitt 2.3.1.1, S. 32 f.). Die Empfehlungen von Hastie et al. 2017 dienen hier als Ausgangspunkt, allerdings werden hier 60% der Daten zum Training verwendet, da mit jeweils 20% Validierungs- und Testdaten immer noch 6.000 Datenpunkte zum Testen der Modellgüte vorhanden sind, was für gute Berechnungsergebnisse ausreichen sollte. In Quellcode 3 ist die Funktion zum Split in die einzelnen Datensätze zu sehen, welche in Quellcode 2 (Zeile 537-540) aufgerufen wurde. Da die verwendete (importierte) Funktion `train_test_split` nur Datensätze in zwei Teildatensätze splitten kann, muss der Vorgang zweimal durchgeführt werden. Durch den Parameter `random_state` wird die pseudo-zufällige Aufteilung in die Datensätze initialisiert. Dadurch, dass diesem ein statischer Wert zugewiesen wurde, sind stets dieselben Samples in den jeweiligen Datensätzen enthalten, was eine Reproduzierbarkeit der Ergebnisse sicherstellt.

```
88. def split_data_ttv(X, y, train, test, val):
89.     #Split in Trainings-, Validierungs- und Testdaten
90.     #Verhältnis Trainings-Test-/Validierungsdaten berechnen
91.     ts=1-train-test
92.     #Erster Split
93.     X_train_test, X_val, y_train_test, y_val=train_test_split(X,
94.                                                               y,
95.                                                               test_size=ts,
96.                                                               random_state=666)
97.
100.    #Neue test_size berechnen
101.    ts=val/(train+test)
102.    #Zweiter Split
103.    X_train, X_test, y_train, y_test=train_test_split(X_train_test,
104.                                                      y_train_test,
105.                                                      test_size=ts,
106.                                                      random_state=666)
107.
113.    #Trainings-, Test- und Validierungsdatensätz zurückgeben
114.    return X_train, y_train, X_test, y_test, X_val, y_val
```

Quellcode 3: Split in Trainings-, Test- und Validierungsdaten

Die Trainings- und Validierungsdaten werden dann an die Funktion der Random-Search-Schleife zur Berechnung und Evaluation der MLP-Konfigurationen gegeben (Zeile 114). Der Code dazu ist in Quellcode 4 und Quellcode 5 zu sehen.

```
244.    def rand_search_mlp(n, X_train, y_train, X_val, y_val, y, y_select):
245.        #Ein Dictionary wird eröffnet. Dort werden die Modellkonfigurationen und
246.        #Testergebnisse dokumentiert
247.        mlp_validation_results = defaultdict(list)
248.        mlp_validation = {}
252.        i=1
253.
254.        while i <= n:
255.
256.            #Startzeit wird zur Bestimmung der Trainingszeit festgehalten
257.            t_start = time.time()
258.
261.            ####################################
262.            #zufällige Hyperparameter festlegen
263.
264.            #Anzahl der Nodes pro Hidden-Layer eingrenzen (Min/Max)
265.            min_nodes = 10
266.            max_nodes = 200
267.
268.            #Anzahl der Hidden-Layer zufällig bestimmen
269.            layer = random.randint(1,3)
270.            #Nodes der ersten Hidden-Layer bestimmen
271.            nodes = random.randint(min_nodes, max_nodes)
272.
273.            #Hidden-Layer und deren Nodes werden in Tuple geschrieben
274.            hidden_layer_sizes = (nodes,)
275.
276.            if layer != 1:
277.                for j in range(1, layer):
278.                    nodes = random.randint(min_nodes, max_nodes)
279.                    hidden_layer_sizes += (nodes,)
280.
281.            #Restliche Hyperparameter werden zufällig ausgewählt
282.            random_state = random.randint(0,19)
283.            activation = random.choice(["relu", "logistic", "tanh", "identity"])
284.            learning_rate_init = random.choice([0.0001, 0.001, 0.01])
285.            batch_size = random.choice([100, 200, 400])
286.            early_stopping = random.choice([True, False])
287.
288.            #Hyperparameter werden zum Training des MLPs weitergegeben
289.            #Die komplette Modellkonfiguration und das Modell an sich
290.            #werden zurückgegeben
291.            model, model_config, model_params = train_mlp(X_train,
292.                                                          y_train,
293.                                                          y_select,
294.                                                          hidden_layer_sizes,
295.                                                          activation,
296.                                                          learning_rate_init,
297.                                                          batch_size,
298.                                                          random_state,
299.                                                          early_stopping)
```

Quellcode 4: Funktion zur Random Search für MLPs

```
301.            #Evaluation des MLPs über mehrere Bewertungsmethoden
302.            rmse, nrmse, mae, mse, r2 = test_model(X_val,
303.                                                   y_val,
304.                                                   y_select,
305.                                                   model,
306.                                                   False,
307.                                                   model_config)
308.
309.            t_time = time.time() - t_start
310.
325.            #Evaluationsdaten und Modellkonfiguration dem Dictionary anfügen
326.            mlp_validation["n_iter_"] = model.n_iter_
327.            mlp_validation["model_no"] = i
328.            mlp_validation["R^2"] = r2
329.            mlp_validation["RMSE"] = rmse
330.            mlp_validation["NRMSE"] = nrmse
331.            mlp_validation["MAE"] = mae
332.            mlp_validation["MSE"] = mse
333.            mlp_validation["t_time"] = t_time
334.
335.            for k, v in chain(mlp_validation.items(), model_params.items()):
336.                mlp_validation_results[k].append(v)
337.
338.            i += 1
339.
340.        #Das Dictionary mit den Resultaten wird in einen DataFrame umgewandelt
341.        #und zurückgegeben
342.        return pd.DataFrame.from_dict(mlp_validation_results)
```

Quellcode 5: Funktion zur Random Search für MLPs (Fortsetzung)

Die wohl wichtigsten Hyperparameter eines MLP sind die Anzahl der Hidden-Layer sowie die Zahl der Nodes pro Layer. Diese beiden werden in MLPRegressor durch den Parameter `hidden_layer_sizes` definiert. Bei dessen Datentyp handelt es sich um ein *Tuple* (ähnlich einer Liste). Bei `hidden_layer_sizes = (5, 20, 8)` würde das MLP über drei Hidden-Layer mit jeweils 5, 20 und 8 Neuronen verfügen. Auf Standardeinstellungen besitzt das MLP eine Hidden-Layer mit 100 Neuronen: `hidden_layer_sizes = (100,)`.

Zur Wahl der optimalen Anzahl von Hidden-Layer und deren Nodes existieren einige Daumenregeln, welche jedoch meist keiner wissenschaftlichen Basis entspringen oder nur in sehr speziellen Szenarien gültig sind. Es gibt daher keine allgemeine Aussage darüber, wie viele Layer und Nodes ein MLP besitzen sollte. Hastie et al. 2017 empfiehlt daher lieber zu viele Nodes zu verwenden, als zu wenige (LeCun et al. 1998, S. 9; Thomas et al. 2015).

Es werden deshalb 1 bis 3 Hidden-Layer mit jeweils 10-200 Neuronen pro Layer getestet (siehe Quellcode 4, Zeile 264-279). Die genaue Spannweite der Neuronen ist hier eher zufällig gewählt und hat keine spezielle Bedeutung.

Der Hyperparameter `random_state` bestimmte die Initialisierung der quasi-zufällig bestimmten Anfangsgewichtungen und Bias-Werte des MLP. Dieser Parameter ist zur Reproduzierbarkeit der Modelle sehr wichtig. Zudem bietet er die Möglichkeit, bei gleichbleibender Modellarchitektur von verschiedenen Punkten im Lösungsraum die Optimierung des MLPs (durch den Gradient Descent [Abschnitt 2.3.2.2.2, S. 42 ff.]) durchzuführen und somit zu unterschiedlichen Modellgüten zu gelangen. Quellcode 4, Zeile 282 zeigt, dass dieser Hyperparameter einen ganzzahligen Wert zwischen 0 und 19 einnehmen kann.

Eine weitere wichtige Einstellmöglichkeit ist die Wahl der Aktivierungsfunktion durch den Parameter `activation`. Hier stehen die vier beschriebenen Funktionen aus Abschnitt 2.3.2.1 (S. 36 ff.) zur Verfügung (siehe Quellcode 4, Zeile 283). Die ReLU-Funktion wird hier standardmäßig verwendet.

Die Lernrate des MLPs kann ebenfalls eingestellt werden (Abschnitt 2.3.2.2.2, S. 42 ff.). Der Hyperparameter `learning_rate_init` bestimmt die anfängliche Lernrate der Backpropagation und besitzt auf Werkseinstellungen den Wert $0{,}001$. Neben diesem Wert soll hier das 10-fache sowie $\frac{1}{10}$ dieses Wertes getestet werden: Quellcode 4, Zeile 284.

Der MLPRegressor verwendet zudem den SGD mit einstellbaren Mini-batch-Größen (Abschnitt 2.3.2.2.3, S. 46 f.). Dies geschieht über `batch_size`. Standardmäßig: `batch_size=min(200, n_samples)`. Da hier weitaus mehr als 200 Samples zum Training des MLPs zur Verfügung stehen, wird mit 100er, 200er und 400er Batches experimentiert (Quellcode 4, Zeile 285).

Bei dem letzten hier betrachteten Hyperparameter handelt es sich um die Verwendung des Early Stoppings (Abschnitt 2.3.2.2.5, S. 50 f.): `early_stopping`. Dieser ist entweder wahr oder falsch (Quellcode 4, Zeile 286). Wird Early Stopping verwendet, werden standardmäßig 10% der Trainingsdaten zur Validierung verwendet.

Quellcode 6 zeigt die in Quellcode 4, Zeile 291-299 aufgerufene Funktion zum Training eines MLPs. Dieser Funktion werden sowohl die Trainingsdaten als auch die im Vorhinein zufällig erstellte Hyperparameter-Konfiguration übergeben (Zeile 116-118). Anschließend

wird mit Hilfe des MLPRegressors das Modell(training) konfiguriert (Zeile 122-130) und das eigentliche Training des Modells durch den Befehl in Zeile 133 initialisiert. Im letzten Schritt werden das fertige Modell sowie die komplette Modellkonfiguration (eine Liste aller Hyperparameter) wieder zurückgegeben.

```
116.    def train_mlp(X, y, y_select, hidden_layer_sizes, activation,
117.                  learning_rate_init, batch_size, random_state,
118.                  early_stopping):
119.
120.        #Hyperparameter für MLP-Training mit den übergebenen
121.        #Parametern einstellen
122.        model = MLPRegressor(
123.                hidden_layer_sizes=hidden_layer_sizes,
124.                activation=activation,
125.                solver="adam",
126.                learning_rate_init=learning_rate_init,
127.                batch_size=batch_size,
128.                random_state=random_state,
129.                early_stopping=early_stopping
130.                )
131.
132.        #MLP mit Trainingsdaten trainieren
133.        model.fit(X, y)
134.
135.        #Es wird zusätzlich noch ein String mit den wichtigsten
136.        #Hyperparametern zusammengestellt
137.        #Dieser wird bei Grafiken zu dem Modell verwendet
138.        model_config = ("MLP y" + str(y_select) + " N=" +
139.                        str(hidden_layer_sizes)  + " A=" + activation +
140.                        " BS=" + str(batch_size)  + " LR=" +
141.                        str(learning_rate_init) + " RS=" + str(random_state) +
142.                        " ES=" + str(early_stopping))
143.
144.        print(model_config)
145.        #Das Modell sowie die kompletten und wichtigsten Hyperparameter
146.        #werden zurückgegeben
147.        model_params = model.get_params()
148.        return model, model_config, model_params
```

Quellcode 6: Funktion zum Training eines MLPs

Nun muss das trainierte MLP auf seine Genauigkeit (Güte) hin geprüft werden. Dazu wird in der Funktion der Random-Search-Schleife (Quellcode 5, Zeile 302-307) eine weitere Unterfunktion aufgerufen, welche das Modell testet. Siehe dazu Quellcode 7. An sie werden die Validierungsdaten und das trainierte MLP übergeben[54].

[54] Die Funktion aus Quellcode 7 wird sowohl zum Testen des Modells durch die Validierungs- als auch die Testdaten verwendet. In diesem Fall werden die Validierungsdaten an die Funktion übergeben. Im Inneren der Funktion werden sie jedoch als Testdaten bezeichnet.

```
150.    def test_model(X_test, y_test, y_select, model, graph, model_config):
151.
152.        #Modell mit Validierungs- bzw. Testdaten testen
153.        y_pred = np.array(model.predict(X_test))
154.
155.        #R^2 wird hier mit den unskalierten Daten berechnet!
156.        r2 = model.score(X_test, y_test)
157.
158.        ####################################
159.        #Zur Berechnung der übrigen Maße wird die Zielgröße
160.        #des Modells wieder hochskaliert
161.
162.        #Spannweite der Zielgröße bestimmen
163.        if y_select == 1:
164.            y_min = 3000
165.            y_max = 14873959.8911015
166.        elif y_select == 2:
167.            y_min = 2790.08
168.            y_max = 19617819.2861244
169.        elif y_select == 3:
170.            y_min = 0.0142352903569158
171.            y_max = 175.201082144131
172.        elif y_select == 4:
173.            y_min = 30900.8
174.            y_max = 199638059.671175
175.
176.        #y_pred und y_test wird aus dem Intervall [0;1] wieder hochskaliert
177.        y_test = np.array(y_test)
178.        unscaler = lambda x: (x * (y_max - y_min) + y_min)
179.        vfunc = np.vectorize(unscaler)
180.        y_pred = vfunc(y_pred)
181.        y_test = vfunc(y_test)
182.
183.        #Berechnen der Kenngrößen mit den hochskalierten Zielgrößen
184.        #RMSE
185.        rmse = calc_rmse(y_pred, y_test)
186.        #NRMSE
187.        #zum Normalisieren des RMSE
188.        ydash = (y_max - y_min)
189.        nrmse = calc_nrmse(rmse, ydash)
190.        #MAE
191.        mae = calc_mae(y_pred, y_test)
192.        #MSE
193.        mse = calc_mse(y_pred, y_test)
194.
241.        #Die berechneten Maße werden zurückgegeben
242.        return rmse, nrmse, mae, mse, r2
```

Quellcode 7: Berechnung der Maße zur Regressionsgüte

Da das MLP nur mit Datenpunkten aus dem Intervall $[0; 1]$ trainiert wurde, produziert es selbst auch nur Werte in diesem Intervall. Für die Berechnung des Bestimmtheitsmaßes R^2 stellt dies kein Problem dar, da es ohnehin dimensionslos ist. Für die übrigen Maße müssen jedoch die geschätzten Ergebnisse des MLPs und die realen Validierungsdaten (`y_test`), wieder aus dem Intervall $[0; 1]$ hochskaliert werden. Dazu wird die Formel der linearen Transformation aus Abschnitt 2.3.2.2.1 (S. 41) umgestellt (Zeile 178) und auf die einzelnen

Datenpunkte angewandt. Anschließend werden die in Abschnitt 2.3.3.1 (S. 51 ff.) beschriebenen Maße berechnet und wieder an die Random-Search-Schleife zur Dokumentation zurückgegeben.

Dieser Vorgang wiederholt sich so oft, bis die gewünschte Anzahl an Hyperparameter-Konfigurationen für das MLP erreicht wurde. Im letzten Schritt wird die Dokumentation über die Konfigurationen und deren Testergebnisse in eine Excel-Tabelle exportiert (Quellcode 2, Zeile 549-554).

Es stellt sich hier die Frage: Wie viele Modelle müssen trainiert werden? Je mehr Modelle trainiert werden, desto höher ist die Chance, die optimalen Hyperparameter gefunden zu haben bzw. ihnen möglich nahe gekommen zu sein. In Anbetracht dessen, dass hier MLP-Hyperparameter für vier Zielgrößen optimiert werden sollen und nur endliche Rechenkapazitäten zur Verfügung stehen, sollten 5.000 Modelle pro Zielgröße ausreichend sein; für das hier betrachtete konventionelle Kommissionieren also insgesamt 20.000 MLPs.

Die Spannweiten der berechneten Zielgrößen aus dem Versuchsplan sind in Tabelle 7 eingetragen.

Tabelle 7: Spannweiten der vier Zielgrößen gug.

	y1_invest	y2_betrieb	y3_anz_kommi	y4_tco
Min.	3.000,00	2.790,08	$1{,}424 \cdot 10^{-2}$	30.900,80
Max.	$1{,}487 \cdot 10^{7}$	$1{,}962 \cdot 10^{7}$	175,20	$1{,}996 \cdot 10^{8}$

Investitionskosten (y1_invest)

Aus den 5.000 trainierten Modellen für die Zielgröße y1_invest muss nun das beste MLP ausgewählt werden. Dazu muss zunächst entschieden werden, welches Regressionsmaß für dieses Problem am besten geeignet ist. Der MAE eignet sich bei gleichverteilten Modellfehlern, wohingegen der RMSE bei normalverteilten Fehlern vorzuziehen ist (siehe 2.3.3.1, S. 51 ff.).

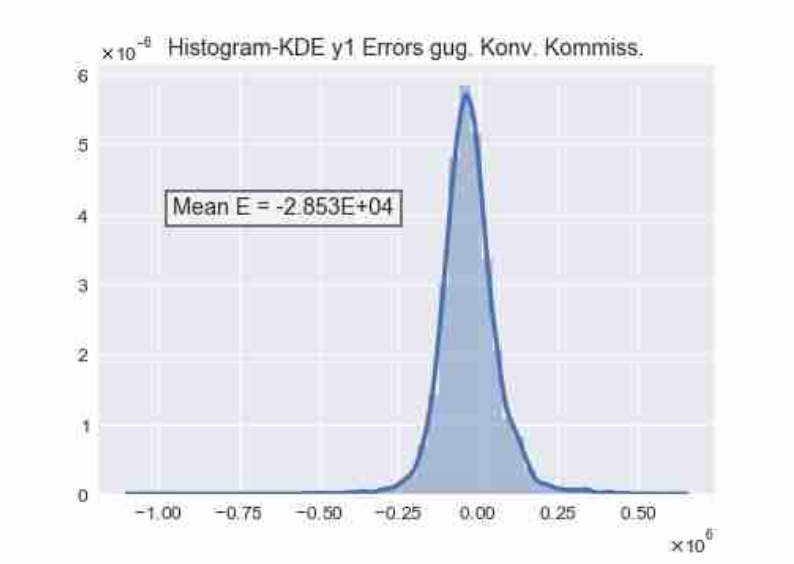


Abbildung 50: Verteilung der Fehler für y1_invest gug.

Um festzustellen, welche Fehlerverteilung vorliegt, wurde das Modell mit dem höchsten R^2 repräsentativ für alle anderen Modelle ausgewählt. Abbildung 50 zeigt eine Kombination aus Histogramm und KDE[55] für die Modellfehler E des MLP. Es zeigt sich, dass die Modellfehler annähernd normalverteilt sind (wobei die Glockenkurve sehr spitz ist). Eine Gleichverteilung der Fehler kann hier ausgeschlossen werden. Folglich sollte der RMSE als Entscheidungsgrundlage für die Modellauswahl dienen.

Tabelle 8 zeigt die 10 besten Modelle bzgl. ihres RMSE auf die Validierungsdaten. Auffällig ist hier, dass alle die Aktivierungsfunktion ReLU und das Early Stopping verwenden. Bei der Betrachtung der Hidden-Layer und deren Nodes ist keine Abhängigkeit zwischen der Anzahl der Nodes und der Modellgüte zu erkennen. Das erste MLP wird zur Berechnung aller folgenden Maße und Interpretationsmethoden für die Zielgröße y1_invest in der gassenungebundenen Betrachtung verwendet.

[55] „*Kerndichtenschätzer*“, engl. „Kernel density estimation“ (KDE). Eine geschätzte Verteilungsfunktion.

Tabelle 8: Top 10 Ergebnisse der Validierungsdaten für y1_invest gug.

	R^2	RMSE	NRMSE	MAE	MSE	activation	batch_size	early_ stopping	hidden_layer_ sizes	learning_ rate_init	random_ state
1	0,99659	9,335E+04	6,277E-03	6,918E+04	8,715E+09	relu	100	TRUE	(90, 70, 194)	0,001	15
2	0,99645	9,526E+04	6,406E-03	6,834E+04	9,074E+09	relu	100	TRUE	(187, 98)	0,001	16
3	0,99628	9,750E+04	6,556E-03	7,153E+04	9,506E+09	relu	100	TRUE	(137, 95, 110)	0,01	18
4	0,99612	9,956E+04	6,695E-03	7,151E+04	9,913E+09	relu	100	TRUE	(135, 74, 88)	0,001	10
5	0,99605	1,004E+05	6,750E-03	7,018E+04	1,008E+10	relu	100	TRUE	(174, 158)	0,01	18
6	0,99604	1,005E+05	6,761E-03	7,316E+04	1,011E+10	relu	200	TRUE	(181, 166, 126)	0,001	7
7	0,99590	1,023E+05	6,877E-03	7,225E+04	1,046E+10	relu	200	TRUE	(117, 37, 137)	0,001	12
8	0,99585	1,030E+05	6,925E-03	7,077E+04	1,061E+10	relu	100	TRUE	(122, 98, 131)	0,01	7
9	0,99582	1,033E+05	6,946E-03	7,711E+04	1,067E+10	relu	100	TRUE	(49, 160, 152)	0,001	3
10	0,99582	1,033E+05	6,946E-03	7,030E+04	1,067E+10	relu	100	TRUE	(196, 182, 69)	0,001	14

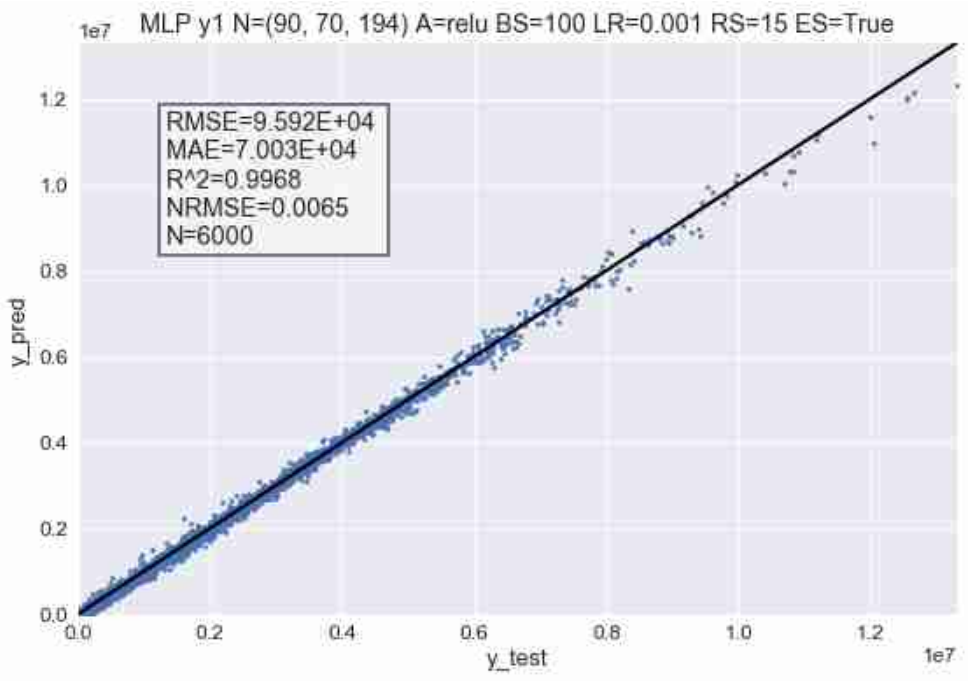


Abbildung 51: Ergebnisse der Testdaten für y1_invest gug.

Um ein verlässliches Maß zu erhalten, wie gut das beste Modell aus den Validierungsdaten wirklich ist, muss es nun mit den bislang unberührten Testdaten getestet werden. Abbildung 51 zeigt den Residual-Plot sowie die errechneten Regressionsmaße aus den Testdaten für das betrachtete MLP. Es zeigt sich, dass die Punkte im unteren und mittleren Bereich von y1_invest recht gleichmäßig nahe um die Referenzlinie verteilt sind. Im oberen Bereich tendiert das Modell jedoch dazu, zu kleine Werte zu schätzen. Ein RMSE von $9{,}592 \cdot 10^4$ erscheint bei der Spannweite des y1_invest von $1{,}487 \cdot 10^7$ sehr gut zu sein.

Die beschriebene Prozedur zur Modellauswahl wiederholt sich inhaltlich für alle folgenden Zielgrößen.

Betriebskosten (y2_betrieb)

Auch hier ist eine sehr spitze Glockenkurve zu erkennen (Abbildung 52), dementsprechend wird der RMSE als entscheidendes Maß verwendet.

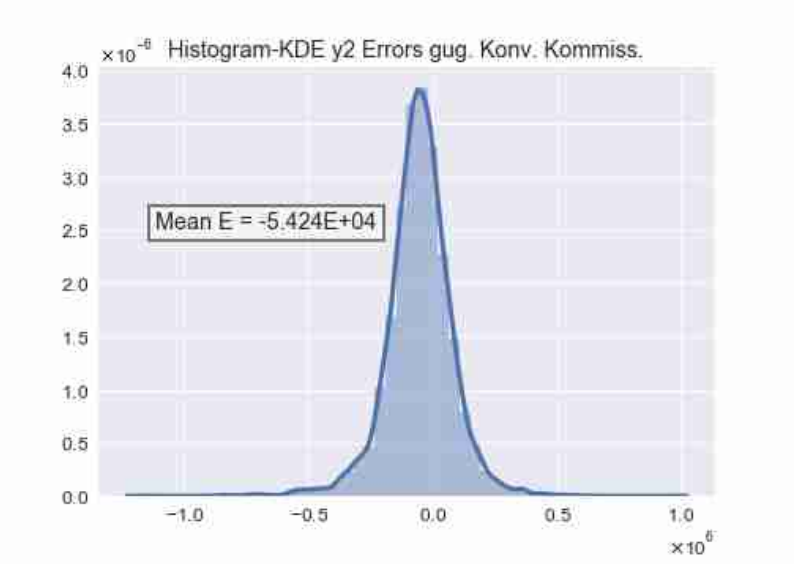


Abbildung 52: Verteilung der Fehler für y2_betrieb gug.

In Tabelle 9 sind die besten 10 Modelle (bzgl. der Validierungsdaten) für die Zielgröße y2_betrieb in der gassenungebundenen Betrachtung abgetragen.

Tabelle 9: Top 10 Ergebnisse der Validierungsdaten für y2_betrieb gug.

	R^2	RMSE	NRMSE	MAE	MSE	activation	batch_ size	early_ stopping	hidden_layer_ sizes	learning_ rate_init	random_ state
1	0,99588	1,404E+05	7,156E-03	1,043E+05	1,970E+10	relu	100	TRUE	(199, 21, 162)	0,001	4
2	0,99552	1,464E+05	7,463E-03	1,094E+05	2,143E+10	relu	100	TRUE	(196, 157)	0,001	16
3	0,99529	1,501E+05	7,654E-03	1,127E+05	2,254E+10	relu	200	TRUE	(54, 132, 184)	0,001	18
4	0,99467	1,597E+05	8,139E-03	1,216E+05	2,549E+10	relu	100	TRUE	(137, 66)	0,001	4
5	0,99453	1,618E+05	8,248E-03	1,210E+05	2,618E+10	relu	100	TRUE	(174, 131, 58)	0,001	16
6	0,99448	1,624E+05	8,279E-03	1,212E+05	2,637E+10	relu	200	TRUE	(187, 130, 128)	0,001	1
7	0,99445	1,628E+05	8,302E-03	1,197E+05	2,652E+10	relu	100	TRUE	(69, 190, 159)	0,001	1
8	0,99435	1,643E+05	8,377E-03	1,206E+05	2,700E+10	relu	200	TRUE	(166, 179, 194)	0,001	19
9	0,99429	1,653E+05	8,426E-03	1,237E+05	2,732E+10	relu	100	TRUE	(124, 150)	0,001	17
10	0,99407	1,683E+05	8,582E-03	1,239E+05	2,834E+10	relu	100	TRUE	(156, 88, 193)	0,001	18

Die Ergebnisse aus den Testdaten für das beste MLP sind in Abbildung 53 zu sehen. Die Schätzung des Modells ist in Anbetracht der großen Spannweite von y2_betrieb ($1{,}961 \cdot 10^7$) weiterhin sehr gut.

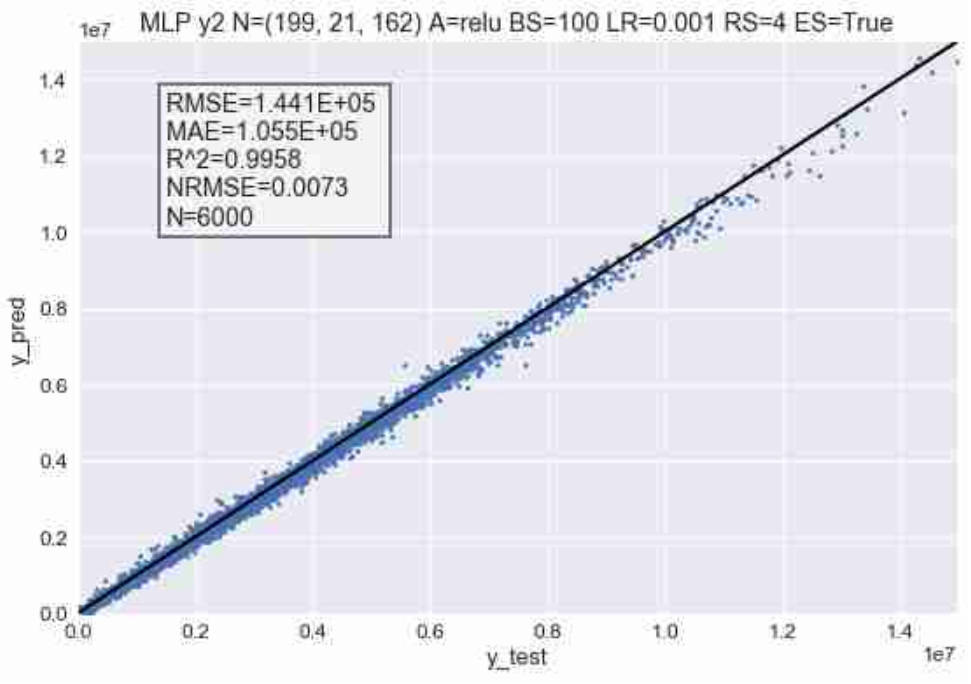


Abbildung 53: Ergebnisse der Testdaten für y2_betrieb gug.

Anzahl Kommissionierer (y3_anz_kommi)

Auch bei y3_anz_kommi sollte das MLP mit Hilfe des RMSE ausgewählt werden (siehe Abbildung 54).

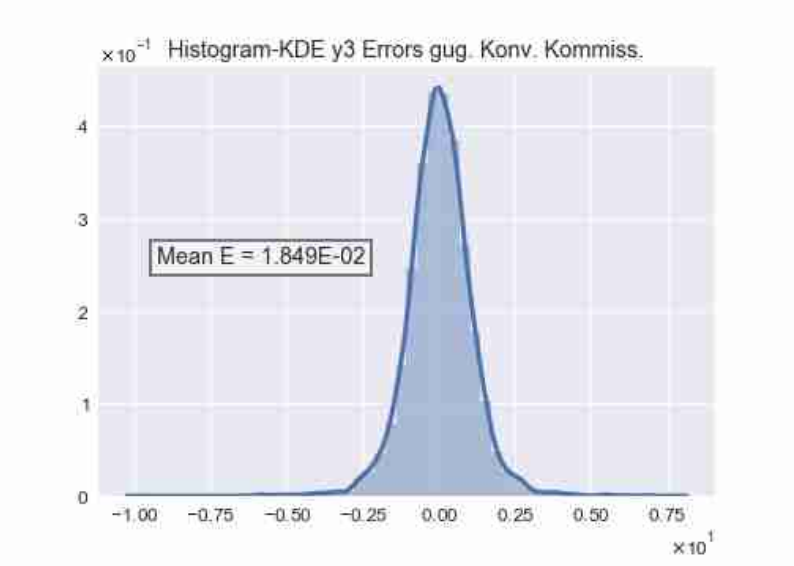


Abbildung 54: Verteilung der Fehler für y3_anz_kommi gug.

In Tabelle 10 kann man erkennen, wie zufällig die Leistungsunterschiede zwischen verschiedenen MLP-Konfigurationen sind. Bislang wurden nur MLPs ausgewählt, welche drei Hidden-Layer besaßen. Nun ist ein MLP mit einer einzigen Hidden-Layer das Beste.

Zudem taucht das erste Mal eine andere Aktivierungsfunktion, neben dem ReLU, in den Top 10 auf: der Tangens Hyperbolicus.

Tabelle 10: Top 10 Ergebnisse der Validierungsdaten für y3_anz_kommi gug.

	R^2	RMSE	NRMSE	MAE	MSE	activation	batch_size	early_stopping	hidden_layer_sizes	learning_rate_init	random_state
1	0,99674	1,039E+00	5,930E-03	7,803E-01	1,079	relu	100	TRUE	(175,)	0,001	5
2	0,99673	1,041E+00	5,941E-03	7,709E-01	1,083	relu	100	TRUE	(24, 103, 138)	0,001	6
3	0,99661	1,059E+00	6,046E-03	7,647E-01	1,122	relu	100	TRUE	(181, 33, 184)	0,001	10
4	0,99660	1,061E+00	6,055E-03	7,655E-01	1,125	relu	200	TRUE	(34, 99, 119)	0,01	7
5	0,99647	1,080E+00	6,167E-03	7,672E-01	1,167	tanh	100	TRUE	(199, 172, 16)	0,001	3
6	0,99643	1,087E+00	6,206E-03	7,897E-01	1,182	relu	100	TRUE	(30, 80, 179)	0,001	15
7	0,99633	1,103E+00	6,294E-03	8,049E-01	1,216	relu	100	TRUE	(187, 69, 122)	0,001	17
8	0,99621	1,121E+00	6,397E-03	8,337E-01	1,256	relu	100	TRUE	(123, 99)	0,001	11
9	0,99618	1,125E+00	6,420E-03	7,868E-01	1,265	relu	100	TRUE	(170, 11, 160)	0,0001	3
10	0,99604	1,145E+00	6,536E-03	8,493E-01	1,311	relu	100	TRUE	(112, 80, 109)	0,001	4

Es zeigt sich in Abbildung 55 anhand des RMSE, dass das ausgewählte Metamodell in der Lage ist, die Anzahl der benötigten Kommissionierer (im Durchschnitt) auf ± 1 genau zu schätzen.

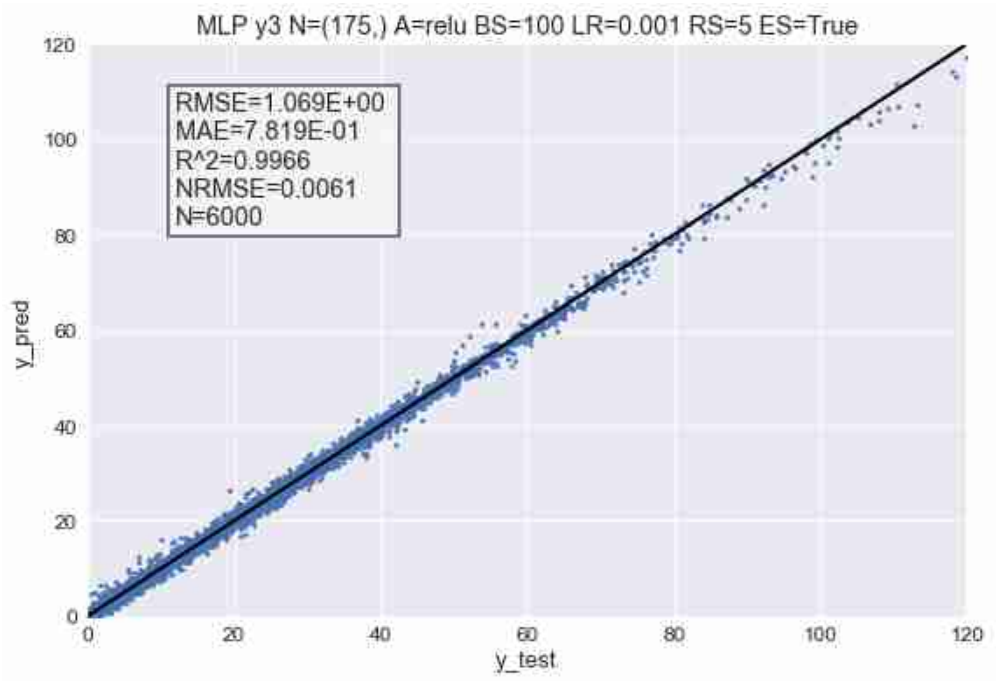


Abbildung 55: Ergebnisse der Testdaten für y3_anz_kommi gug.

TCO (y4_tco)

Auch bei der Total Cost of Ownership sollte der RMSE verwendet werden (Abbildung 56).

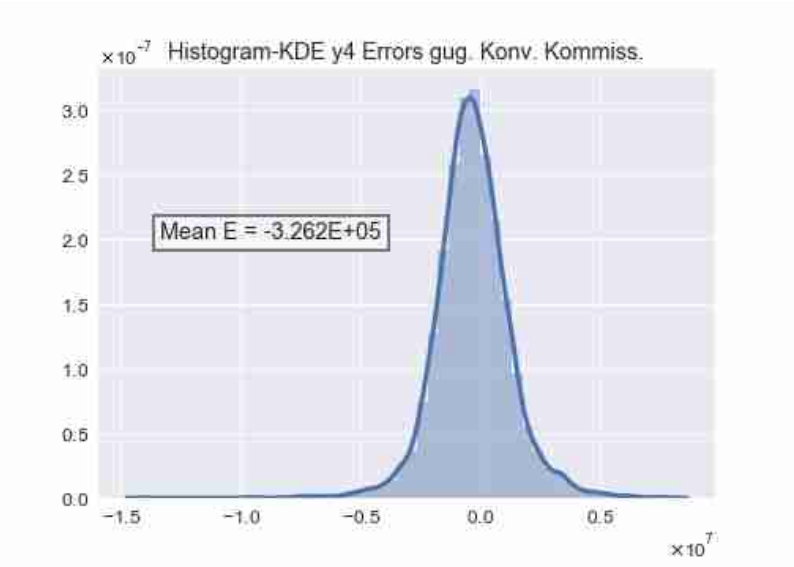


Abbildung 56: Verteilung der Fehler für y4_tco gug.

Tabelle 11 zeigt die Top 10 Metamodelle für die Zielgröße y4_tco.

Tabelle 11: Top 10 Ergebnisse der Validierungsdaten für y4_tco gug.

	R^2	RMSE	NRMSE	MAE	MSE	activation	batch_ size	early_ stopping	hidden_layer_ sizes	learning_ rate_init	random_ state
1	0,99526	1,571E+06	7,872E-03	1,183E+06	2,469E+12	relu	100	TRUE	(88, 120, 176)	0,001	14
2	0,99501	1,612E+06	8,075E-03	1,196E+06	2,598E+12	relu	100	TRUE	(199, 74, 29)	0,001	6
3	0,99496	1,620E+06	8,118E-03	1,203E+06	2,626E+12	relu	100	TRUE	(126, 99, 200)	0,001	1
4	0,99464	1,671E+06	8,373E-03	1,233E+06	2,793E+12	relu	100	TRUE	(83, 108)	0,001	13
5	0,99464	1,671E+06	8,373E-03	1,229E+06	2,793E+12	relu	200	TRUE	(150, 171, 19)	0,001	19
6	0,99459	1,679E+06	8,409E-03	1,255E+06	2,817E+12	relu	100	TRUE	(140, 48)	0,001	19
7	0,99458	1,681E+06	8,420E-03	1,275E+06	2,825E+12	relu	200	TRUE	(163, 127)	0,001	18
8	0,99456	1,683E+06	8,432E-03	1,239E+06	2,832E+12	relu	200	TRUE	(105, 85, 133)	0,001	0
9	0,99439	1,709E+06	8,564E-03	1,295E+06	2,922E+12	relu	100	TRUE	(171, 137, 177)	0,001	2
10	0,99427	1,727E+06	8,652E-03	1,270E+06	2,983E+12	relu	100	TRUE	(136, 97, 105)	0,001	9

Das beste MLP wurde mit den Testdaten geprüft. Die Ergebnisse dieses Tests sind in Abbildung 57 zu sehen.

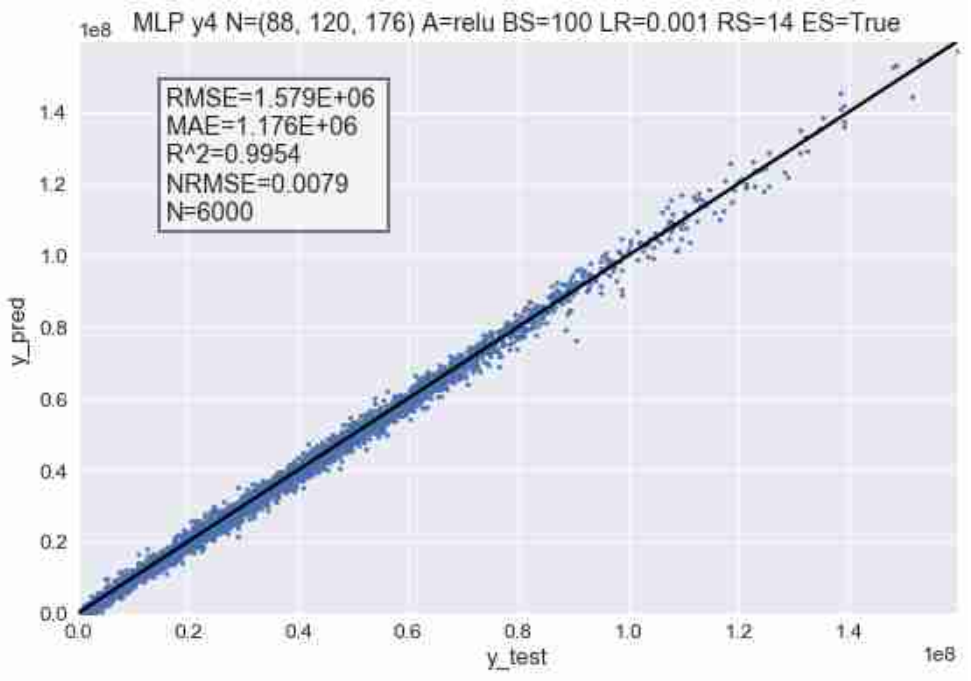


Abbildung 57: Ergebnisse der Testdaten für y4_tco gug.

3.2.3 Anwendung von Interpretationsmethoden

Nachdem für alle Zielgrößen die besten MLPs ausgewählt wurden, können nun die in Abschnitt 2.4 (S. 56 ff.) beschriebenen Interpretationsmethoden auf die Modelle angewendet werden. Im ersten Schritt soll bestimmt werden, welche Prädiktoren für die Schätzung der jeweiligen Zielgröße am wichtigsten sind. Die geschieht über die Berechnung der Relative Importance (RI). Anschließend werden die wichtigsten Prädiktoren über PDP und ICE weiter untersucht.

Sämtliche PDP und ICE dieser Arbeit wurden mit dem Python Paket *PDPBox* (0.2.0) erstellt[56]. Bei den PDP und ICE-Plots gilt es zu beachten, dass sowohl die Prädiktoren als auch die Zielgrößen im Intervall $[0; 1]$ bleiben. Dies ist hier auch kein Problem, da ohnehin nicht die absoluten Werte der betrachteten Größen von Interesse sind, sondern viel mehr deren Verläufe und Beziehungen untereinander. Zum Umrechnen kann jedoch für die Zielgrößen die Tabelle 7 auf S. 113 herangezogen werden. Für die Prädiktoren gilt Tabelle 3 auf S. 92.

[56] Eine Dokumentation ist online unter https://pdpbox.readthedocs.io/en/latest/ zu finden.

Alle erstellten Grafiken (auch die hier nicht im Detail behandelten) sind im Datenanhang hinterlegt:
„03_Analyse_von_Wirkzusammenhängen\01_Konventionelles_Kommissionieren_gassen ungebunden \ 03_Interpretation“

Investitionskosten (y1_invest)

Zur Berechnung der RI soll hier der Olden-Algorithmus (Connection Weights Approach) verwendet werden. Quellcode 8 zeigt die Implementierung des Algorithmus in Python. Aus der Multiplikation der Gewichtungen zwischen den Schichten des MLPs ergibt sich ein Vektor mit 18 Einträgen. Dieser Vektor beinhaltet für jeden Prädiktor die jeweilige RI. Da hier die Bias-Werte des Modells nicht berücksichtigt werden, schwanken die absoluten Werte der RI von MLP zu MLP. Aufgrund dessen werden die RI-Werte transformiert, sodass ihre absoluten Werte in der Summe $= 1$ ergeben (Zeile 355-357). Dadurch erhält man für jeden Prädiktor einen prozentualen Wert, welcher aussagt, wie stark die Zielgröße von ihm abhängt. Das Vorzeichen zeigt an, ob der Prädiktor das Ergebnis eher anhebt oder drückt.

```
345.    def mlp_olden(model, x_label, ind, model_config, plot, save):
346.        #Matrizenmultiplikation der Connection Weights (CW)
347.        #CW zwischen Input und erster Hidden-Layer
348.        cw = np.array(model.coefs_[0])
349.
350.        #Matrizenmultiplikation mit den übrigen CW des MLP
351.        for i in range(1, model.n_layers_ - 1):
352.            cw = np.matmul(cw, np.array(model.coefs_[i]))
353.
354.        #Normalisieren der RI-Einträge
355.        cw_sum = sum(np.absolute(cw))
356.        scaler = lambda x: (x/cw_sum)
357.        vfunc = np.vectorize(scaler)
358.
359.        #Fertiger Vektor mit RI der einzelnen Prädiktoren
360.        cw = vfunc(cw)
361.        cw = cw.ravel()
```

Quellcode 8: Implementierung des Olden-Algorithmus

Abbildung 58 zeigt die RI für die Zielgröße y1_invest in der gassenungebundenen Betrachtung. Offensichtlich sind die Anzahl der Gassen sowie die Anzahl der LM pro Gasse ausschlaggebend für die Investitionskosten.

Die LM-Ebenen, LM-Kosten, LM-Fachkapazität sowie die Kosten der ITK pro Fach liegen ähnlich nah beieinander, die Kosten für LHM sind vergleichsweise wenig entscheidend für die Investitionskosten.

Betrachtet man diese Faktoren gemeinsam, kann man davon ausgehen, dass sich die Investitionskosten für dieses Kommissioniersystem primär aus den Kosten für die einzelnen Fächer bilden. Dazu kommen die Kosten für die verwendeten LM. Diese Annahme kann dadurch begründet werden, da die LM-Ebenen/-Fachkapazität zusammen mit den LM im Kommissionierlager über die Anzahl der Gassen sowie die Anzahl der LM in Reihe die Summe der Fächer bilden. Die ITK-Kosten pro Fach und die LHM-Kosten bestimmen dazu, wie hoch die Kosten pro Fach sind. Eventuell wurde hier die Spannweite für den Faktor LHM-Kosten zu eng gewählt. Dies könnte ein Grund dafür sein, dass er im Vergleich zu den ITK-Kosten pro Fach weitaus unbedeutender für das Modell ist.

Die RI der Anzahl der Gassen sowie der Anzahl der LM in Reihe wird deshalb so hoch sein, da sie nicht nur in die Anzahl der Fächer hineinspielen, sondern auch für die Anzahl der LM im Lager verantwortlich sind. Zusammen mit den LM-Kosten ergibt sich eine weitere Kostenquelle für die Investition.

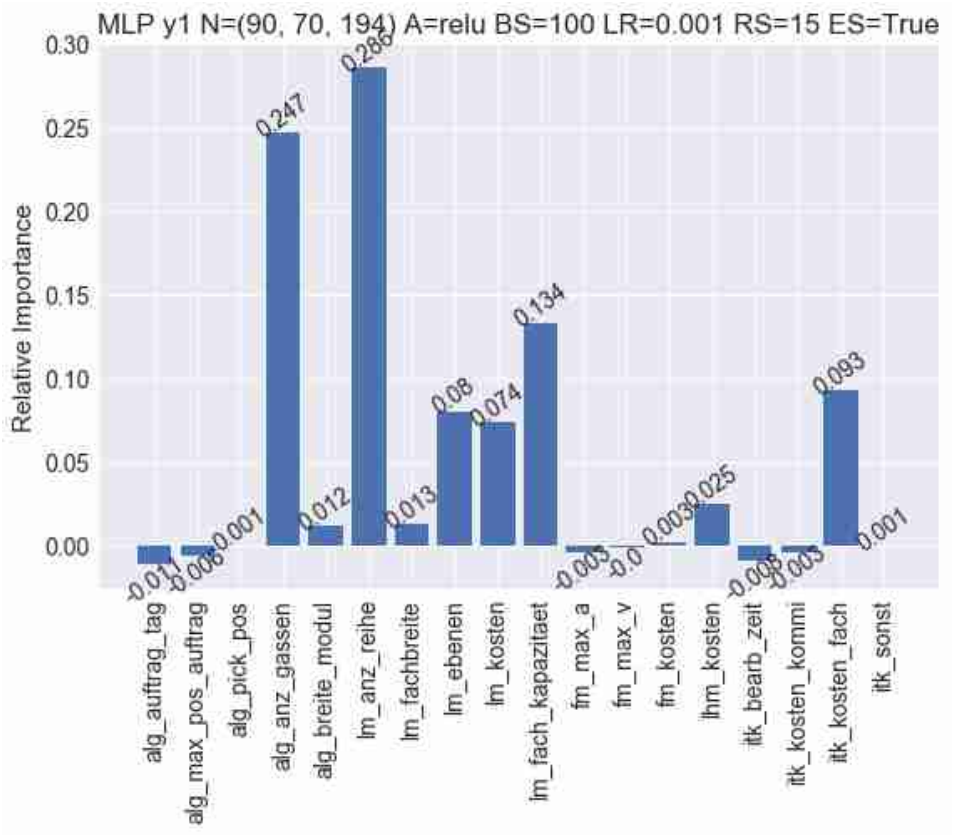


Abbildung 58: Relative Importance für y1_invest gug.

Interessanterweise scheinen die Anzahl bzw. die damit verbundenen Kosten für die benötigten FM, im Vergleich zu dem LM und Fächern, weitestgehend irrelevant für die gesamten Investitionskosten zu sein, da ihre RI sehr niedrig ist.

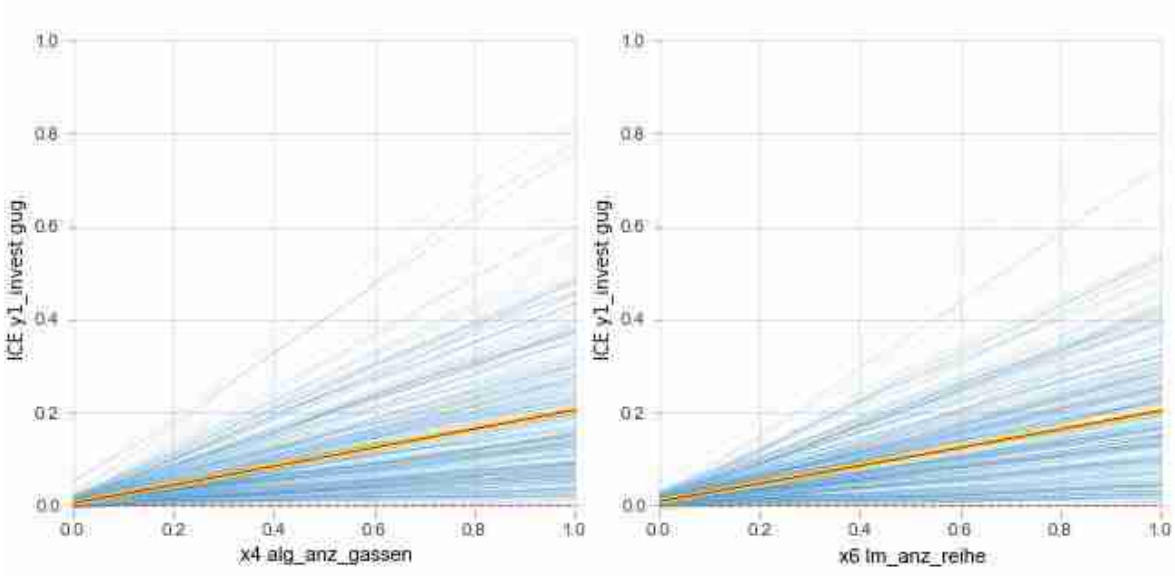


Abbildung 59: ICE-Plots für alg_anz_gasse und lm_anz_reihe gug.

Die Aufträge pro Tag wurden wohl vom MLP fehlinterpretiert, da nicht davon auszugehen ist, dass eine hohe Anzahl von täglichen Aufträgen dazu führt, dass in irgendeiner Weise die Investitionskosten verringert werden.

Die Abbildung 59 zeigt ICE-Plots für die beiden Prädiktoren mit der höchsten RI. Der Verlauf des PDP ist jeweils in Rot gehalten. Offensichtlich ist der Verlauf bei beiden Prädiktoren annähernd linear. Allerdings wird die Zielgröße bei beiden im höheren Bereich durch einen zusätzlichen Faktor verstärkt. Dies kann man anhand der sich auffächernden ICE-Linien erkennen. Hier zeigt sich der Vorteil des ICE gegenüber dem PDP. Diese Verstärkung hätte man so in einem PDP nicht ausfindig machen können.

Der Eindruck der Verstärkung spiegelt sich in Abbildung 60 wieder. Es zeigt sich, dass die beiden Prädiktoren, für sich allein genommen, keine hohen Investitionskosten produzieren können. Werden sie jedoch kombiniert, steigen die Kosten schnell an. Dies macht auch Sinn, da die Größe des Lagers durch eine Multiplikation dieser beiden Faktoren maßgeblich definiert ist.

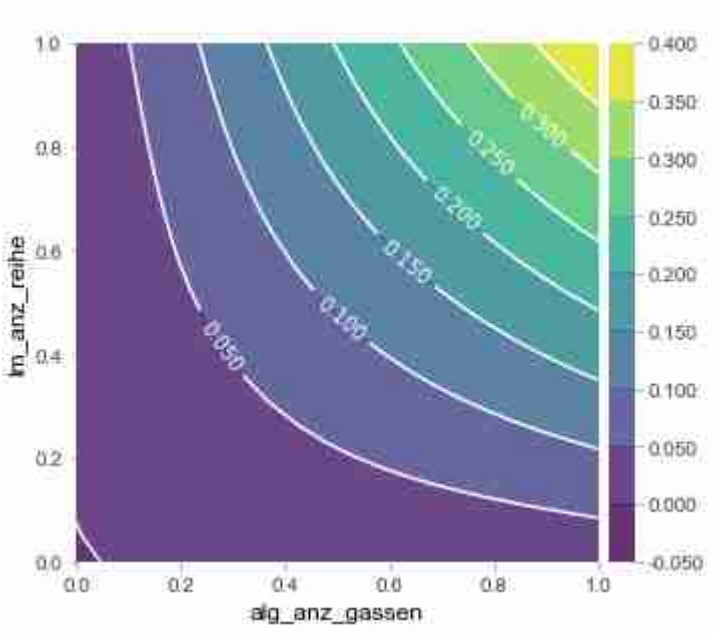


Abbildung 60: 3D-PDP alg_anz_gassen und lm_anz_reihe für y1_invest gug.

Ähnliche Effekte können auch bei den Prädiktoren lm_fach_kapazitaet und itk_kosten_fach ausgemacht werden: Siehe dazu Abbildung 61 und Abbildung 62.

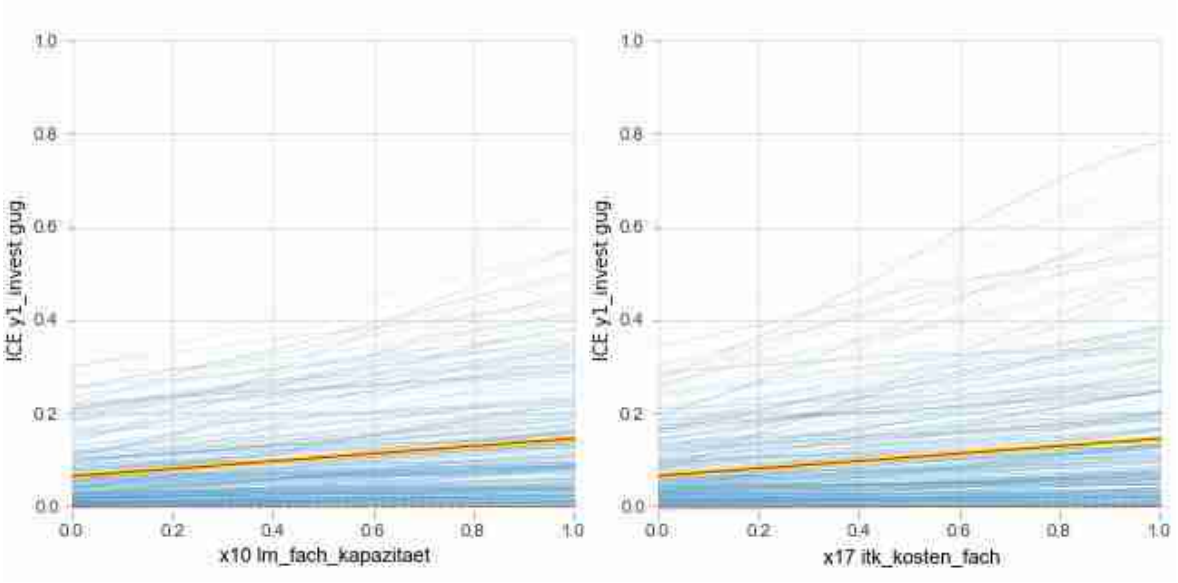


Abbildung 61: ICE-Plots für lm_fach_kapazitaet und itk_kosten_fach gug.

Aus den übrigen ICE-Plots konnte kein zusätzlicher Informationsgewinn gezogen werden, da sie entweder nur leicht linear ansteigen oder über den kompletten Verlauf hinweg konstant bleiben (also die Zielgröße nicht beeinflussen). Des Weiteren können alle PDPs zu den Investitionskosten (y1_invest) im Anhang A-10 (S. 210 f.) eingesehen werden.

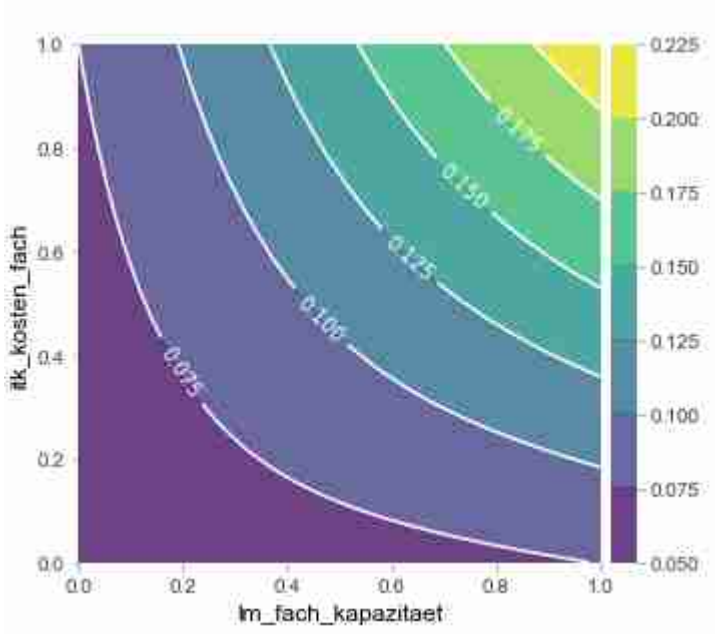


Abbildung 62: 3D-PDP lm_fach_kapazitaet und itk_kosten_fach für y1_invest gug.

Betriebskosten (y2_betrieb)

Beim Betrachten der RI für die Zielgröße Betriebskosten (Abbildung 63) fällt direkt auf, dass die Anzahl der Gassen zusammen mit der Anzahl der Aufträge pro Tag und der maximalen

Verfahrgeschwindigkeit des FM den größten Einfluss auf die Zielgröße aufweisen. Zudem erzeugen eine hohe Geschwindigkeit sowie eine hohe Beschleunigung des FM eine Abnahme der Betriebskosten. Zusammen mit der relativ hohen Bedeutung der Bearbeitungszeit spricht dies dafür, dass die Personalkosten bzw. die Kosten für die Kommissionierer einen sehr großen Anteil an den Betriebskosten ausmachen. Je schneller ein Kommissionierer verfahren kann, desto mehr Kommissionieraufträge kann er in der gleichen Zeit abarbeiten. Dementsprechend werden bei hoher Verfahrgeschwindigkeit weniger Kommissionierer benötigt, um die gleiche Kommissionierleistung bereitzustellen.

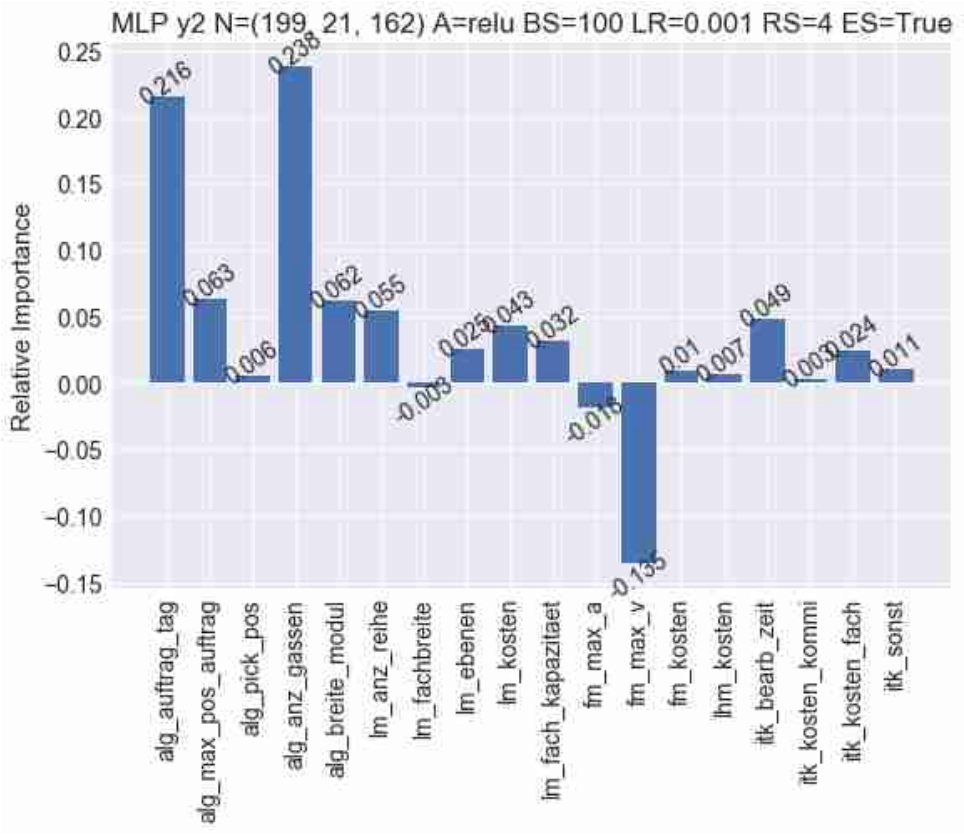


Abbildung 63: Relative Importance für y2_betrieb gug.

Betrachtet man die ICE-Plots für die Fortbewegung des FM, fällt auf, dass der Verlauf der Betriebskosten zunächst abnimmt und dann schließlich abflacht. Interessanterweise scheint hier eine Sättigung bei der Geschwindigkeit im Bereich von $0{,}3 \cong 0{,}9\,{}^{m}/_{S}$ zu existieren.

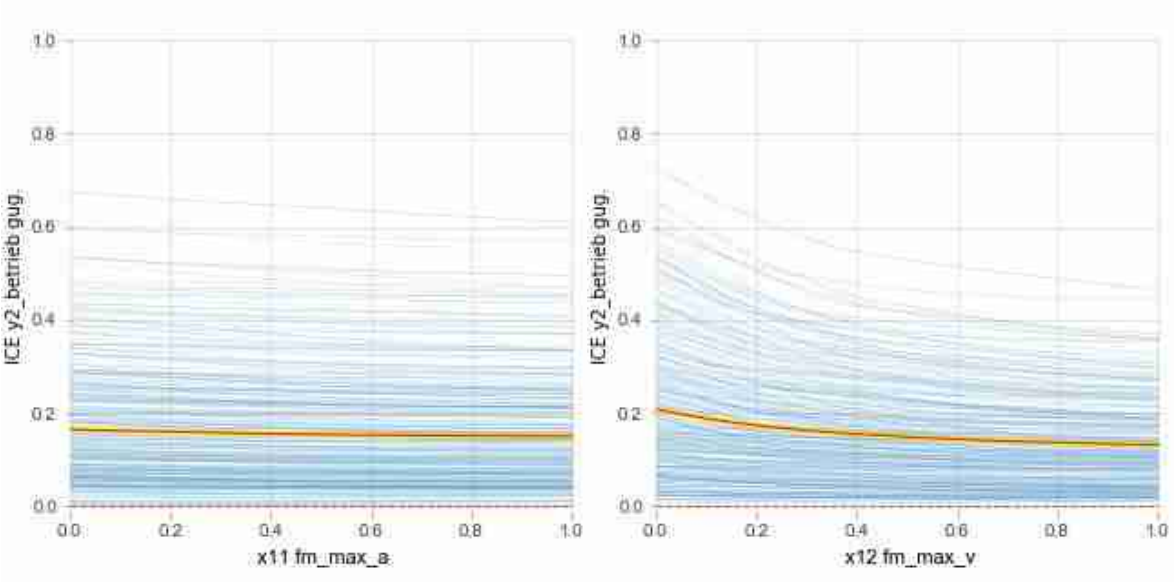


Abbildung 64: ICE-Plots für fm_max_a und fm_max_v gug.

Bei der Betrachtung der c-ICE-Plots in Abbildung 65 erkennt man, dass die Betriebskosten durchschnittlich um $0{,}07 \cong 1.400.000€$ gesenkt werden können, wenn die FM-Geschwindigkeit von der hier betrachteten Mindestgeschwindigkeit $\left(0{,}5\,\frac{m}{s}\right)$ auf die maximal mögliche Geschwindigkeit $\left(1{,}7\,\frac{m}{s}\right)$ gesteigert wird. Bei $0{,}4 \cong 1\,\frac{m}{s}$ können allerdings schon durchschnittlich $0{,}05 \cong 980.000€$ eingespart werden. Dementsprechend sollte das FM mindestens mit einer Geschwindigkeit von $1\,\frac{m}{s} \cong 3{,}6\,\frac{km}{h}$ verfahren, um gute Betriebskosteneinsparungen zu erreichen. Der Einfluss der Beschleunigung erscheint hier nur ca. $\frac{1}{5}$ so stark zu sein wie der der Geschwindigkeit.

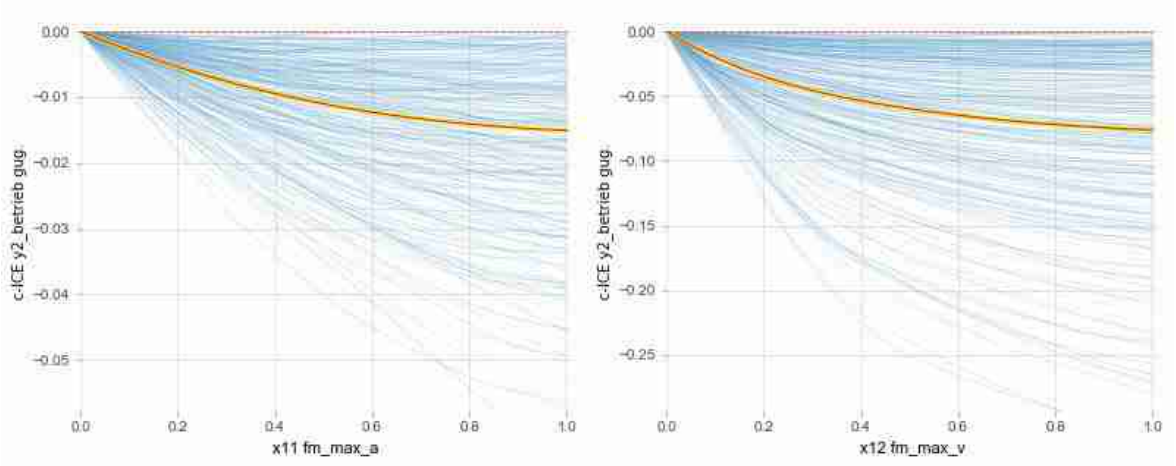


Abbildung 65: c-ICE für fm_max_a und fm_max_v gug.

Auch hier können wiederum die übrigen ICE-Plots im Datenanhang eingesehen werden. Die PDPs sind in A-11 (S. 212 f.) zu finden. Sie sind wiederum entweder linear ansteigend oder konstant, wie man es aus der RI erwarten konnte.

Anzahl Kommissionierer (y3_anz_kommi)

Wie sich in Abbildung 66 zeigt, stimmen die RI der Betriebskosten mit denen der Anzahl der benötigten Kommissionierer weitestgehend überein. Die Verfahrgeschwindigkeit des FM sowie die Anzahl der Aufträge pro Tag fallen hier nur noch stärker ins Gewicht, wohingegen die Kosten-Prädiktoren abgefallen sind. Auffällig ist, dass die Bearbeitungszeit ebenfalls einen sehr niedrigen RI-Wert aufweist. Rein logisch ist dies nicht zu erklären, es handelt sich vermutlich um eine Fehlinterpretation des Metamodells.

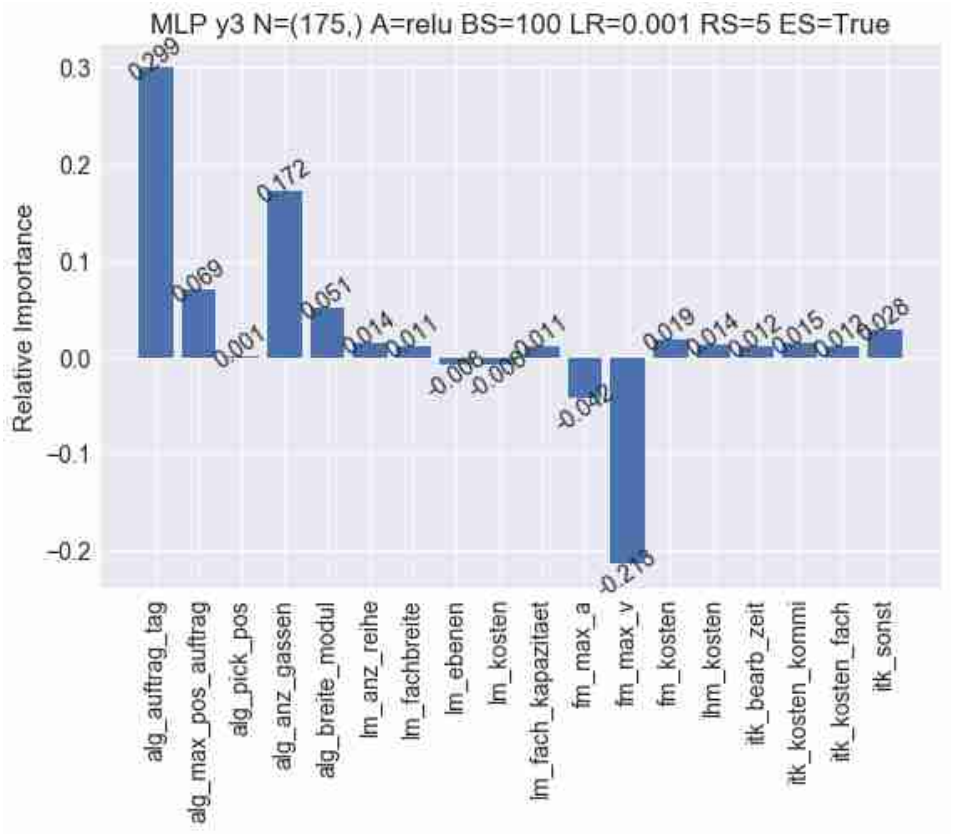


Abbildung 66: Relative Importance für y3_anz_kommi gug.

Schaut man sich jedoch Abbildung 67 an, lässt sich feststellen, dass die Bearbeitungszeit sehr wohl einen Einfluss auf die Anzahl der Kommissionierer hat. Zum Vergleich ist die Anzahl der Aufträge pro Tag ebenfalls abgetragen, welche eine sehr hohe RI aufweist.

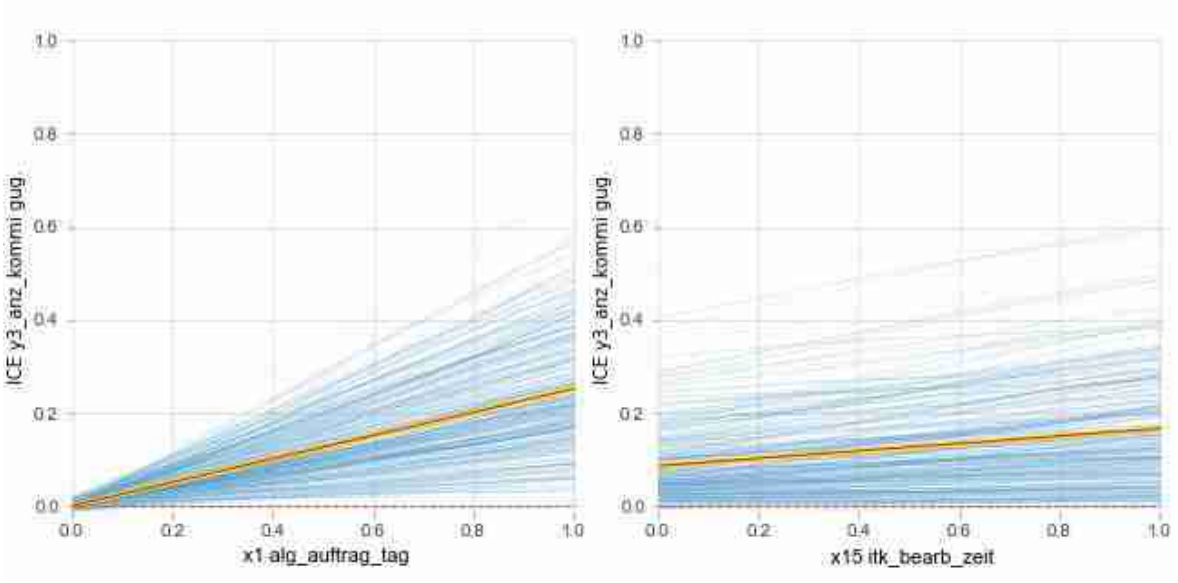


Abbildung 67: ICE-Plots für alg_auftrag_tag und itk_bearb_zeit gug.

Beim Vergleich der c-ICE für die Beschleunigung/Geschwindigkeit des FM mit denen aus Abbildung 65 ist festzustellen, dass sie sowohl vom Verlauf als auch von den relativen Zahlenwerten nahezu identisch sind. Dementsprechend wird hier auf die Interpretation im vorigen Abschnitt verwiesen.

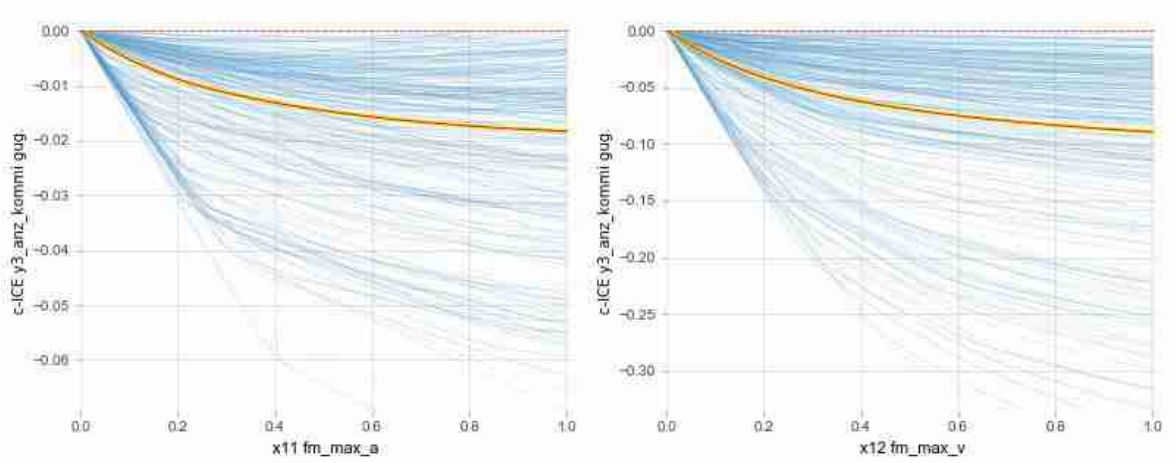


Abbildung 68: c-ICE für fm_max_a und fm_max_v

Interessant ist nun, inwiefern die Dimensionierung des Kommissioniersystems in die Anzahl der benötigten Kommissionierer hineinspielt. Die Dimensionierung ist durch die Anzahl der Gassen und die Länge der einzelnen Gassen definiert, wobei sich die Gassenlänge aus der Fachbreite sowie der Anzahl der LM pro Regaleinheit berechnet (siehe Abbildung 69). Wie man in dem 3D-PDP für die Gassenlänge erkennen kann, ist die Fachbreite eher zweitrangig gegenüber der LM in Reihe. Dies mag damit zusammenhängen, dass die Spannweite für

diesen Prädiktor recht eng ist. Vergleicht man die Gassenlänge mit der Anzahl der Gassen, zeigt sich, dass hier vor allem die Anzahl der Gassen für die benötigte Anzahl der Kommissionierer ausschlaggebend ist, da die Spannweite des 3D-PDP lediglich von 0,122 bis 0,136 geht, wohingegen die des PDP im ICE-Plot von ca. 0,08 bis 0,2 reicht.

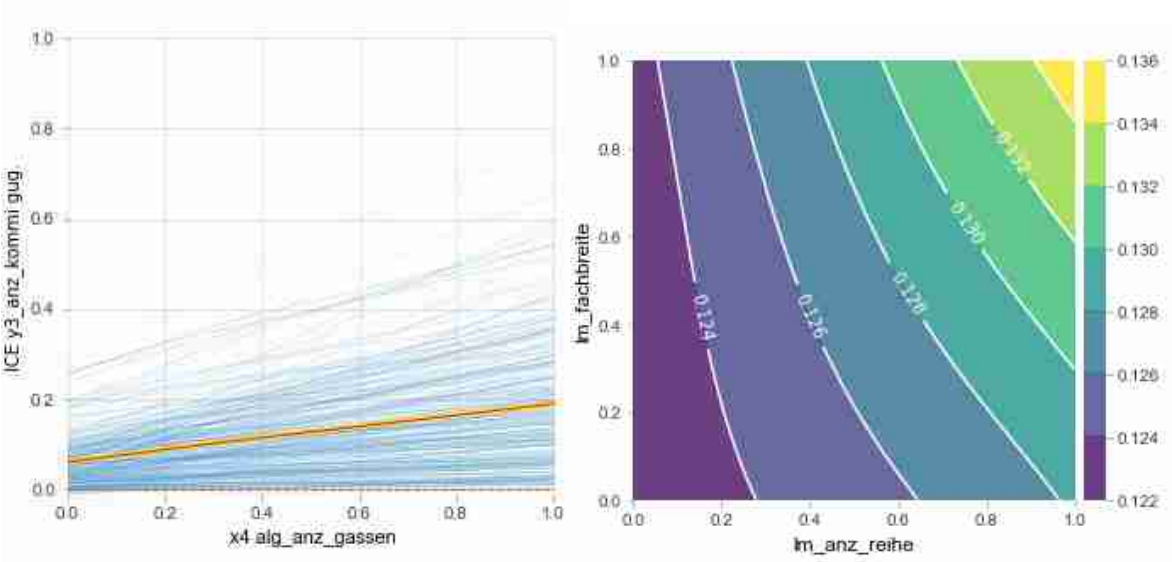


Abbildung 69: ICE für alg_anz_gassen und 3D-PDP Gassenlänge für y3_anz_kommi gug.

Die übrigen ICE-Plots sind im Datenanhang zu finden. Die PDPs in A-12 ab S. 214.

TCO (y4_tco)

Die TCO berechnet sich hier aus den Investitionskosten und den Betriebskosten über 10 Jahre. Dementsprechend kann davon ausgegangen werden, dass die RI des TCO der der Betriebskosten sehr ähnlich sein wird. In Abbildung 70 wird diese Annahme weitestgehend bestätigt. Über die RI des TCO wird allerdings der Einfluss der Bearbeitungszeit wesentlich höher angegeben als die RI der Betriebskosten bzw. Anzahl der Kommissionierer. Die Verfahrgeschwindigkeit sowie die Kapazität des FM (über alg_max_pos_auftrag) werden hier ebenfalls als wichtig eingeschätzt.

Die Anschaffungskosten für ITK (abgesehen von denen pro Fach) und die FM-Kosten schlagen bei dem TCO offensichtlich überhaupt nicht ins Gewicht, was so nicht zu erwarten war.

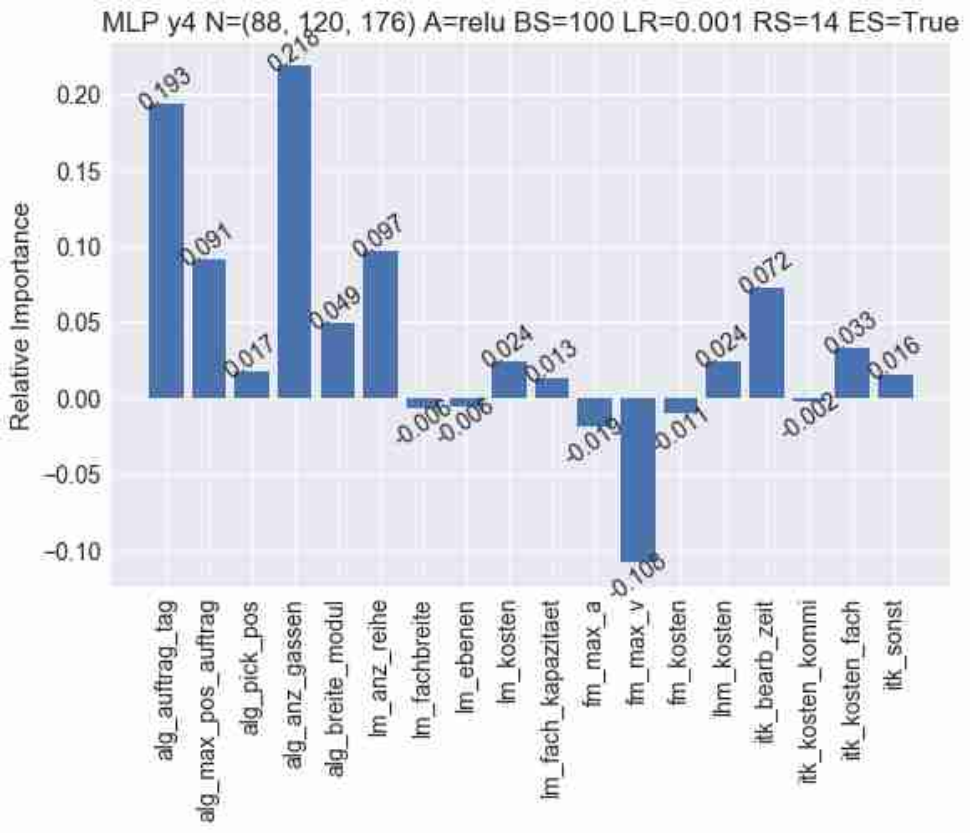


Abbildung 70: Relative Importance für y4_tco gug.

Der 3D-PDP in Abbildung 71 greift die beiden Prädiktoren auf, die laut der RI (Abbildung 70) die Leistung eines Kommissionierers maßgeblich beeinflussen. Es zeigt sich, dass bei einer hohen Geschwindigkeit und gleichzeitig niedriger Bearbeitungszeit der TCO vergleichsweise gering ist. Auch hier ist wieder die Schwelle bei der Verfahrgeschwindigkeit bei $0{,}4 \cong 1\frac{m}{s}$ zu erkennen.

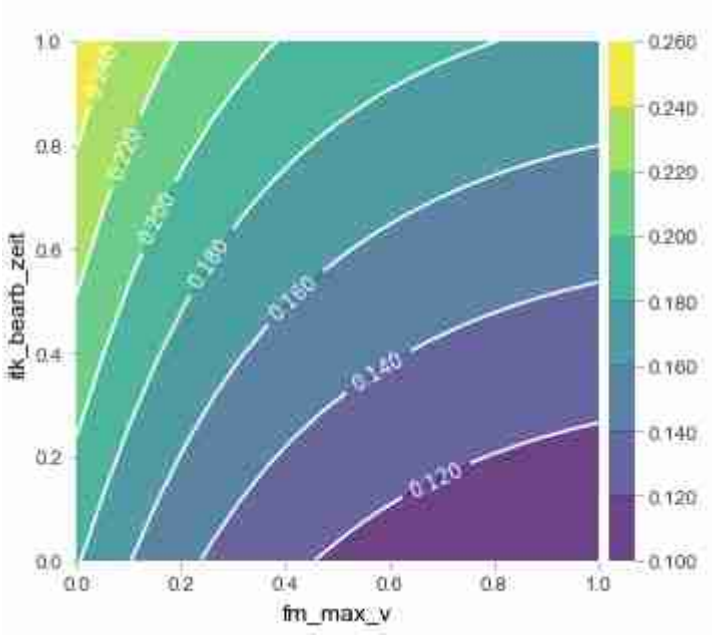


Abbildung 71: 3D-PDP fm_max_v und itk_bearb_zeit für y4_tco gug.

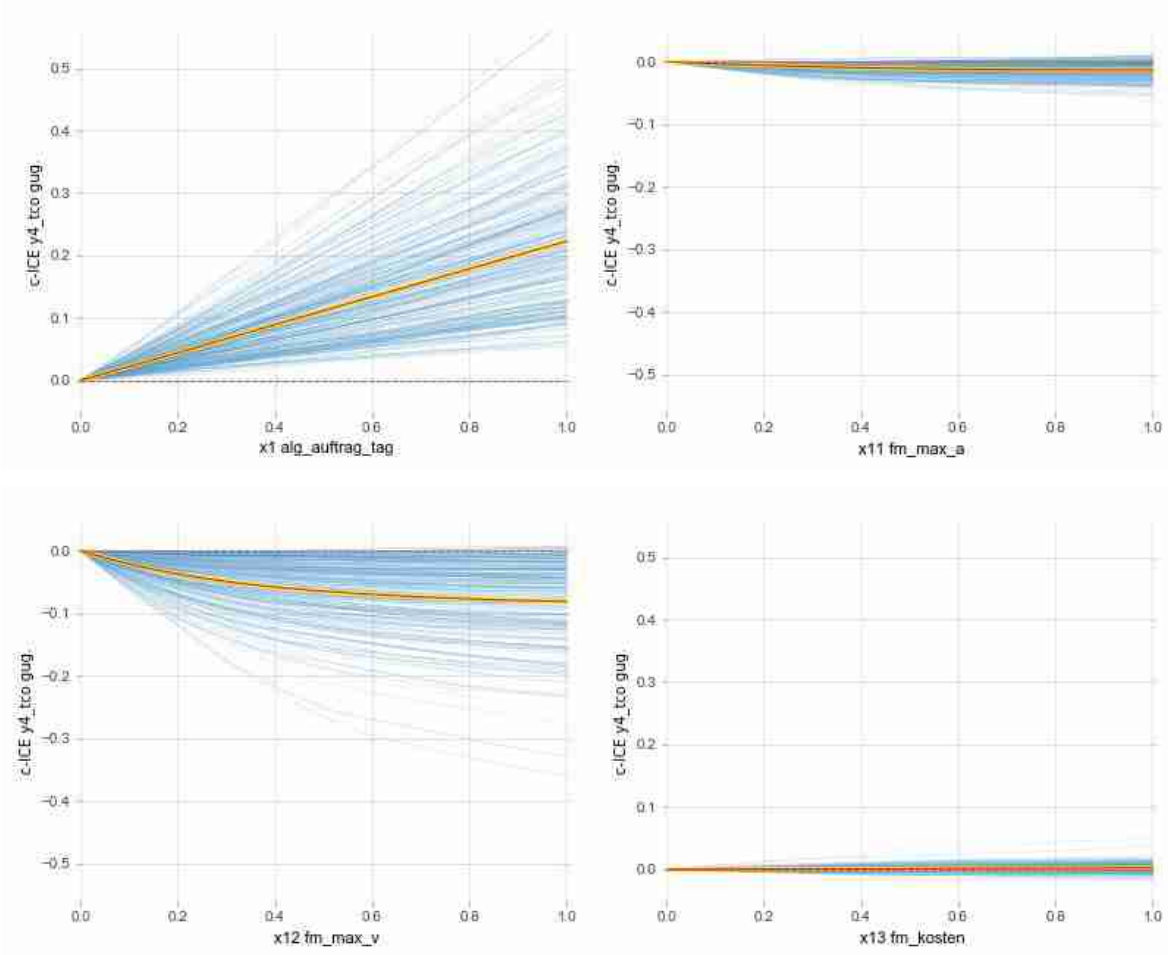


Abbildung 72: c-ICE für FM gug.

Betrachtet man die Prädiktoren, welche das FM repräsentieren, genauer, kann festgestellt werden, dass die Kosten des FM nur sehr geringe Auswirkungen auf den TCO haben. Im Gegensatz dazu spielt die Beschleunigung und insbesondere die Geschwindigkeit eine wesentliche Rolle bei der Bildung des TCO (Abbildung 72). Die Kapazität des FM sollte als Einflussgröße des TCO jedoch nicht außer Acht gelassen werden. Ist sie im Vergleich zur durchschnittlichen Anzahl von Positionen pro Auftrag zu gering, müssen dementsprechend mehr Kommissionieraufträge im Allgemeinen bedient werden. Dadurch kann es dazu kommen, dass mögliche Einsparungen durch eine höhere Geschwindigkeit des FM wieder kompensiert werden. Dementsprechend sollte bei der Wahl des FM nicht auf den Anschaffungspreis geachtet werden, sondern auf die hier angesprochenen technischen Daten.

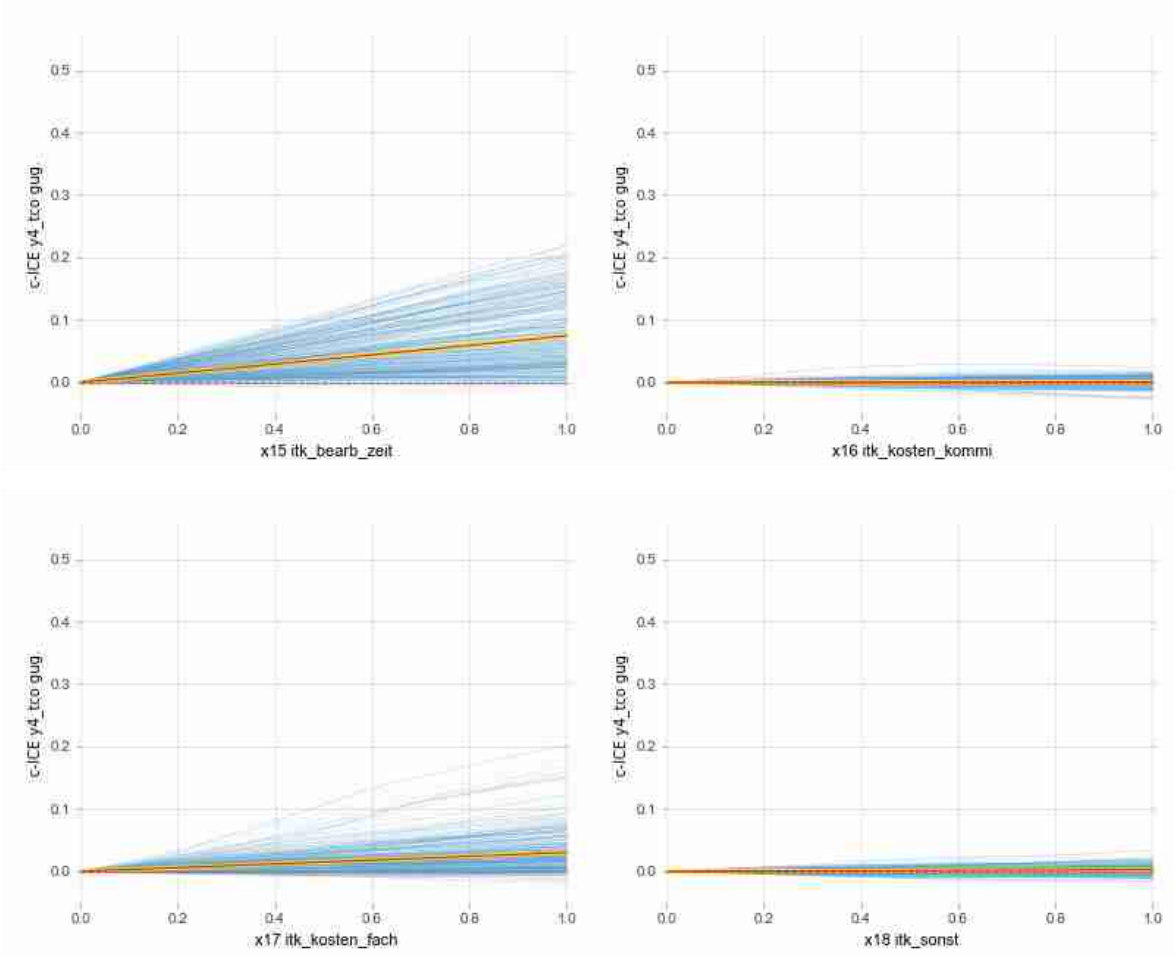


Abbildung 73: c-ICE für ITK gug.

Bei der ITK sieht es ähnlich aus (Abbildung 73). Die Kosten pro Kommissionierer und die sonstigen Kosten sind eher zweitrangig. Wichtiger ist jedoch eine möglichst kurze Bearbeitungszeit und nicht allzu hohe Kosten pro Lagerfach. Die Bearbeitungszeit hebt den TCO durchschnittlich um $0{,}08 \cong 1{,}596 \cdot 10^7$ Euro an.

Abgesehen davon sind keine nennenswerten neuen Erkenntnisse aus den PDP bzw. ICE zu ziehen, da sie auch hier entweder monoton ansteigen oder weitestgehend konstant verlaufen.

Die übrigen ICE-Plots sind im Datenanhang zu finden. Alle PDPs für die Zielgröße y4_tco sind in A-13 ab S. 216 aufgelistet.

3.3 Gassengebundenes konventionelles Kommissionieren

Die zweite hier betrachtete Variante des konventionellen Kommissionierens stellt die gassengebundene Betrachtung (gg.) dar. Der verwendete Python-Code kann im Datenanhang unter „03_Analyse_von_Wirkzusammenhängen\02_Konventionelles_Kommissionieren_gassengebunden\02_Modell\MA_Model_Interpret_gg_FINAL.py“ in voller Länge gefunden werden.

3.3.1 Versuchsplanung

Die Versuchsplanung hier ist weitestgehend identisch zu der gassenungebundenen Betrachtung. Bei den Zielgrößen handelt es sich wiederum im die bereits bekannten: Investitionskosten, Betriebskosten, Anzahl der Kommissionierer sowie der TCO. Als quantitative Variablen werden hier dieselben 18 Faktoren verwendet wie in der gassenungebundenen Betrachtung. Zum einen erleichtert dies die Vergleichbarkeit zwischen den Betrachtungsweisen und zum anderen sind hier dieselben Faktoren für die vier Zielgrößen relevant. Die in Tabelle 3 (S. 92) festgelegten Spannweiten für diese Faktoren bleiben ebenfalls bestehen.

Allerdings werden für den Vollfaktorplan zur Identifikation der besten kategorischen Variablen lediglich 64 Versuche benötigt. Dies liegt daran, dass hier nur zwei Wegstrategien vorhanden sind. Bei der gassenungebundenen Betrachtung konnten insgesamt 10

verschiedene Kombinationen für die Wegstrategie gebildet werden (siehe Abbildung 43, S. 95). Die 18 quantitativen Faktoren wurden für die Berechnung des Vollfaktorplans wiederum auf die Mittel ihrer Spannweiten festgesetzt. Der komplette Vollfaktorplan für die gg. Betrachtung kann im Datenanhang[57] eingesehen werden.

Tabelle 12: Durchschnittswerte der Zielgrößen des Vollfaktorplans gg.

Investitionskosten	Betriebskosten	Anzahl Kommissionierer	TCO
$\bar{y}_1 = 1{,}54 \cdot 10^6$	$\bar{y}_2 = 1{,}77 \cdot 10^6$	$\bar{y}_3 = 9{,}18$	$\bar{y}_4 = 1{,}92 \cdot 10^7$

Die durchschnittlichen Werte der vier Zielgrößen sind in Tabelle 12 zu sehen. Diese sind z.T. im Vergleich zu der gug. Betrachtung wesentlich niedriger. Die Investitionskosten sind fast identisch zur gug. Betrachtung. Allerdings sind die durchschnittlichen Betriebskosten nur halb so groß und die durchschnittliche Anzahl der Kommissionierer nur $\frac{1}{3}$; der TCO ist in etwa halb so groß.

In Abbildung 74 sind die Effektdiagramme für das Lager-Layout, die Verteilung der Artikel im Lager, die Ansteuerung sowie die Wegstrategie abgetragen. Auch hier schneidet das Zentralganglayout wesentlich besser ab als das Kopfganglayout. Noch deutlicher fällt dies bei der Verteilung der Artikel im Lager aus: Hier ist ebenfalls eine Exponentialverteilung der Gleichverteilung vorzuziehen. Weiterhin macht die Art der Ansteuerung nun einen Unterschied. Der geordnete Fall ist durchschnittlich etwas besser als der Ungeordnete (ca. 300.000 Euro). Bei der Wegstrategie zeigt sich, dass für den Fall, dass Start und Ziel nicht identisch sind, durchschnittlich ca. 350.000 Euro eingespart werden können.

[57] Siehe: „03_Analyse_von_Wirkzusammenhängen\02_Konventionelles_Kommissionieren_gassen gebunden \01_Versuchsplanung\Tool_Input\Vorversuch“

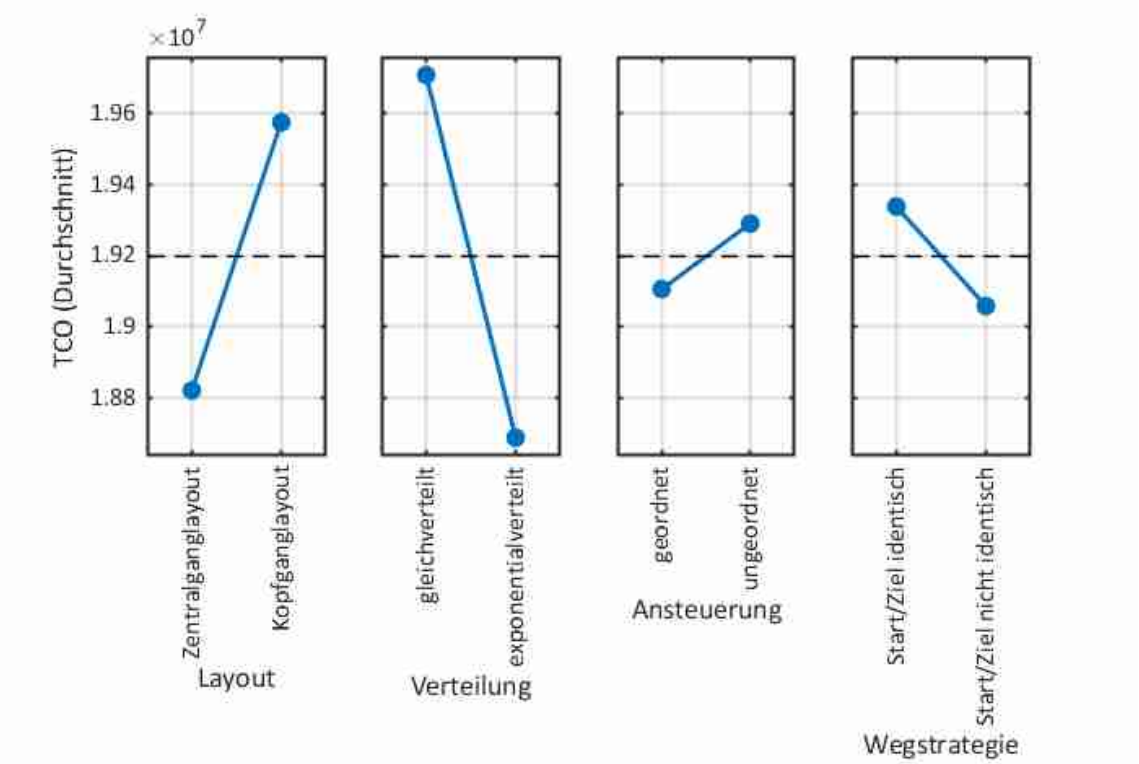


Abbildung 74: Effektdiagramme für Layout, Verteilung, Ansteuerung und Wegstrategie

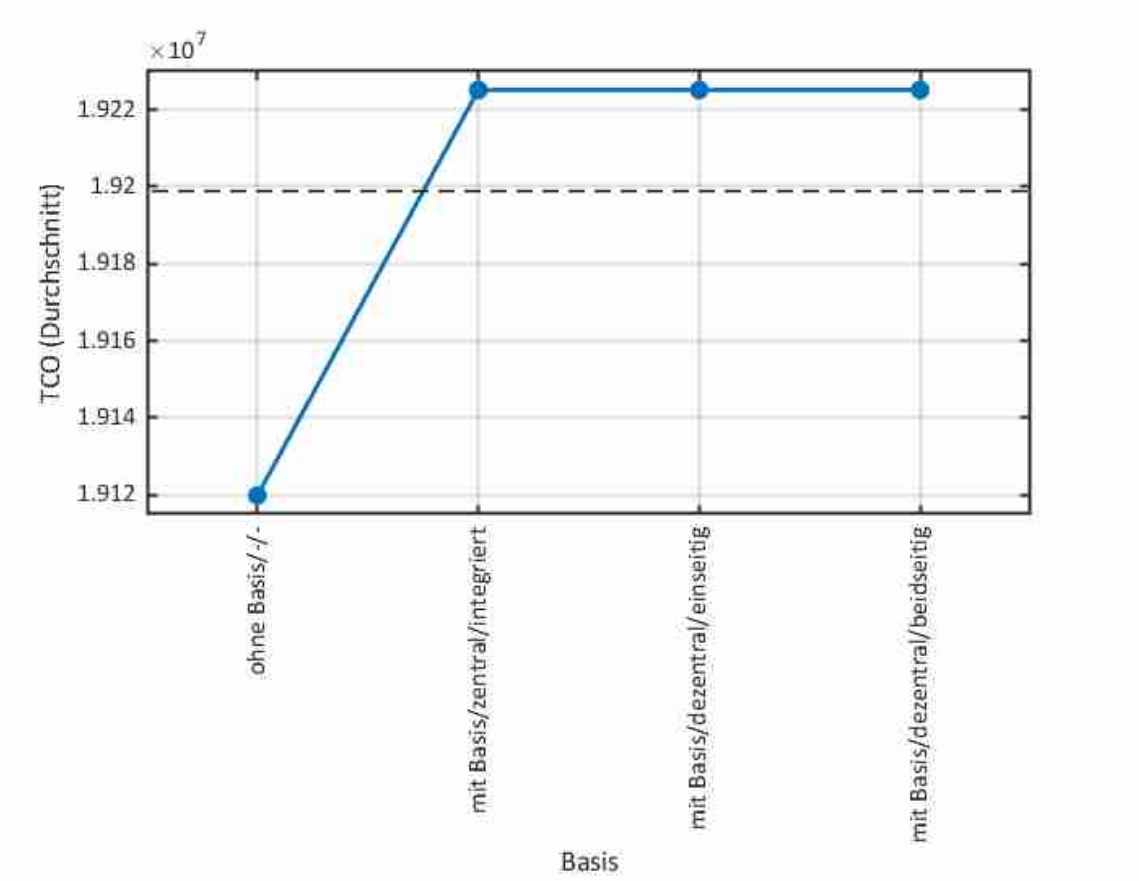


Abbildung 75: Effektdiagramm für Basis

Der Einsatz einer Basis ist auch hier (bzgl. des durchschnittlichen TCO) nicht ratsam (siehe Abbildung 75). Es macht auch offensichtlich keinen Unterschied, wo sich die Basis befindet. Es wird hier nur zwischen einer vorhandenen und nichtvorhandenen Basis differenziert.

Im nächsten Schritt werden die Zweifachwechselwirkungen der Faktoren untereinander untersucht. Diese sind in Abbildung 76 zu sehen.

Im Gegensatz zu denen in der gug. Betrachtung, sind hier einige Wechselwirkungen zwischen den Faktoren auszumachen:
Eine ungeordnete Ansteuerung sollte möglichst nicht mit einer Gleichverteilung der Artikel kombiniert werden. Diese ist im Vergleich zur ungeordneten Ansteuerung mit Exponentialverteilung durchschnittlich ca. 150.000 Euro teurer. Bei einer geordneten Ansteuerung liegt dieser Unterschied lediglich bei ca. 50.000 Euro.

Bei der Verteilung der Artikel im Lager fällt auf, dass es dort in zwei Fällen auf die Kombination mit anderen Faktoren ankommt, welche Einstellung vorzuziehen ist. Bei W01 ist eine Exponentialverteilung vorzuziehen, wohingegen bei W02 eine Gleichverteilung leicht besser ist. Das Gleiche trifft für A01 und A02 zu.

L	**Layout**
L01	Zentralganglayout
L01	Kopfganglayout
V	**Verteilung**
V01	gleichverteilt
V02	exponentialverteilt
A	**Ansteuerung**
A01	geordnet
A02	ungeordnet
B	**Basis**
B01	ohne Basis/-/-
B02	mit Basis/zentral/integriert
B03	mit Basis/dezentral/einseitig
B04	mit Basis/dezentral/beidseitig
W	**Wegstrategie**
W01	Start/Ziel identisch
W02	Start/Ziel nicht identisch

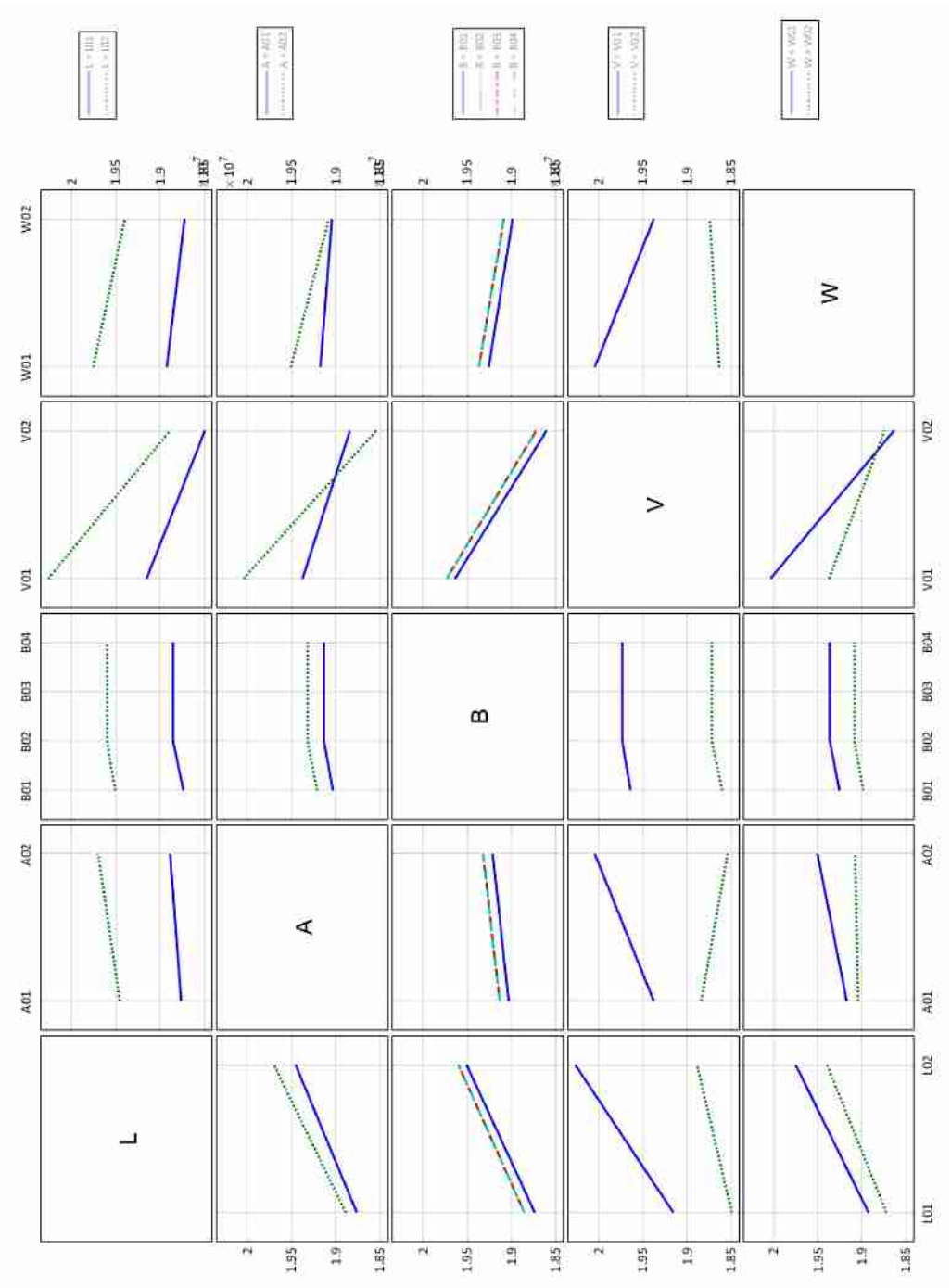


Abbildung 76: Wechselwirkungen gassengebundenes konventionelles Kommissionieren

Tabelle 13: Top 5 des Vollfaktorplans (gassengebunden)

Lager-Layout	Verteilung der Artikel im Lager	Art der Ansteuer-ung	Art der Betracht-ung	Wegstrategie	Geschwindig-keitsveränder-ung bei Gassen-wechsel	Kommiss-ionieren mit/ohne Basis	Lage der Basis	Zonen-durch-gang	TCO
Zentral-ganglayout	exponentialverteilt	ungeordnet	gassen-gebunden	Start/Ziel nicht identisch	-	ohne Basis	-	-	18.210.218,09
Zentral-ganglayout	exponentialverteilt	geordnet	gassen-gebunden	Start/Ziel identisch	-	ohne Basis	-	-	18.345.317,59
Zentral-ganglayout	exponentialverteilt	ungeordnet	gassen-gebunden	Start/Ziel identisch	-	ohne Basis	-	-	18.345.823,64
Zentral-ganglayout	exponentialverteilt	ungeordnet	gassen-gebunden	Start/Ziel nicht identisch	-	mit Basis	zentral	integriert	18.351.107,01
Zentral-ganglayout	exponentialverteilt	ungeordnet	gassen-gebunden	Start/Ziel nicht identisch	-	mit Basis	dezentral	einseitig	18.351.107,01

In Tabelle 13 sind die besten fünf Konfigurationen aus dem Vollfaktorplan aufgelistet. Wie man anhand des 4. und 5. Eintrages erkennen kann, spielt die Lage der Basis offensichtlich überhaupt keine Rolle für die Berechnung des TCO. Das Zentralgang-Layout sowie die Exponentialverteilung der Artikel im Lager dominieren hier klar. Die ungeordnete Ansteuerung ist zwar über alle Versuche hinweg im Durchschnitt die schlechtere Wahl, allerdings ist sie in vier von fünf der o.g. Fällen vertreten. Offensichtlich schwankt sie sehr stark in ihren Ausprägungen, was auch mit den Erkenntnissen aus den Wechselwirkungen zwischen Verteilung im Lager und der Ansteuerung übereinstimmt.

Tabelle 14: Konfiguration für gassengebundenes konventionelles Kommissionieren

Faktor	Faktorstufe
Lager-Layout	Zentralganglayout
Verteilung der Artikel im Lager	exponentialverteilt
Art der Ansteuerung	ungeordnet
Art der Betrachtung	gassengebunden
Wegstrategie	Start/Ziel nicht identisch
Geschwindigkeitsveränderung bei Gassenwechsel	-
Kommissionieren mit/ohne Basis	Ohne Basis
Lage der Basis	-
Zonendurchgang	-

In Tabelle 14 ist die ausgewählte Konfiguration für die gassengebundene Betrachtung zu sehen. Da die quantitativen Faktoren komplett identisch zu denen der gug. Betrachtung

sind, kann hier der gleiche Versuchsplan mit 30.000 Datenpunkten verwendet werden wie bei der gug. Betrachtung.

3.3.2 Metamodelltraining und -auswahl

Das Training sowie die Auswahl des besten Modells für jede Zielgröße erfolgt nach genau derselben Prozedur wie bei der gug. Betrachtung. Auf eine tiefergehende Beschreibung wird deshalb weitestgehend verzichtet. Die Spannweiten der vier betrachteten Zielgrößen sind in Tabelle 15 aufgelistet.

Tabelle 15: Spannweiten der vier Zielgrößen gg.

	y1_invest	y2_betrieb	y3_anz_kommi	y4_tco
Min.	3.000,00	1.690,08	$3{,}225 \cdot 10^{-3}$	19.900,80
Max.	$1{,}487 \cdot 10^{7}$	$9{,}573 \cdot 10^{6}$	56,19	$1{,}069 \cdot 10^{8}$

Die Bestimmung des geeigneten Regressionsmaßes und die Auswahl des besten MLPs erfolgt auch hier nach dem gleichen Muster wie bei der gassenungebundenen Betrachtung. Die einzelnen Graphen und Top 10 können bei Bedarf im Anhang eingesehen werden: A-6 (S. 206), A-7 (S. 207), A-8 (S. 208) und A-9 (S. 209).

3.3.3 Anwendung von Interpretationsmethoden

Auch hier sind die Achse der PDPs und ICE für die Zielgrößen und auch die Prädiktoren in ihren jeweiligen Spannweiten normalisiert worden. Zur Umrechnung aus dem Intervall $[0; 1]$ in die realen Werte kann für die Prädiktoren die Tabelle 3 auf S. 92 verwendet werden. Für die vier Zielgrößen sollte dementsprechend die Tabelle 15 auf S. 140 benutzt werden.

Auch hier können sämtliche Grafiken im Datenanhang gefunden werden. Siehe dazu: „03_Analyse_von_Wirkzusammenhängen\02_Konventionelles_Kommissionieren_gassengebunden\03_Interpretation“.

Investitionskosten (y1_invest)

Die RI der einzelnen Prädiktoren für die Zielgröße y1_invest ist in Abbildung 77 graphisch dargestellt. Die wichtigsten Einflussfaktoren sind hiernach die Anzahl der Gassen sowie die

Anzahl der LM in Reihe. Zusammen mit den LM-Kosten geben sie die Kosten für die LM des Kommissioniersystems vor. Darauf folgen die Prädiktoren, welcher direkt mit der Anzahl der Fächer bzw. LHM in Verbindung stehen: LM-Ebenen, Kapazität eines LM Faches und die ITK Kosten pro Fach.

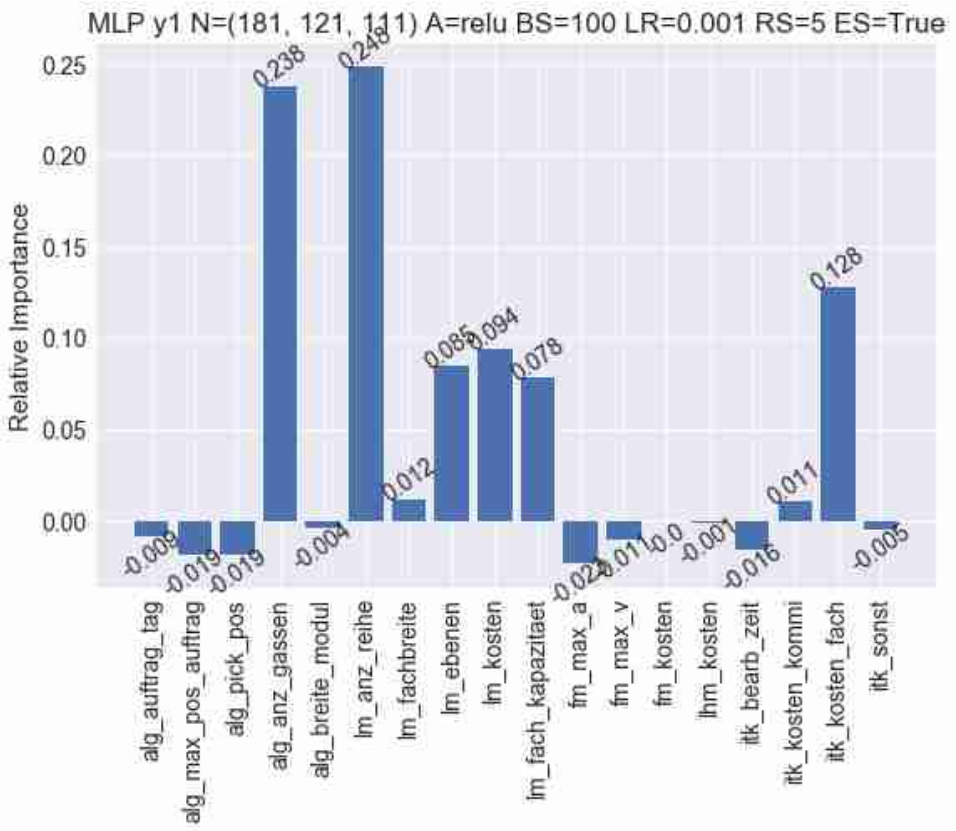


Abbildung 77: Relative Importance für y1_invest gg.

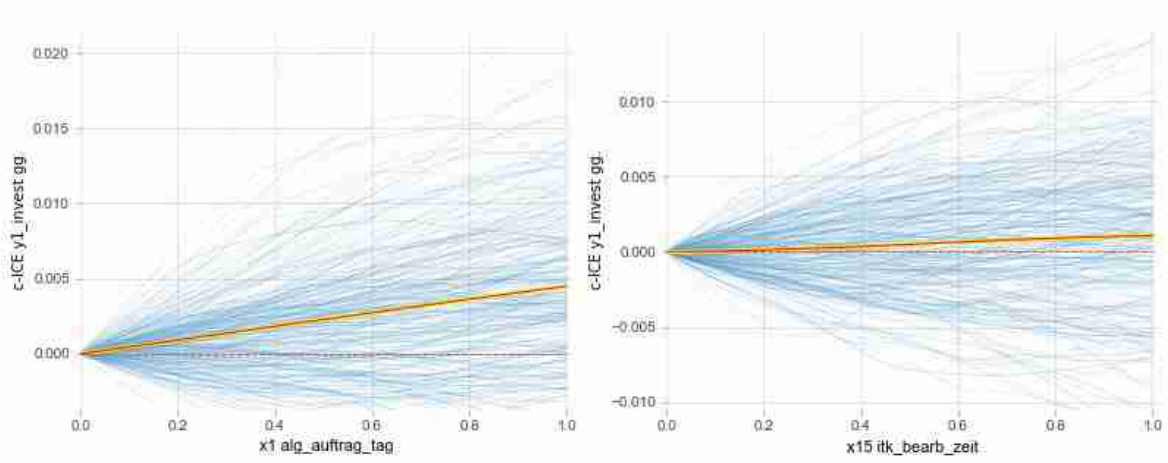


Abbildung 78: c-ICE-Plot für alg_auftrag_tag und itk_bearb_zeit gg.

Die negativen Werte, insbesondere bei Leistungsparametern wie der Anzahl der Aufträge pro Tag und der Picks pro Position sowie der Bearbeitungszeit sind sehr wahrscheinlich Fehlinterpretationen geschuldet. Um dies zu überprüfen, können c-ICE-Plots herangezogen werden. Abbildung 78 zeigt für zwei der benannten Prädiktoren, dass sie entgegen der RI die Zielgröße eher anheben. Wobei es zu erwähnen gilt, dass diese positive Beeinflussung der Investitionskosten durch die beiden Prädiktoren sehr gering ist.

Da der Aufbau der RI sowie die Spannweite der Investitionskosten in der gg. Betrachtung fast identisch zu der gug. Betrachtung sind, kann für weitere Erklärungen/Deutungen an dieser Stelle auf die Ausführungen ab S. 121 verwiesen werden. Alle ICE- und c-ICE-Plots können im Datenanhang eingesehen werden. Die PDPs zu allen Prädiktoren sind in A-14 (S. 218) abgebildet.

Betriebskosten (y2_betrieb)

Die Abbildung 79 zeigt die Relative Importance für die Betriebskosten in der gassengebundenen Betrachtung. Auffällig ist hier, dass die RI-Werte, im Vergleich zu den bisher behandelten RI, recht gleichmäßig auf alle Prädiktoren verteilt sind.

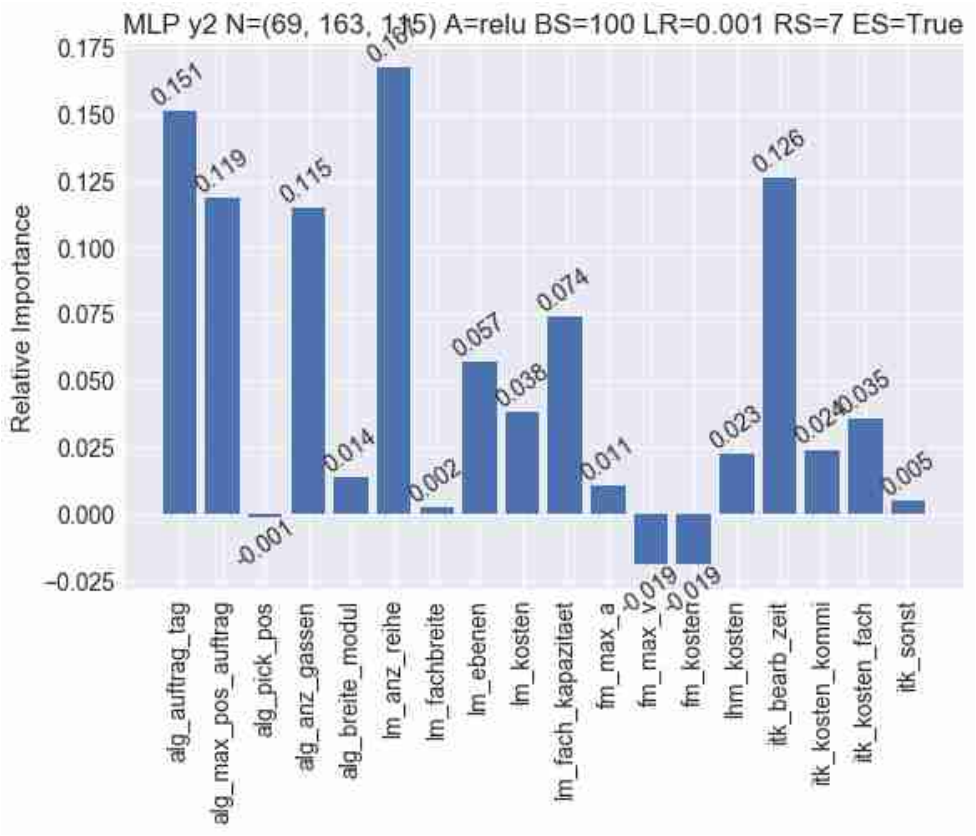


Abbildung 79: Relative Importance für y2_betrieb gg.

Die Beschleunigung und max. Geschwindigkeit des Kommissionierers scheinen hier keinen großen Einfluss auf die Betriebskosten zu besitzen. Dieser Eindruck wird durch die beiden c-ICE in Abbildung 80 bestätigt. Ebenfalls ist hier interessant, dass der Verlauf der maximalen Geschwindigkeit leicht exponentiell ins Negative zu verlaufen scheint, wobei er massiven Streuungen unterliegt; zudem ist dieser Effekt minimal ($0{,}002 \cong 19.150€$). Der Verlauf der Beschleunigung erscheint gesättigt zu werden bzw. weitestgehend linear zu verlaufen. Ebenfalls widerspricht er der Bewertung in der RI.

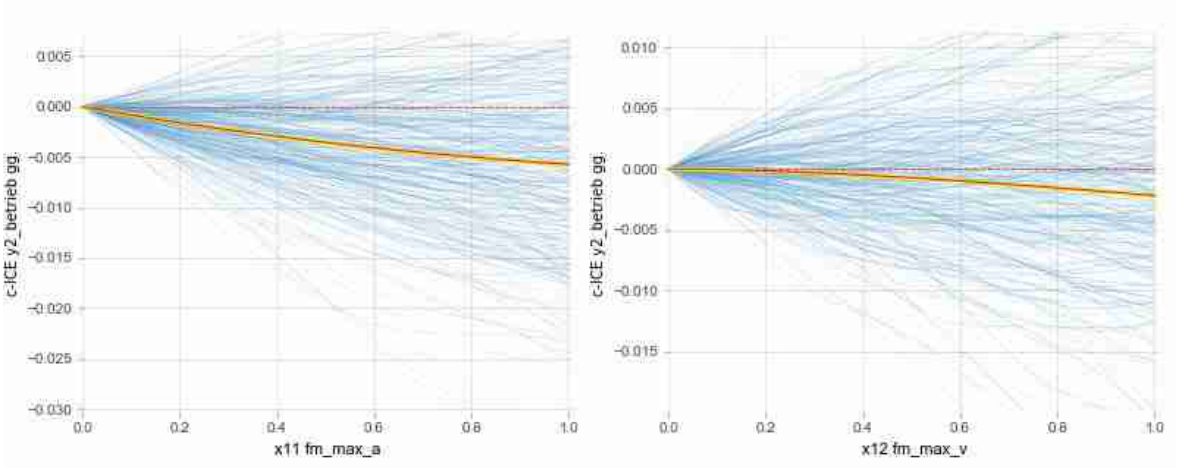


Abbildung 80: c-ICE für fm_max_a und fm_max_v gg.

Ein weiterer Punkt liegt in den Kosten der FM und LHM (siehe Abbildung 81). Es zeigt sich, dass die FM-Kosten die Betriebskosten nur sehr leicht anheben. Bei den LHM-Kosten sind Geraden mit unterschiedlicher Steigung zu erkennen, was dafürspricht, dass sie durch andere Faktoren verstärkt werden. Da sich die Kosten für die LHM aus den Eigenkosten sowie aus deren Anzahl ergeben, wird es sich hierbei wahrscheinlich um die Anzahl der LM und deren Kapazität handeln. Es sind hier wahrscheinlich ihre Abschreibungskosten, welche den Betriebskosten zugerechnet werden.

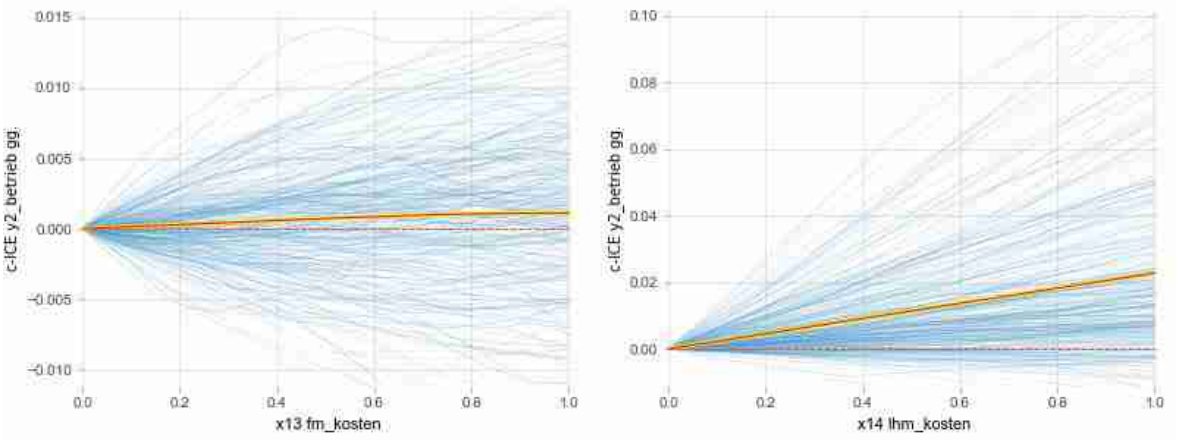


Abbildung 81: c-ICE für fm_kosten und lhm_kosten gg.

In Anbetracht der 3D-PDPs aus Abbildung 82 gewinnt man den Eindruck, dass die Eigenkosten der LHM hier eher nebensächlich sind. Die schiere Anzahl der LHM überwiegt

die Kosten des Einzelnen. Diese Einschätzung ist wiederum mit den Werten der RI kompatibel. Die übrigen PDPs sind in A-15 (S. 220 f.).

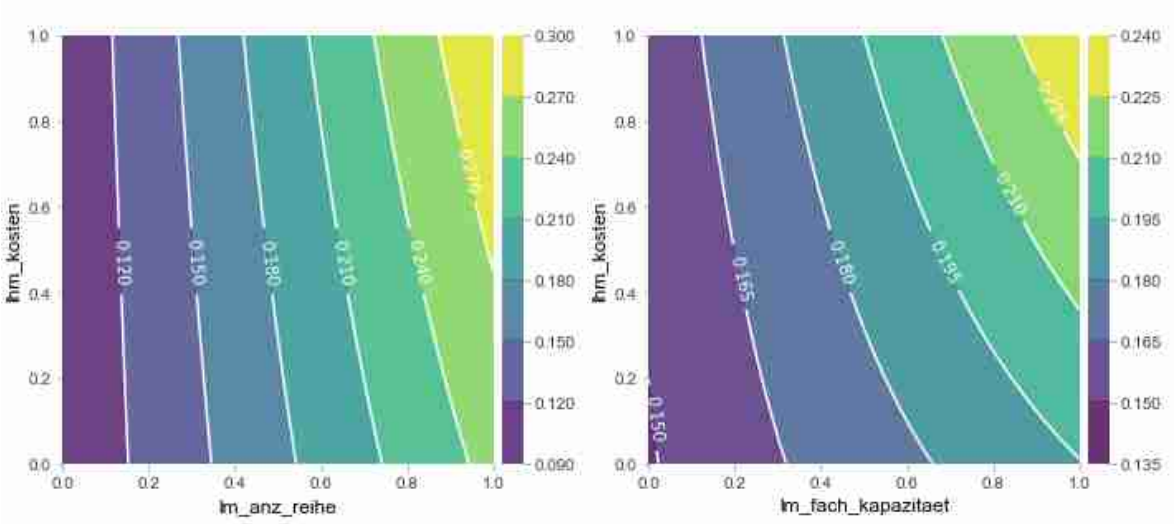


Abbildung 82: 3D-PDPs lhm_kosten mit lm_anz_reihe bzw. lm_fach_kapazitaet für y2_betrieb gg.

Anzahl Kommissionierer (y3_anz_kommi)

Die RI für die Anzahl der benötigten Kommissionierer ist in der gassengebundenen Betrachtung sehr eindeutig (Abbildung 83). Offensichtlich haben hier lediglich fünf der insgesamt 18 Prädiktoren einen Einfluss auf die Anzahl der Kommissionierer. Dabei spielt die Anzahl der Aufträge pro Tag, die Kapazität des Kommissionierers (über alg_max_pos_auftrag) sowie die Bearbeitungszeit die größte Rolle bei der Berechnung von y3_anz_kommi. Die Anzahl der LM in Reihe sowie die maximale Geschwindigkeit des FM sind ebenfalls für die Zielgröße von Bedeutung, jedoch den übrigen drei Prädiktoren untergeordnet. Alle anderen Prädiktoren sind laut der RI für die Berechnung der Zielgröße weitestgehend irrelevant.

Interessanterweise scheint hier die Bewegung des Kommissionierers weitestgehend irrelevant für die Anzahl der benötigten Kommissionierer zu sein. Lediglich die Bearbeitungszeit spielt in die Berechnung hinein. Betrachtet man den 3D-PDP aus Abbildung 84, bestätigt sich dieser Eindruck. Die Anzahl der Kommissionierer wird dort durch die Bearbeitungszeit dominiert. Ein Anstieg der Geschwindigkeit wirkt sich nur sehr minimal auf das Ergebnis aus, wohingegen schon kleine Änderungen der Bearbeitungszeit große Veränderungen in der Schätzung hervorrufen.

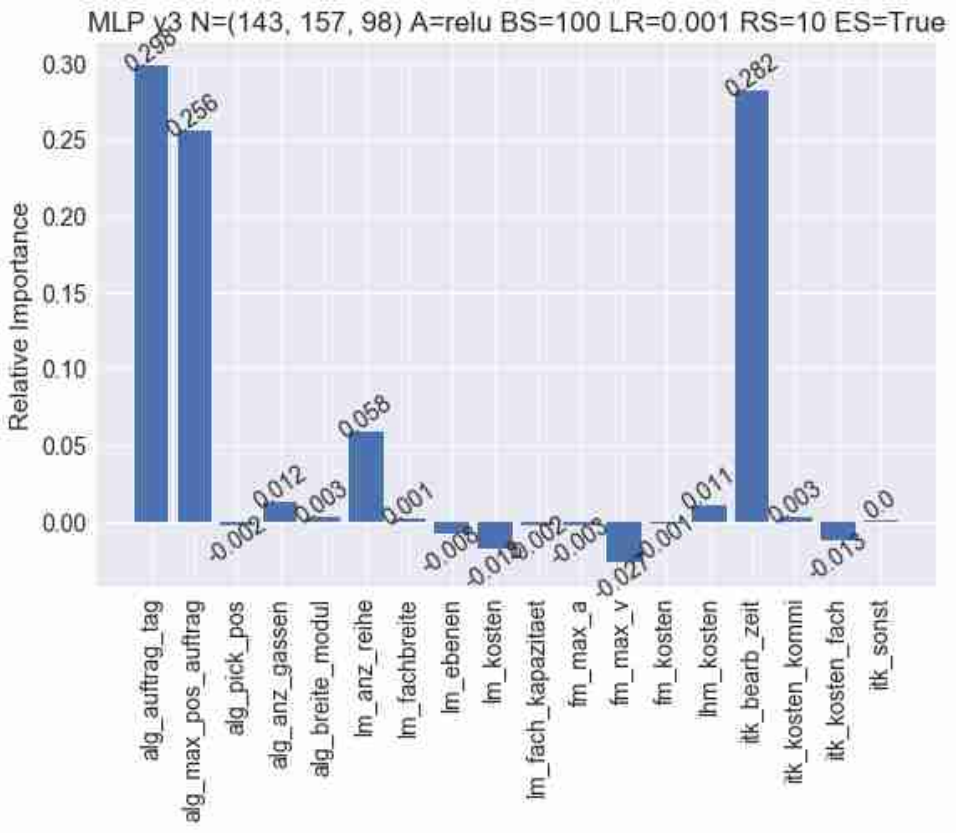


Abbildung 83: Relative Importance für y3_anz_kommi gg.

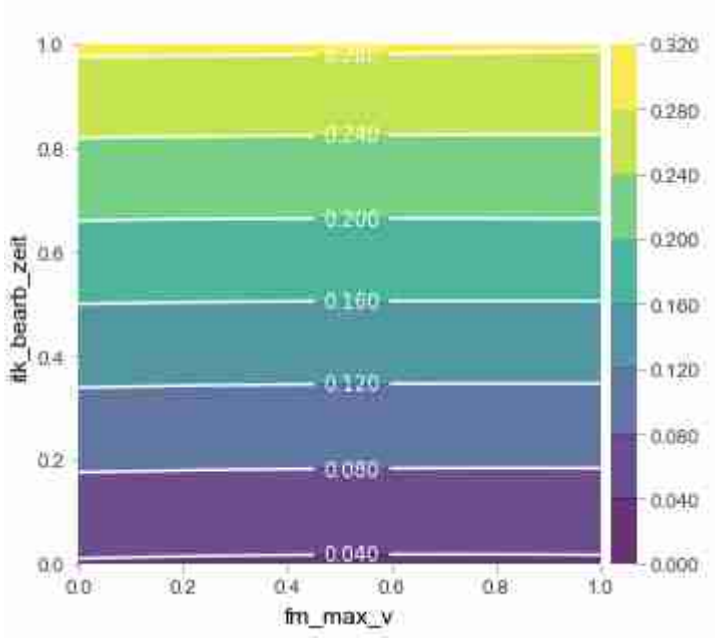


Abbildung 84: 3D-PDP fm_max_v und itk_bearb_zeit für y3_anz_kommi gg.

Es kann hier davon ausgegangen werden, dass die drei wichtigsten Prädiktoren sich gegenseitig verstärken und somit eine hohe Anzahl an Kommissionierern erzeugen.

Abbildung 85 bestätigt dies. Wenn nur einer dieser Prädiktoren einen sehr niedrigen Wert annimmt, kann keine hohe Zielgröße erzeugt werden. Sind wiederum all diese Prädiktoren hoch, wird durch die gegenseitige Verstärkung ein sehr hoher Wert gebildet. Die Wechselwirkungen zwischen allen drei Prädiktoren lassen sich in einer Abbildung leider nicht graphisch darstellen.

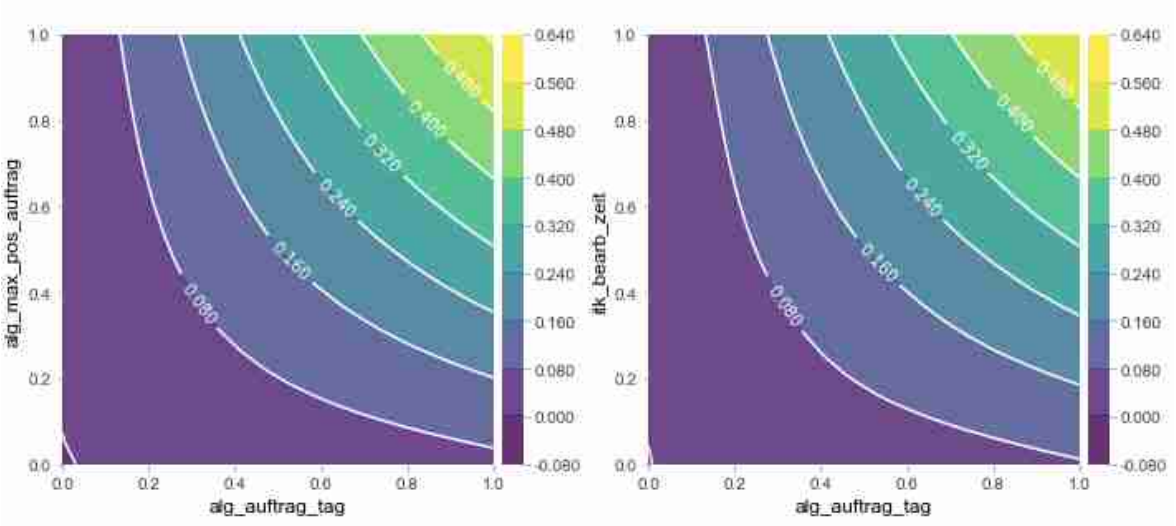


Abbildung 85: 3D-PDP alg_auftrag_tag mit alg_max_pos_auftrag bzw. itk_bearb_zeit für y3_anz_kommi gg.

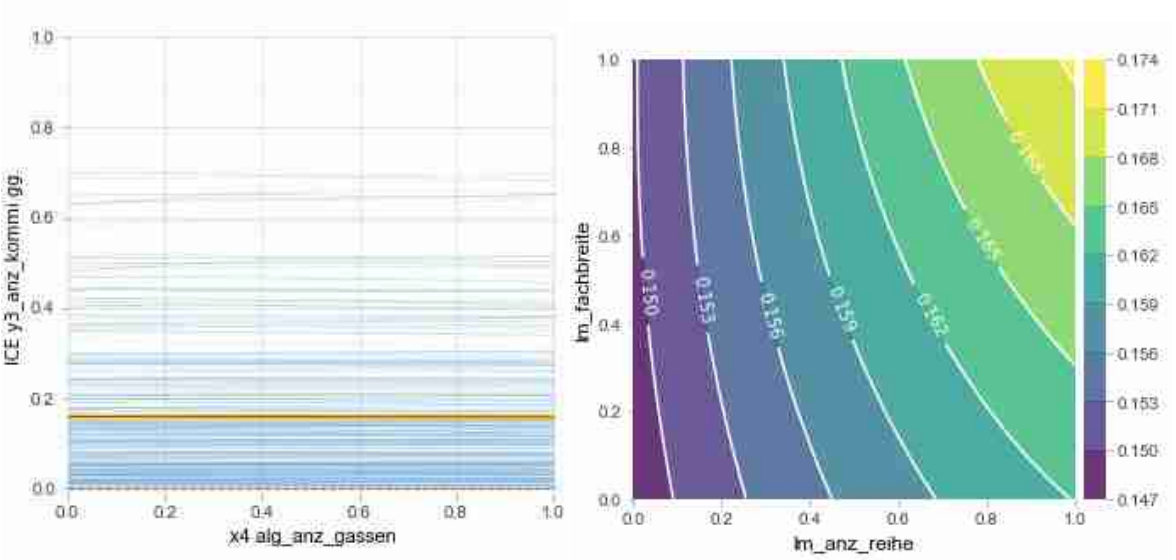


Abbildung 86: ICE für alg_anz_gassen und 3D-PDP Gassenlänge für y3_anz_kommi gg.

Ebenfalls auffällig ist, dass die Lagergröße offensichtlich keinen bzw. nur einen sehr kleinen Einfluss auf die Anzahl der benötigten Kommissionierer hat. Dies wird angesichts der Verläufe in dem ICE-Plot und dem 3D-PDP aus Abbildung 86 klar. Die Anzahl der Gassen hat

hier offensichtlich überhaupt keinen Einfluss und Gassenlänge über die Fachbreite und LM in Reihe nur einen minimal positiven. Offenbar werden hier durchschnittlich lediglich $(0{,}174 - 0{,}147) = 0{,}027 \cong 1{,}5$ Kommissionierer über die Lagergröße definiert.

Alle ICE/c-ICE zur Anzahl der Kommissionierer in der gassengebundenen Betrachtung sind im Datenanhang zu finden. Die PDPs sind im A-16 (S. 222 f.) abgetragen.

TCO (y4_tco)

Auf Grund des hohen Anteils der Betriebskosten an dem TCO kann eine gewisse Ähnlichkeit der RI (Abbildung 87) zu der aus Abbildung 79 festgestellt werden. Allerdings spielen hier die Anteile aus den Investitionskosten zusätzlich hinein. Die Anschaffungskosten für Lagerfächer und LM sind hier deutlich präsenter: itk_kosten_fach von 0,035 auf 0,104 bzw. lhm_kosten von 0,023 auf 0,041.

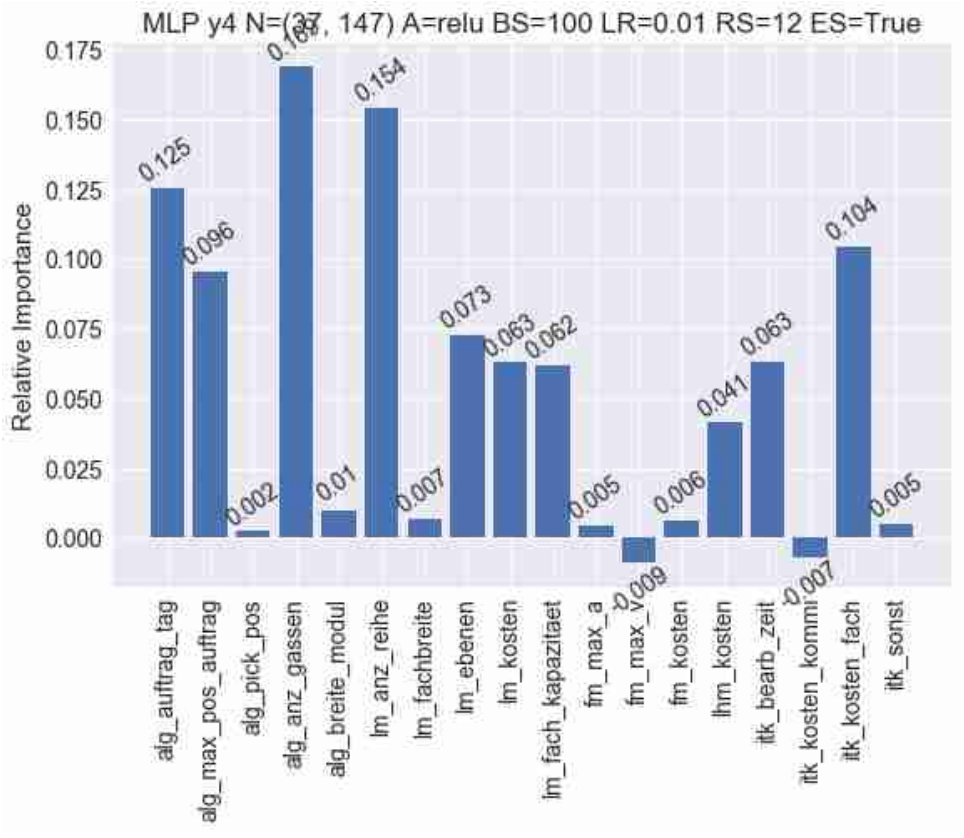


Abbildung 87: Relative Importance für y4_tco gg.

Die Anschaffungskosten für ITK (pro Kommissionierer und sonstige) sowie die FM-Kosten sind hier erstaunlich irrelevant. Es scheint so, als würde der TCO hauptsächlich über die Lagergröße (Anschaffungs-/Betriebskosten für LM, LHM und ITK) und die Personalkosten

definiert. Die Personalkosten spiegeln sich hier in der hohen Wichtigkeit der alg_auftrag_tag sowie der itk_bearb_zeit wieder.

Die c-ICE aus Abbildung 88 beschreiben die Verläufe des TCO für die vier Prädiktoren, welche das ITK-System beschreiben. Zur besseren Vergleichbarkeit ist die TCO-Achse bei allen Vieren gleich skaliert. Wie man ganz klar erkennen kann, spielen die Kosten pro Kommissionierer sowie die sonstigen ITK-Kosten bzgl. des TCO fast überhaupt keine Rolle. Die Kosten pro Fach sowie die Bearbeitungszeit des Systems sind hier die hauptsächlichen Kostentreiber, wobei die Bearbeitungszeit einen etwa doppelt so großen Einfluss hat wie die Kosten pro Fach. Geht man davon aus, dass ein relativ teures ITK-System kurze Bearbeitungszeiten garantieren kann, sollte dieses System in Anbetracht des TCO im Allgemeinen einem günstigen System mit langen Bearbeitungszeiten vorgezogen werden. Die Bearbeitungszeit hebt den TCO durchschnittlich um $0{,}14 \cong 1{,}534 \cdot 10^7$ Euro an.

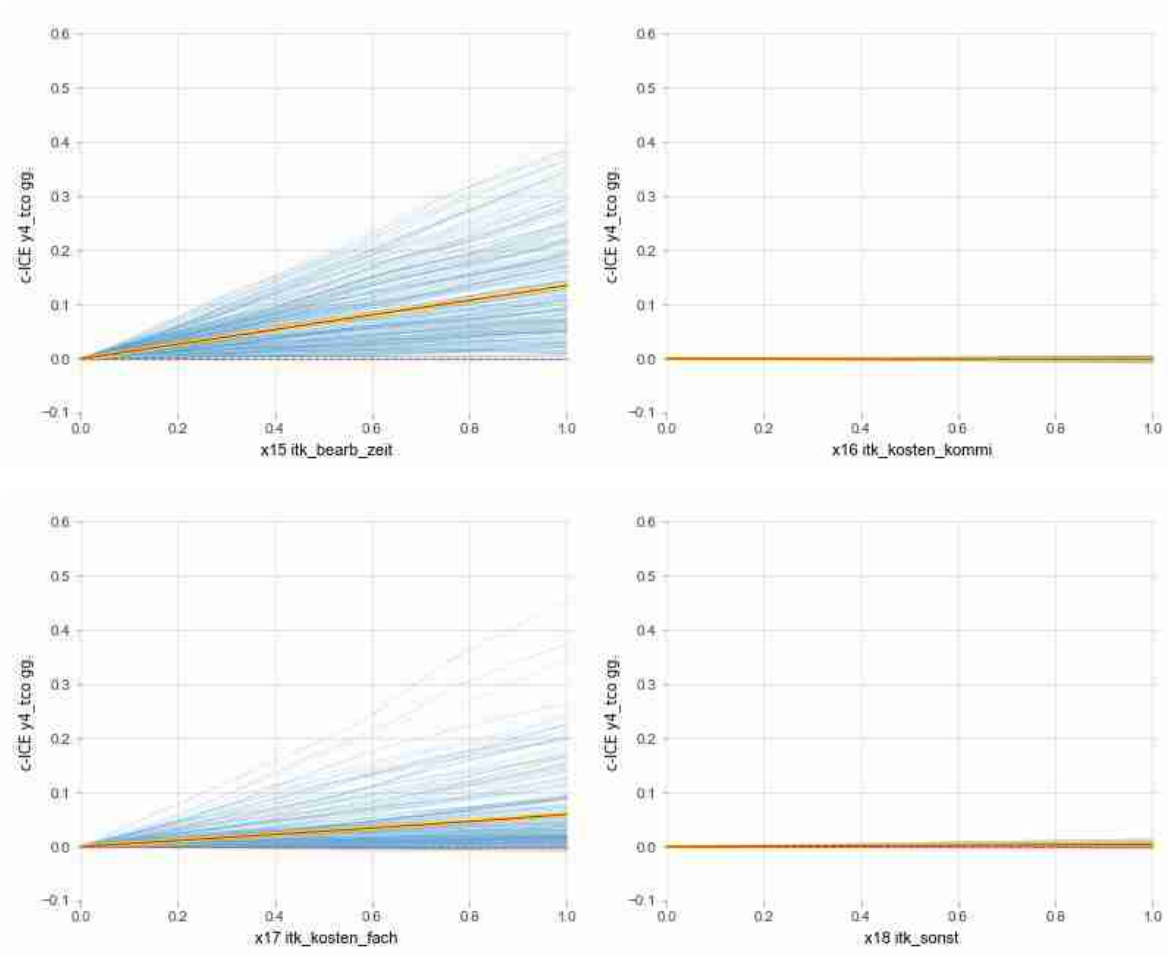


Abbildung 88: c-ICE für ITK gg.

Bei der Betrachtung des FM (Abbildung 89) ist wiederum festzustellen, dass es offensichtlich fast keine Bedeutung in der Berechnung des TCO hat. Es sollte auch hier darauf geachtet werden, dass die Kapazität des FM ausreichend ist. Ansonsten steigt die Anzahl der Aufträge pro Tag proportional zur fehlenden Kapazität an, was einen Anstieg des TCO zur Folge hat. Weiterhin spiegelt sich hier die Funktionsweise der gassengebundenen Betrachtung wieder: Dadurch, dass die Beschleunigung im Vergleich zur Geschwindigkeit einen höheren Ausschlag der Zielgröße erzeugt, kann davon ausgegangen werden, dass hier viele Beschleunigungsvorgänge stattfinden sowie die zurückgelegten Strecken recht kurz sind.

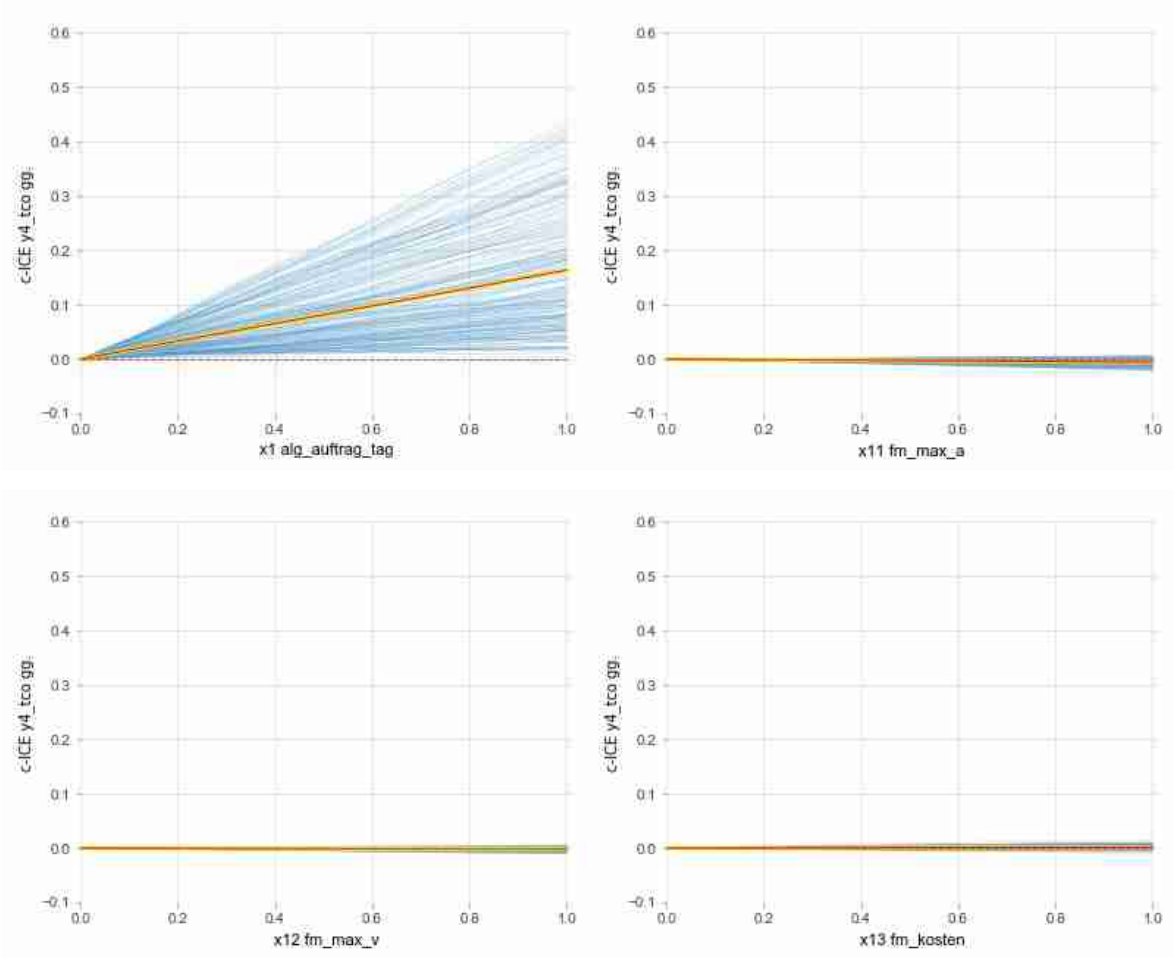


Abbildung 89: c-ICE für FM gg.

Auch hier sind alle c-ICE / ICE im Datenhang zu finden. Die PDPs sind in A-17 (S. 224) gelistet.

3.4 Vergleich der Kommissioniersysteme

Zunächst werden lediglich die Resultate der Versuche betrachtet (Tabelle 16). Die dazugehörigen Effektdiagramme sind in Abbildung 90 zu sehen.

Tabelle 16: Vergleich der Zielgrößen

	y1_invest	y2_betrieb	y3_anz_kommi	y4_tco
Gassenungebundene Betrachtung (gug.)				
Min.	3.000,00	2.790,08	$1{,}424 \cdot 10^{-2}$	30.900,80
Max.	$1{,}487 \cdot 10^{7}$	$19{,}620 \cdot 10^{6}$	175,20	$1{,}996 \cdot 10^{8}$
Durchschnitt	$1{,}599 \cdot 10^{6}$	$3{,}103 \cdot 10^{6}$	22,21	$3{,}262 \cdot 10^{7}$
Gassengebundene Betrachtung (gg.)				
Min.	3.000,00	1.690,08	$0{,}322 \cdot 10^{-2}$	19.900,80
Max.	$1{,}487 \cdot 10^{7}$	$9{,}573 \cdot 10^{6}$	56,19	$1{,}069 \cdot 10^{8}$
Durchschnitt	$1{,}572 \cdot 10^{6}$	$1{,}775 \cdot 10^{6}$	9,06	$1{,}932 \cdot 10^{7}$

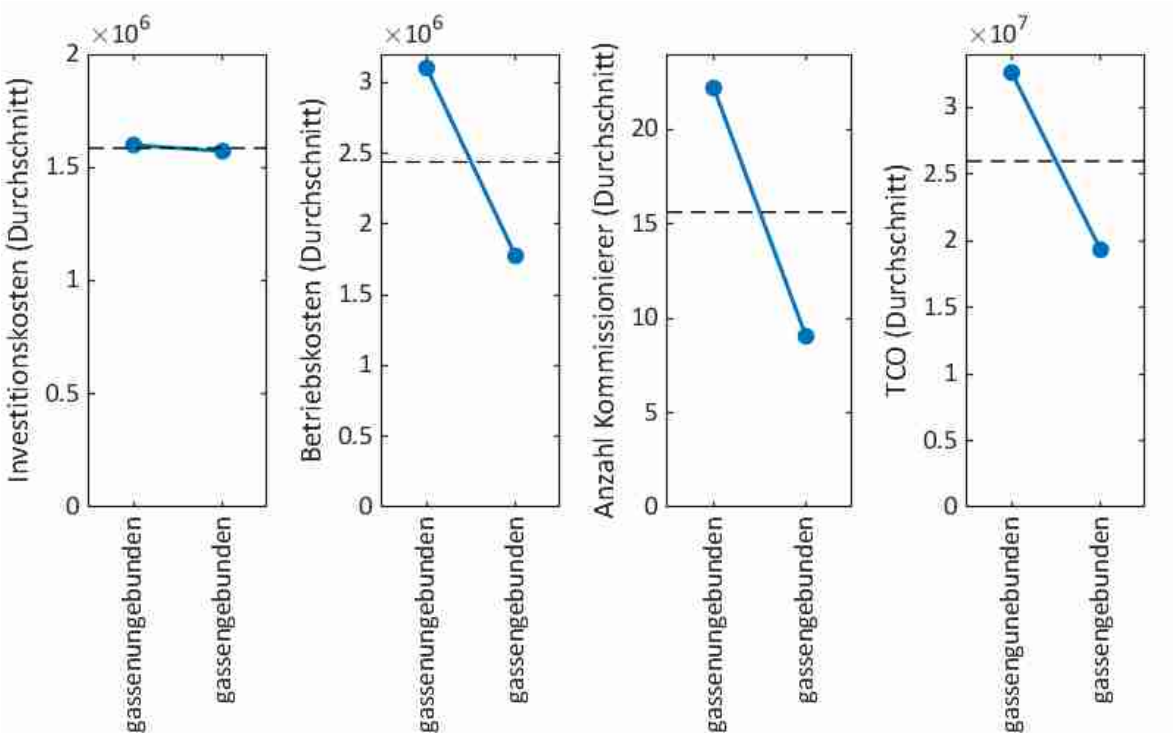


Abbildung 90: Effektdiagramm der Zielgrößen

Wie man erkennen kann, sind die Investitionskosten bei beiden Betrachtungsweisen nahezu identisch. Auffällig ist jedoch, dass die Betriebskosten der gg. Betrachtung

durchschnittlich nur in etwa halb so groß sind wie die der gug. Betrachtung. Dies spiegelt sich auch in der Anzahl der benötigten Kommissionierer wieder, wobei hier der Effekt noch stärker auftritt. Offensichtlich werden in der gg. Betrachtung grundsätzlich wesentlich weniger Kommissionierer benötigt als in der gug. Betrachtung. Dadurch, dass die Investitionskosten praktisch gleich und die Betriebskosten in etwa doppelt so hoch sind, ist auch der gug. TCO fast doppelt so groß wie der gg. TCO.

Investitionskosten (y1_invest)

Bei der Kalkulation der Kosten für LM und LHM kann zwischen gug. und gg. eigentlich kein Unterschied entstehen. Die leichten Differenzen müssen daher indirekt aus der Anzahl der Kommissionierer hervorgehen. Dadurch, dass die gg. durchschnittlich weniger als die Hälfte der Kommissionierer benötigt, müssen dementsprechend auch weniger FM- und ITK-Komponenten angeschafft werden.

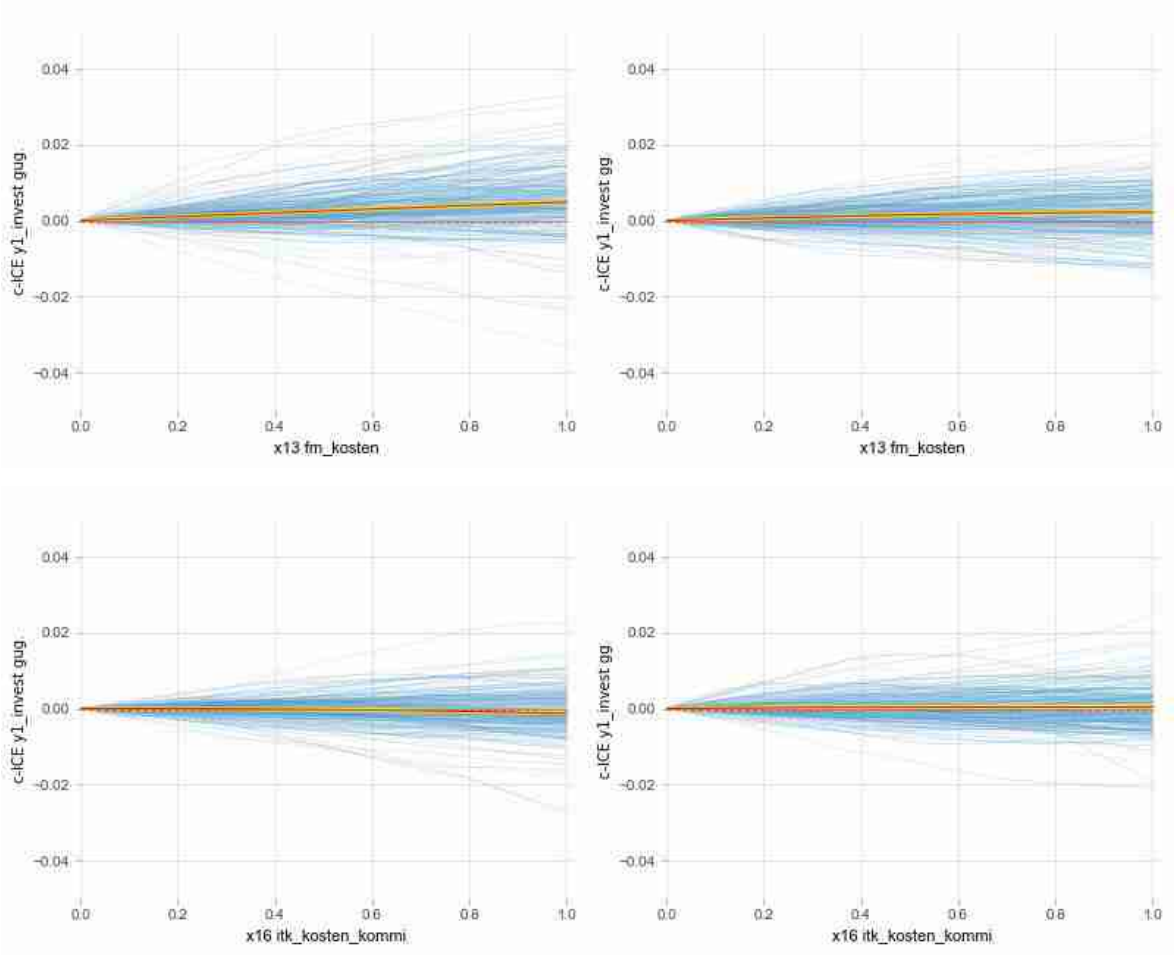


Abbildung 91: c-ICE für fm_kosten, itk_kosten_kommi in gug. und gg.

Abbildung 91 zeigt den c-ICE für die beiden relevanten Prädiktoren in der gug. (links) und gg. (rechts) Betrachtung. Die FM-Kosten scheinen hier der ausschlaggebende Punkt für die leichte Differenz zwischen den Betrachtungsweisen zu sein.

Betriebskosten (y2_betrieb)

Bei den Betriebskosten fällt auf, dass in der gg. Betrachtung die RI-Werte viel gleichmäßiger auf die Prädiktoren verteilt sind als bei der gug. Betrachtung. Hier sind nur 3 Prädiktoren für die Kosten hauptverantwortlich. Zudem wirkt sich eine hohe Verfahrgeschwindigkeit des FM bei der gug. Betrachtung wesentlich günstiger auf die Betriebskosten aus als bei der gg. Betrachtung: Bei maximaler Geschwindigkeit liegt in der gug. Betrachtung die durchschnittliche Ersparnis bei rund $0{,}07 \cong 1.400.000$ Euro, bei der gg. Betrachtung können nur durchschnittlich rund $0{,}002 \cong 19.150$ Euro eingespart werden.

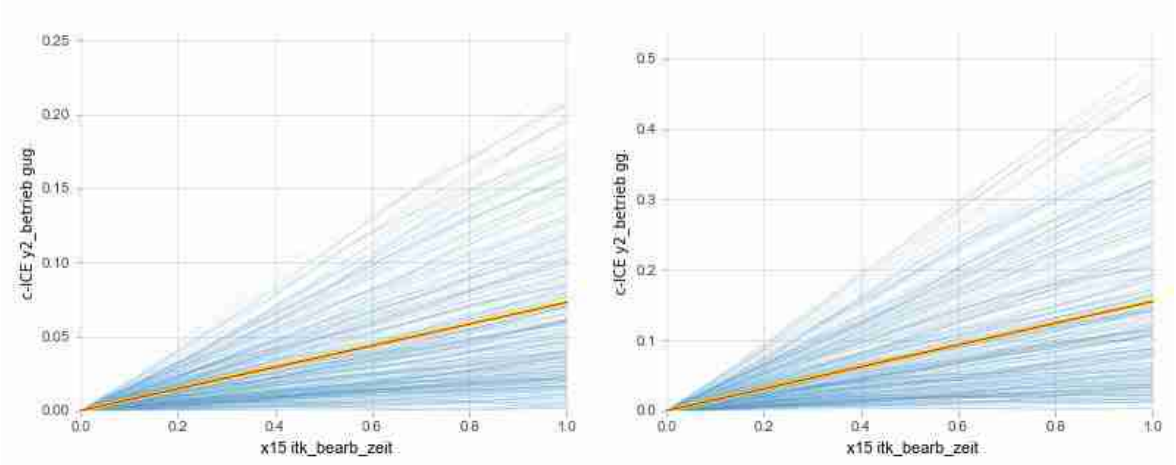


Abbildung 92: c-ICE für itk_bearb_zeit gug und gg.

Die Bearbeitungszeit fällt scheinbar bei der gg. Betrachtung jedoch deutlich mehr ins Gewicht. Eine maximale Bearbeitungszeit erhöht die Betriebskosten in der gug. Betrachtung durchschnittlich um $0{,}075 \cong 1.470.000$ Euro, in der gg. Betrachtung sind es $0{,}16 \cong 1.530.000$ Euro. Daraus lässt sich schließen, dass die Kosten absolut in etwa gleich sind, allerdings ist der Anteil in der gg. Betrachtung auf Grund der wesentlich niedrigeren Gesamtbetriebskosten wesentlich höher.

Für den Fall, dass Betriebskosten eingespart werden müssen, sollten bei beiden Betrachtungsweisen die Bearbeitungszeiten möglichst minimiert werden. Bei der gug. Betrachtung sollte außerdem eine höhere Geschwindigkeit der FM erzielt werden. Dies

spielt bei der gg. Betrachtung offensichtlich überhaupt keine Rolle. Die übrigen Prädiktoren sind wohl für Abschreibungs-, Energiekosten usw. ausschlaggebend, wohingegen die hier betrachteten hauptsächlich die Personalkosten beschreiben.

Anzahl Kommissionierer (y3_anz_kommi)

Die Eindrücke aus den Betriebskosten werden hier bzgl. der Aufteilung in Prädiktoren, welche hauptsächlich Nebenkosten beschreiben und Prädiktoren, welche die Personalkosten ausmachen, bestätigt. Die RI der Nebenkosten-Prädiktoren ist hier in beiden Betrachtungsweisen weitaus unbedeutender, als noch in den Betriebskosten.

Bei der gg. Betrachtung spielen offensichtlich die Wegzeiten in Form der Gassenlänge und Geschwindigkeit nur eine sehr kleine Rolle (siehe Abbildung 83). Die Anzahl der Gassen im System ist hier offenbar weitestgehend irrelevant. Ganz im Gegensatz zur gug. Betrachtung: Hier ist die Anzahl der Gassen nach der Geschwindigkeit des FM und der Anzahl der Aufträge pro Tag der wichtigste Prädiktor (siehe Abbildung 66).

Zur Personaleinsparung bleibt in der gg. Betrachtung im Grunde genommen lediglich die Verwendung eines ITK, welches schnellere Bearbeitungszeiten garantiert. Bei der gug. Betrachtung sollte der Fokus zunächst auf der Geschwindigkeit des FM gelegt werden, wobei die größten Einsparungen bis $0{,}4 \cong 1\frac{m}{s}$ zu erwarten sind. Die Verwendung eines schnellen ITK sollte hier ebenfalls in Betracht gezogen werden.

Interessant ist hier auch die Berücksichtigung der Lagergröße. Bei der gg. Betrachtung macht der Unterschied zwischen dem kleinsten und größten Lager durchschnittlich lediglich 1,5 Kommissionierer aus. Bei der gug. Betrachtung gehen durchschnittlich, nur über die Anzahl der Gassen, 21 Kommissionierer hervor[58].

TCO (y4_tco)

Da sich der TCO zu 10 Teilen aus den Betriebskosten (setzen sich primär aus den Personalkosten zusammen) und nur einem Teil aus den Investitionskosten zusammensetzt

[58] Siehe Abbildung 69 (S. 128): $0{,}2 - 0{,}08 = 0{,}12$ hochskaliert mit Tabelle 7 (S. 111) ergibt $21{,}024$ Kommissionierer

(die bei beiden Betrachtungsweisen nahezu identisch sind), können hier im Endeffekt nur die Erkenntnisse aus den vorigen Zielgrößen wiederholt werden.

Zur Minimierung des TCO sollten in der gug. Betrachtung FM gewählt werden, die eine hinreichende Kapazität gewähren sowie eine möglichst hohe maximale Geschwindigkeit aufweisen können. Die Anschaffungskosten sind hier bzgl. des TCO weitestgehend bedeutungslos (siehe Abbildung 72, S. 132). In der gg. Betrachtung hingegen ist die Beschleunigung des FM wichtiger als die Geschwindigkeit, wobei der Anteil beider am TCO im Grunde genommen zu vernachlässigen ist, genauso wie die Anschaffungskosten (siehe Abbildung 89, S. 150).

Bei der Wahl des ITK sind bei beiden Betrachtungsweisen die Anschaffungskosten und die Kosten pro Kommissionierer für den TCO irrelevant (siehe für gug. Abbildung 73, S. 133 und für gg. Abbildung 88, S. 149). Die Bearbeitungszeit fällt bei der gg. Betrachtung zwar sehr ins Gewicht, beim Vergleich der absoluten Werte in Euro ist jedoch fast kein Unterschied auszumachen:

- Gassenungebundene Betrachtung: $0{,}08 \cong 1{,}596 \cdot 10^7$ Euro
- Gassengebundene Betrachtung: $0{,}14 \cong 1{,}534 \cdot 10^7$ Euro

Dementsprechend sollten beide Betrachtungsweisen ein möglichst schnelles ITK verwenden, wobei die Anschaffungskosten in den hier betrachteten Spannweiten der Prädiktoren weitestgehend uninteressant sind.

3.5 Rechenzeiten und Rundungen

Vergleich der Rechenzeiten

Ein wichtiges Argument für die Verwendung von Metamodellen ist die wesentlich kürzere Rechenzeit im Vergleich zu einer direkten Berechnung in einem Excel-Tool. Um dies zu verifizieren, müssen die Rechenzeiten von Excel-Tool und Metamodell miteinander verglichen werden.

Wie in Abschnitt 3.2.1.4 (S. 103) beschrieben, verfügt das Makro zur Durchführung des Versuchsplans im Excel-Tool über eine Funktion, welche die Rechenzeiten der Versuchsdurchführung aufzeichnet. Da die Rechenzeiten im Excel-Tool mitunter

Schwankungen unterliegen, soll der gesamte Versuch 10 Mal durchgeführt werden, um so ein verlässlicheres Maß für die durchschnittliche Rechenzeit pro Sample zu bekommen. Die Systemkonfiguration des verwendeten Computers kann in A-1 (S. 202) eingesehen werden.

Tabelle 17 zeigt die aufgezeichneten Rechenzeiten des Excel-Tools bei 10 Versuchen. Durchschnittlich benötigt es 821,21 Sekunden zur Berechnung von 30.000 Samples, bzw. $2{,}737 \cdot 10^{-2}$ Sekunden pro Sample.

Tabelle 17: Excel-Tool-Rechenzeiten bei N=30.000 Samples

Versuchslauf	Gesamte Rechenzeit [Sekunden]	Rechenzeit pro Sample [Sekunden/Sample]
1	814,15	$2{,}713 \cdot 10^{-2}$
2	819,45	$2{,}732 \cdot 10^{-2}$
3	820,65	$2{,}736 \cdot 10^{-2}$
4	823,95	$2{,}746 \cdot 10^{-2}$
5	800,57	$2{,}669 \cdot 10^{-2}$
6	826,28	$2{,}754 \cdot 10^{-2}$
7	829,76	$2{,}766 \cdot 10^{-2}$
8	832,37	$2{,}775 \cdot 10^{-2}$
9	819,97	$2{,}733 \cdot 10^{-2}$
10	824,90	$2{,}750 \cdot 10^{-2}$
Durchschnitt	$\mathbf{821{,}21}$	$\mathbf{2{,}737 \cdot 10^{-2}}$

Zur Rechenzeitermittlung der MLPs wurde die Funktion aus Quellcode 9 verwendet. Dabei wird eine zufällige 30.000×18 Matrix im Intervall $[0; 1]$ erzeugt, für die anschließend von dem betrachteten Modell Zielgrößen berechnet wird. Die dafür notwendige Zeit wird aufgezeichnet und präsentiert.

```
503.    def benchmark(number_calc, model):
504.        #Zufällige (number_calc x 18)-Matrix im Intervall [0;1] wird erstellt
505.        a = np.random.random((number_calc, 18))
506.        #Startzeit wird aufgezeichnet
507.        t_start = time.time()
508.        #Modell schätzt Werte für die zufällige Matrix
509.        model.predict(a)
510.        #Endzeit wird aufgezeichnet und mit Startzeit verrechnet
511.        t_end = time.time()
512.        t_time = t_end - t_start
513.        time_sample = t_time/number_calc
514.        #Ergebnis wird präsentiert
515.        print("Zeit:", t_time, "Sekunden bei", len(a), "Berechnungen")
516.        print("Zeit pro Sample:", '{0:.3E}'.format(time_sample))
```

Quellcode 9: Benchmark-Funktion

Auch hier wurden wiederum 10 Versuche durchgeführt. Die Ergebnisse können in Tabelle 18 eingesehen werden. Es wurden hier die in Abschnitt 3.2.2 ausgewählten MLPs für die gassenungebundene Betrachtung verwendet.

Tabelle 18: MLP-Rechenzeiten bei N=30.000 Samples

Versuch	y1_invest		y2_betrieb		y3_anz_kommi		y4_tco	
	Gesamte Rechenzeit [Sekunden]	Rechenzeit pro Sample [Sekunden/ Sample]	Gesamte Rechenzeit [Sekunden]	Rechenzeit pro Sample [Sekunden/ Sample]	Gesamte Rechenzeit [Sekunden]	Rechenzeit pro Sample [Sekunden/ Sample]	Gesamte Rechenzeit [Sekunden]	Rechenzeit pro Sample [Sekunden/ Sample]
1	0,1189	3,963E-06	0,1409	4,697E-06	0,0620	2,065E-06	0,1409	4,697E-06
2	0,1199	3,998E-06	0,1359	4,531E-06	0,0650	2,165E-06	0,1389	4,631E-06
3	0,1189	3,965E-06	0,1329	4,431E-06	0,0650	2,165E-06	0,1379	4,597E-06
4	0,1292	4,305E-06	0,1339	4,464E-06	0,0620	2,065E-06	0,1429	4,764E-06
5	0,1219	4,064E-06	0,1319	4,397E-06	0,0680	2,265E-06	0,1689	5,630E-06
6	0,1249	4,164E-06	0,1369	4,564E-06	0,0630	2,099E-06	0,1519	5,064E-06
7	0,1199	3,998E-06	0,1349	4,498E-06	0,0650	2,165E-06	0,1479	4,931E-06
8	0,1249	4,164E-06	0,1339	4,464E-06	0,0620	2,065E-06	0,1409	4,697E-06
9	0,1199	3,998E-06	0,1339	4,464E-06	0,0650	2,165E-06	0,1399	4,664E-06
10	0,1269	4,231E-06	0,1339	4,464E-06	0,0630	2,099E-06	0,1519	5,064E-06
Durch-schnitt	**0,1225**	**4,085E-06**	**0,1349**	**4,50E-06**	**0,0640**	**2,132E-06**	**0,1462**	**4,874E-06**

Werden die MLPs einzeln mit dem Excel-Tool verglichen, ergeben sich Rechenzeitersparnisse mit dem Faktor 12.830 (y3_anz_kommi) bis Faktor 5.610 (y4_tco). Man kann hier erkennen, dass die Rechenzeit der MLPs mit der Anzahl der Neuronen im Netz korreliert. Möchte man jedoch alle vier Zielgrößen mit Hilfe von MLPs berechnen, müssen die Einzelrechenzeiten aufaddiert werden. Dadurch ergibt sich immerhin noch eine Ersparnis um den Faktor $\frac{821{,}21}{0{,}1225+0{,}1349+0{,}0640+0{,}1462} = 1.756{,}22.$

Rundungen im Excel-Tool

Das zur Berechnung der einzelnen Größen verwendete Excel-Tool benutzt insbesondere bei der Präsentierung der Zielgrößen Rundungen. Diese Rundungen sind für den normalen Gebrauch des Tools auch durchaus nützlich, da man bspw. keine halben LHM einsetzen kann. Allerdings sind diese Rundungen bei der Identifikation von allgemeinen Rechenvorschriften durch ein Metamodell hinderlich. Dies liegt insbesondere daran, dass die gerundete Zielgröße sich trotz veränderlichen Prädiktor-Werten z.T. konstant bleibt. Deshalb wurden diese störenden Rundungen zur Berechnung der hier verwendeten Testfelder entfernt.

Abbildung 93 und Abbildung 94 zeigen die Residual-Plots für die Betriebskosten und die Anzahl der benötigten Kommissionierer bei einem Testfeld, welches aus gerundeten Werten berechnet wurde. Wie man erkennen kann, bilden sich durch diese Rundungen drei Äste im Residual-Plot heraus. Bei dem ersten Ast (vertikal) werden offensichtlich zu hohe Schätzungen abgegeben, der mittlere Ast ist wahrscheinlich mehr oder weniger zufällig nah am echten Ergebnis dran, wobei bei dem dritten Ast (horizontal) zu geringe Werte geschätzt werden. Dementsprechend schlecht ist die Vorhersagegenauigkeit des Modells.

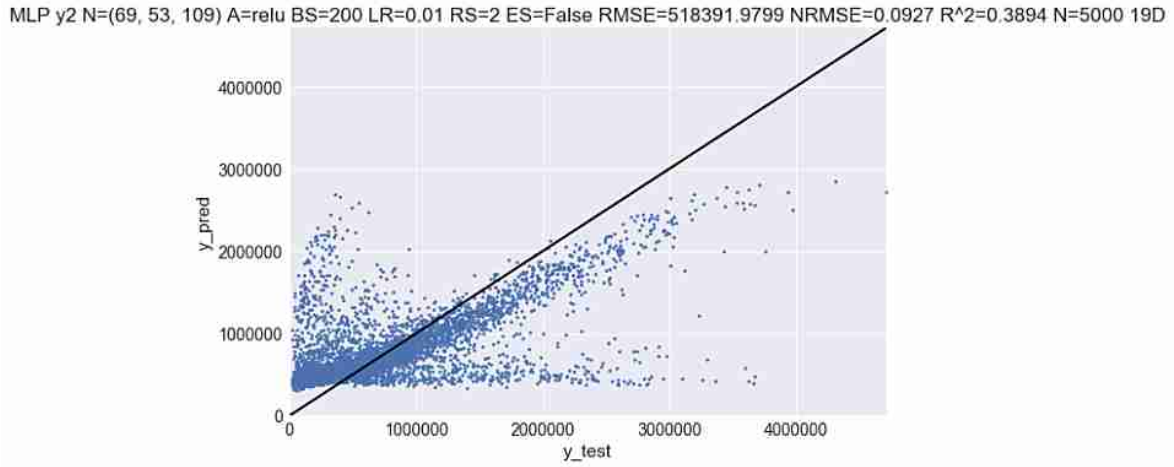


Abbildung 93: Residual-Plot für die Betriebskosten mit Rundungen

Darüber hinaus kann man in Abbildung 94 erkennen, dass dort nur ganzzahlige Werte im realen System vorkommen können. Auch dies kann ein Metamodell nicht identifizieren bzw. schätzen. Weiterhin kann man erkennen, dass das Modell eine gewisse Mindest- und Maximalzielgröße vorauszusetzen scheint, die jedoch in den Testdaten nicht existent ist.

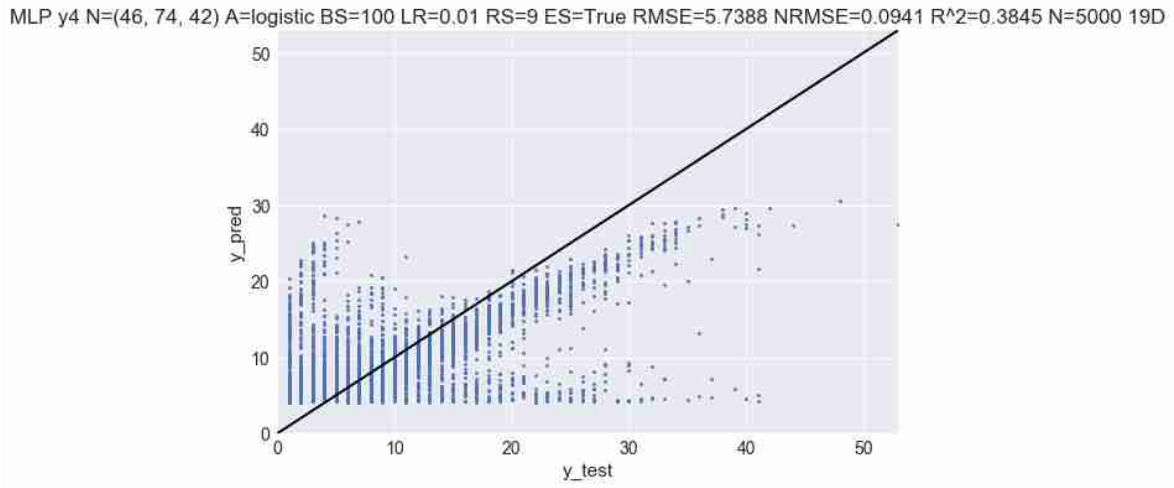


Abbildung 94: Residual-Plot für die Anzahl der Kommissionierer mit Rundungen

4 Optimierung von Kommissioniersystemen

In diesem Kapitel soll ein Genetischer Algorithmus entwickelt werden, welcher beispielhaft ein konventionelles Kommissioniersystem so konfiguriert, dass die TCO minimiert wird. Dazu werden die Modelle und Erkenntnisse aus Abschnitt 3 genutzt. Es wird hier die gassenungebundene Betrachtung gewählt, da sich in Abschnitt 3 gezeigt hat, dass bei dieser Betrachtungsweise mehr Prädiktoren einen Einfluss auf die Zielgröße aufweisen, als bei der gassengebundenen Betrachtung. Daher kann davon ausgegangen werden, dass diese Betrachtungsweise auch dementsprechend schwerer zu optimieren ist.

Der hier verwendete Python-Code ist in voller Länge im Datenanhang unter „04_Optimierung_von_Kommissioniersystemen\MA_Optimierung_GA_FINAL.py“ zu finden.

4.1 Vorgehensweise

Die Entwicklung eines GAs ist relativ komplex, da viele Faktoren ineinandergreifen und sich gegenseitig beeinflussen. Zudem ist keine allgemeine Herangehensweise oder Empfehlung vorhanden, die besagt, welche GA-Komponenten (für ein gegebenes Problem) ideal sind.

Im ersten Schritt soll das Ziel der Optimierung durch einen GA definiert werden. Für welche Größe oder Größen sollen optimale Einstellungen gefunden werden? Sind Nebenbedingungen vorhanden? Welche Faktoren stehen zur Verfügung bzw. sollen optimal eingestellt werden? Soll das Optimierungsproblem einmal gelöst werden oder sehr oft und schnell?

Sind diese Fragen geklärt, können potenziell taugliche GA-Komponenten ausgewählt werden. Hierbei gilt das *No-free-Lunch-Theorem* (Abschnitt 2.5.9, S. 77), d.h. es gibt keine GA-Konfiguration, welche für alle Problemarten optimal ist. Dementsprechend müssen Komponenten ausgewählt werden, welche vielversprechend sind bzw. es müssen mehrere Komponenten miteinander verglichen werden. Anschließend erfolgt die Implementierung der GA-Komponenten in Python.

Stehen mehrere Komponenten zur Auswahl, müssen diese in Testläufen miteinander verglichen werden. Dazu muss zunächst geklärt werden, welche Maße das im ersten Schritt

definierte Ziel der Optimierung am besten widerspiegeln. Weiterhin muss geklärt werden, wie viele Versuche benötigt werden, um eine Aussage über die Leistungsfähigkeit der einzelnen Komponenten für das erklärte Ziel zu erlangen.

Im letzten Schritt wird dann die beste GA-Konfiguration ausgewählt und verwendet.

In Abbildung 95 sind noch einmal die beschriebenen Schritte zur Optimierung durch einen GA zu sehen.

Abbildung 95: Vorgehensweise bei der Optimierung durch einen GA

4.2 Zieldefinition

Das Ziel der Optimierung soll eine Minimierung der TCO sein, da diese Größe sowohl die Investitionskosten als auch die Betriebskosten (hier) über 10 Jahre beinhaltet.

Als Nebenbedingung wird die Anzahl der Stellplätze S des Kommissioniersystems aufgeführt. Das Lager soll eine definierbare Mindestanzahl von Lagerplätzen bieten können, damit ein Betrieb möglich ist. Würde hier keine Mindestgröße angegeben, würde das optimale Kommissioniersystem schlichtweg nicht existieren, denn bei einem nicht-vorhandenen System ist der $\mathrm{TCO} = 0$ und somit minimal. Weiterhin ist eine Extrapolation, wie auch in Abschnitt 3, nicht zulässig. Dies muss daher in Form von zusätzlichen Nebenbedingungen ebenfalls verhindert werden.

Ausgangspunkt sind hier die durch das Metamodell betrachteten Faktoren/Prädiktoren. Man könnte diese 18 Faktoren direkt optimieren, jedoch ist davon auszugehen, dass die daraus resultierende optimale Systemkonfiguration nicht auf dem Markt zu erhalten ist bzw. auch technisch nicht umsetzbar ist. Dementsprechend ist es äußerst fraglich, was diese Optimierung für einen Nutzen hätte. Sinnvoller wäre es, hier real existierende Systemkomponenten (verschiedene LM, FM usw.) für eine Optimierung in Betracht zu ziehen und eine optimale Zusammenstellung aus ihnen zu generieren. Dies hätte den Vorteil, dass genau definiert ist, welche Kombination aus unterschiedlichen LM, FM, LHM, ITK zusammen mit der Lagergröße, unter Erfüllung der Nebenbedingungen, optimal ist.

Da Kommissioniersysteme i.d.R. nicht im Sekundentakt geplant werden, sondern sich die Planungszeit über Monate und Jahre hinwegzieht, ist hier der Faktor Zeit eher zweitrangig (one-off Problem). Dementsprechend soll die Optimierung ggf. etwas länger dauern und dafür möglichst das globale Optimum finden, als eine sehr schnelle Rechenzeit bei niedrigerer Genauigkeit aufweisen.

4.3 Komponenten des Genetischen Algorithmus

Im Folgenden werden einige Komponenten aus dem Evolutionary-Computation Framework[59] *DEAP* (Version 1.2.2) in Python verwendet (Quellcode 10). Dieses Framework bietet einige Grundfunktionen eines GAs an, die recht einfach im Code implementiert werden können[60].

```
16. from deap import base
17. from deap import creator
18. from deap import tools
```

Quellcode 10: Importierung von DEAP

[59] Zu Deutsch: „Programmiergerüst“

[60] Eine detaillierte Dokumentation zu DEAP kann in Fortin et al. 2012 oder im Internet unter https://deap.readthedocs.io/en/master/ gefunden werden.

4.3.1 Kodierung

Die erste Frage, die sich hier stellt, ist: Wie sollte der GA kodiert werden?

In Abschnitt 3 wurden insgesamt 18 numerische Faktoren identifiziert, mit denen ein konventionelles Kommissioniersystem bzgl. des TCO beschrieben werden kann[61]. Da schon ein Metamodell (MLP) für diese Zielgröße vorhanden ist, bietet sich dessen Verwendung als Fitness-Funktion hier an. Dementsprechend sollten bzw. müssen mindestens diese 18 Prädiktoren aus den Phänotypen resultieren. Abbildung 96 zeigt die Ebene der Kodierung und benötigten Vorgänge, um von der einen Ebene auf die andere zu gelangen. Die Breite der Kästen soll die Anzahl der Variablen pro Ebene für diesen Fall andeuten.

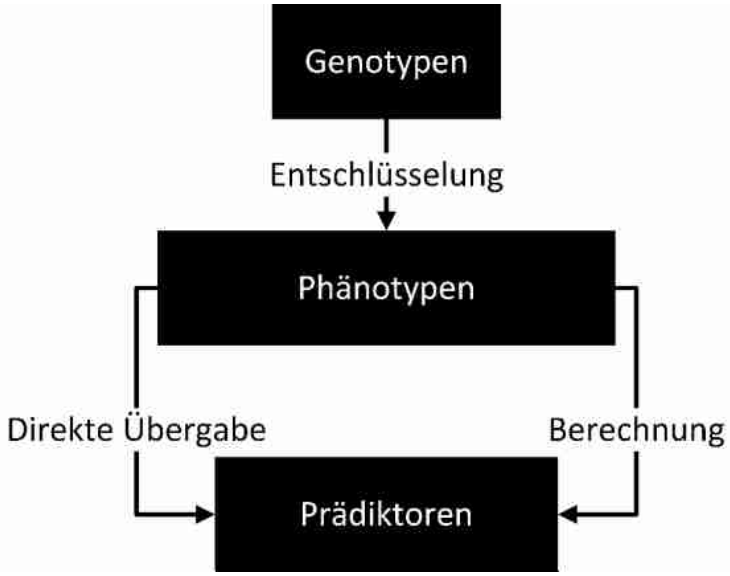


Abbildung 96: Ebenen der Kodierung

Die meisten Phänotypen sind hier identisch zu den Prädiktoren und können direkt übergeben werden, allerdings besteht hier auch eine Ausnahme: In Abschnitt 3.2.1.2 (S. 84) wurden die „Breite der Gasse“ und die „LM-Tiefe“ durch den Prädiktor *alg_breite_modul* substituiert, da sich diese aus der Summe der beiden Faktoren ergibt. Diese Substitution muss durch eine Berechnung rückgängig gemacht werden. Es werden also die Breite der Gasse (hier repräsentiert durch den Phänotypen: *fm_breite*) und die LM-Tiefe (*lm_fach_tiefe*) miteinander verrechnet und dann in Form des Prädiktors

[61] Dies gilt natürlich nur für ein System mit den dort festgelegten kategorischen Faktoren.

alg_breite_modul, zusammen mit den übrigen Prädiktoren, zur Schätzung des TCO an das Metamodell übergeben.

Es werden hier also insgesamt 19 Phänotypen und 18 Prädiktoren betrachtet. Von den Phänotypen sollen jedoch drei durch den Nutzer einstellbar sein. Es handelt sich hierbei um *alg_auftrag_tag*, *alg_pos_auftrag* und *alg_pick_pos*. Durch diese drei wird die Leistungsanforderung an das Kommissioniersystem vorgegeben. Daher werden sie nicht in die Optimierung eingeschlossen, sondern fließen als Konstanten, über die Optimierung hinweg, in die Schätzung des Metamodells mit ein.

Fraglich ist jedoch noch, inwiefern die übrigen 16 Phänotypen in Genotypen verschlüsselt werden können. Das Ziel der Optimierung besteht darin, eine Kombination von Komponenten und der Lagergröße zu finden, welche den TCO minieren. Diese beiden müssen daher in dem Genotypen enthalten sein.

Die Größe des Lagers wird durch die Anzahl der Gassen sowie der Anzahl der LM nebeneinander bestimmt. Jede Komponente des Kommissioniersystems wird über 3 bis 5 Phänotypen/Prädiktoren beschrieben. Diese treten bei realen Komponenten jedoch immer nur zusammen auf. Dementsprechend können sie nicht einzeln in den Genotypen einfließen, sondern müssen gruppiert werden. Tabelle 19 soll dies verdeutlichen. Die vier Ausprägungen können bei realen Systemen nur gemeinsam auftreten. Das ITK 1 ist durch die vier Einträge in Spalte 2 definiert, wohingegen das ITK 2 durch die in Spalte 3 definiert ist. Daher müsste die Nummerierung der einzelnen Komponenten des Kommissionier-systems als Genotyp betrachtet werden.

Tabelle 19: Gruppierung von Phänotypen

ITK Nummer	1	2
itk_bearb_zeit	40	10
itk_kosten_kommi	10	200
itk_kosten_fach	0	20
itk_sonst	3.000	50.000

Zur Auswahl steht hier die binäre Kodierung oder ganzzahlig als Integer (Abschnitt 2.5.1, S. 65). Eine Kodierung als Fließkommazahl würde hier keinen Sinn ergeben, da die Anzahl der Gängesowie die Anzahl der LM in Reihe nur ganzzahlige Werte annehmen können. Das Gleiche gilt für die Nummerierung der Systemkomponenten.

Die Wahl fällt hier auf eine ganzzahlige Kodierung. Dies hat den Vorteil, dass der Lösungsraum für den Fall, dass neue Komponenten zur Auswahl stehen, leicht erweitert werden kann. Zudem ist die Implementierung im Code wesentlich leichter und intuitiver als bei einer binären Kodierung. Der Genotyp besteht demnach aus 6 ganzzahligen Werten: $z_1, z_2, \dots, z_6$ (siehe Tabelle 20).

Tabelle 20: Genotyp des Genetischen Algorithmus

Gen	Beschreibung	Typ
z1_Anz_Gassen	Anzahl Gassen im Lager	numerisch (ganzzahlig)
z2_Anz_LM	Anzahl LM nebeneinander	numerisch (ganzzahlig)
z3_LM	Gewähltes LM	kategorisch
z4_FM	Gewähltes FM	kategorisch
z5_LHM	Gewähltes LHM	kategorisch
z6_ITK	Gewähltes ITK	kategorisch

4.3.2 Initialisierung

Die Initialisierung der Anfangspopulation ist in Quellcode 11 dargestellt und ausreichend kommentiert. Es gilt jedoch zu erwähnen, dass DEAP in der Lage ist, mehrere Fitness-Werte zu berücksichtigen. In Zeile 529 sind die Gewichtungen zwischen den beiden Fitness-Werten definiert. Das negative Vorzeichen gibt an, dass beide Werte minimiert werden sollen. Bei der ersten Fitness, welche mit 99% gewichtet ist, handelt es sich um den TCO. In der zweiten Fitness wird der Penalty-Wert für ungültige Lösungen festgehalten. Darauf wird im nächsten Abschnitt näher eingegangen.

Die in Zeile 103-106 importierten Phänotypen der Komponenten wurden zufällig erzeugt. Es handelt sich hierbei um 50 zufällige LM, 40 FM, 5 LHM sowie 10 ITK. Die Spannweiten der Phänotypen pro Komponente können im A-18 (S. 226) eingesehen werden. Der Code

zur zufälligen Erstellung dieser Komponenten ist im Datenanhang[62] hinterlegt. Der erste Eintrag der importierten Phänotypen entspricht hierbei dem Genotyp 1, z.B. der 1. Eintrag aus „LM_Konfig.csv“ entspricht $z3\_LM = 1$. So wird eine etwaige Sortierung der importierten Komponenten auf die Gene übertragen.

```
100.    def run_ga(model, n_pop, tournsize, cxpb, indpb, mutpb, mutpb_1, mutpb_2,
101.               conv_limit, restarts, show, slots_plan, auftraege_tag):
102.        #Reale Systemkomponenten importieren
103.        lm_konfig = import_data('LM_Konfig.csv')
104.        fm_konfig = import_data('FM_Konfig.csv')
105.        lhm_konfig = import_data('LHM_Konfig.csv')
106.        itk_konfig = import_data('ITK_Konfig.csv')
107.
528.        #Definition Optimierungskriterium: Fitness (TCO) soll minimiert werden
529.        creator.create("FitnessMin", base.Fitness, weights=(-0.99, -0.01))
530.
531.        #Individuen-Klasse wird erzeugt
532.        creator.create("Individual", list, fitness=creator.FitnessMin)
533.        toolbox = base.Toolbox()
534.
535.        #Wertebereiche für Gene z
536.        #Anzahl Gassen
537.        z1_min = 1
538.        z1_max = 40
539.        #Anzahl LM nebeneinander
540.        z2_min = 1
541.        z2_max = 30
542.        #LM
543.        z3_min = 1
544.        z3_max = len(lm_konfig)
545.        #FM
546.        z4_min = 1
547.        z4_max = len(fm_konfig)
548.        #LHM
549.        z5_min = 1
550.        z5_max = len(lhm_konfig)
551.        #ITK
552.        z6_min = 1
553.        z6_max = len(itk_konfig)
554.
555.        #Untere und obere Schranke für Mutation
556.        lower = [z1_min, z2_min, z3_min, z4_min, z5_min, z6_min]
557.        upper = [z1_max, z2_max, z3_max, z4_max, z5_max, z6_max]
```

Quellcode 11: Initialisierung der Anfangspopulation

[62] Siehe: „04_Optimierung_von_Kommissioniersystemen\MA_Optimierung_Samples_Generator_01.py“

```
559.        #Genotypen werden benannt, Art der Erzeugung in der Startpopulation,
560.        #Art der Erzeugung in der Startpopulation (zufällige Integer),
561.        #Obere und untere Schranke des Wertebereiches
562.        toolbox.register("z1_Anz_Gassen", random.randint, z1_min, z1_max)
563.        toolbox.register("z2_Anz_LM", random.randint, z2_min, z2_max)
564.        toolbox.register("z3_LM", random.randint, z3_min, z3_max)
565.        toolbox.register("z4_FM", random.randint, z4_min, z4_max)
566.        toolbox.register("z5_LHM", random.randint, z5_min, z5_max)
567.        toolbox.register("z6_ITK", random.randint, z6_min, z6_max)
568.
569.        #Genotypen werden der individual-Klasse zugeordnet
570.        toolbox.register("individual", tools.initCycle, creator.Individual,
571.                         (toolbox.z1_Anz_Gassen,
572.                          toolbox.z2_Anz_LM,
573.                          toolbox.z3_LM,
574.                          toolbox.z4_FM,
575.                          toolbox.z5_LHM,
576.                          toolbox.z6_ITK
577.                          ))
578.
579.        #zufällige Population erzeugen
580.        toolbox.register("population", tools.initRepeat, list, toolbox.individual)
604.        pop = toolbox.population(n=n_pop)
```

Quellcode 12: Initialisierung der Anfangspopulation (Fortsetzung)

Die Anzahl der Individuen (n_pop) kann durch den Nutzer beliebig eingestellt werden. Die Individuen der Population beinhalten ihren Genotypen sowie die beiden Fitness-Werte. Eines dieser Individuen kann bei den hier betrachteten 6 Genen z.B. so aussehen: $\texttt{individual} = [28, 27, 19, 15, 2, 4]$. Dieses Kommissioniersystem verfügt in diesem Fall über 28 Gassen, 27 LM in Reihe, LM #19, FM #15, LHM #2 und ITK #4.

DEAP bietet zudem einige nützliche Werkzeuge an, mit denen die Performance des GA überwacht werden kann (Quellcode 13). Dazu zählt zum einen die sogenannte „Hall of Fame“ (hof). Diese bildet über alle Individuen, die jemals im GA betrachtet wurden, eine Bestenliste. Die Individuen in dieser Liste sind einzigartig, d.h. ein und dieselbe Konfiguration kann auch nur ein einziges Mal in der Hall of Fame auftreten. Die Anzahl der Individuen in dieser Bestenliste ist über die Variable `maxsize` frei einstellbar.

```
586.    #Hall of Fame wird angelegt: maxsize gibt an wie viele Einträge enthalten sind
587.    hof = tools.HallOfFame(maxsize=10)
588.
589.    #Logbook für Minimum und Durschnitts-Fitness pro Generation erstellen
590.    stats = tools.Statistics(key=lambda ind: ind.fitness.values)
591.    stats.register("min", np.min, axis=0)
592.    stats.register("mean", np.mean, axis=0)
593.    logbook = tools.Logbook()
```

Quellcode 13: Statistik-Tools in DEAP

Weiterhin kann ein sogenanntes „Logbook“ erstellt werden. Es zeichnet statistische Merkmale über die Generationen hinweg auf. In diesem Fall wird pro Generation die Fitness des besten Individuums sowie die durchschnittliche Fitness über alle zulässigen Individuen der Population festgehalten. Die Anzahl der zulässigen Individuen in der aktuellen Population wird ebenfalls aufgezeichnet. Diese Werte können anschließend zur Analyse in Graphen dargestellt werden.

4.3.3 Fitness-Funktion

Das Ziel der Optimierung liegt darin, eine Systemkonfiguration zu finden, welche unter Einhaltung der Nebenbedingungen den TCO über 10 Jahre minimiert. Zur Berechnung des TCO wird das MLP aus Abschnitt 3.2.2 (S. 119) verwendet. Man könnte auch hier theoretisch die Fitness mit Hilfe des Excel-Tools berechnen, allerdings wären in dem Fall die benötigten Rechenzeiten im Vergleich zu einem MLP nicht mehr handhabbar (siehe Abschnitt 3.5, S. 155). Bei den Nebenbedingungen handelt es sich um eine vom Nutzer definierte Anzahl von Stellplätzen, welche das System mindestens aufweisen soll sowie einer Anzahl von Aufträgen, die durch das System pro Tag zu bewältigen sind. Weiterhin muss sich die Berechnung des Metamodells in dem Lösungsraum bewegen, für den es trainiert wurde, denn auch hier ist keine Extrapolation zulässig.

Die Hauptfunktion zur Evaluation eines Individuums ist in Quellcode 14 zu sehen. Im ersten Schritt muss der Genotyp des Individuums in die einzelnen Phänotypen entschlüsselt werden (Zeile 299). Dazu wird das Dictionary i verwendet. Es beinhaltet sämtliche Informationen zu dem aktuellen Kommissioniersystem. Quellcode 15 zeigt das leere Phänotypen-Dictionary aus der Funktion `get_i`, in welcher der Phänotyp aus den einzelnen Komponenten zusammengesetzt wird.

```
296.    def evaluate_individual(individual, lm_konfig, fm_konfig, lhm_konfig,
297.                            itk_konfig, slots_plan):
298.        #Dictionary wird aus Merkmalen des Individuums zusammengestellt
299.        i = get_i(individual, lm_konfig, fm_konfig, lhm_konfig, itk_konfig)
300.        #Anzahl Stellplätze berechnen
301.        slots_real = calc_slots(i)
302.        #Einschränkungen des Gültigkeitsbereichs des Metamodells
303.        max_auftrag_tag = 8000
304.        max_breite_modul = 6
305.        #Prüfen ob mindestens eine der Nebenbedingungen verletzt wurde
306.        if slots_real < slots_plan or (
307.                i['alg_auftrag_tag'] > max_auftrag_tag) or (
308.                        i['alg_breite_modul'] > max_breite_modul) or (
309.                                i['fm_kapazitaet'] < 1) :
310.
311.            #Normalisierung der verletzten Nebenbedingung nach Carlson 1995
312.            def normalize_constraint(g, h):
313.                d = (h-g) / (g+h)*2
314.                return d
315.
316.            #Penalty-Werte für erletzte Nebenbedingungen berechnen
317.            #Kapazität des FM
318.            if i['fm_kapazitaet'] < 1:
319.                penalty_kapazitaet = normalize_constraint(i['fm_kapazitaet'], 1)
320.            else:
321.                penalty_kapazitaet = 0
322.            #Anzahl der Stellplätze
323.            if slots_real < slots_plan:
324.                penalty_slots = normalize_constraint(slots_real, slots_plan)
325.            else:
326.                penalty_slots = 0
327.            #Breite des Moduls
328.            if i['alg_breite_modul'] > max_breite_modul:
329.                penalty_breite_modul = normalize_constraint(
330.                        max_breite_modul, i['alg_breite_modul'])
331.            else:
332.                penalty_breite_modul = 0
333.            #Aufträge pro Tag
334.            if i['alg_auftrag_tag'] > max_auftrag_tag:
335.                penalty_auftrag_tag = normalize_constraint(
336.                        max_auftrag_tag, i['alg_auftrag_tag'])
337.            else:
338.                penalty_auftrag_tag = 0
339.
340.            #Summe der normalisierten Penalties bilden für
341.            #Gesamt-Penalty-Wert
342.            penalty = (penalty_kapazitaet + penalty_slots +
343.                       penalty_breite_modul + penalty_auftrag_tag)
344.
345.            #Fitness (TCO) wird auf einen sehr hohen Wert gesetzt
346.            tco = 399999999
347.            #Fitness und Penalty-Wert zurückgeben
348.            return (tco, penalty)
349.        #Wenn keine Nebenbedingung verletzt wurde
350.        #kann die Fitness durch das MLP berechnet werden
351.        else:
352.            #Fitness ausrechnen (TCO über 10 Jahre)
353.            tco = calc_tco(i, model)
354.            #Fitness und Penalty-Wert zurückgeben
355.            return (tco, 0)
```

Quellcode 14: Bewertung eines Individuums

```
116.    def get_i(individual, lm_konfig, fm_konfig, lhm_konfig, itk_konfig):
117.        i = {'alg_auftrag_tag':'',      #x1
118.             'alg_pos_auftrag':'',
119.             'alg_max_pos_auftrag':'',  #x2
120.             'alg_pick_pos':'',         #x3
121.             'alg_anz_gassen':'',       #x4
122.             'alg_breite_modul':'',     #x5
123.             'lm_anz_reihe':'',         #x6
124.             'lm_fachbreite':'',        #x7
125.             'lm_fachtiefe':'',
126.             'lm_ebenen':'',            #x8
127.             'lm_kosten':'',            #x9
128.             'lm_fach_kapazitaet':'',   #x10
129.             'fm_breite':'',
130.             'fm_max_a':'',             #x11
131.             'fm_max_v':'',             #x12
132.             'fm_kapazitaet':'',
133.             'fm_kosten':'',            #x13
134.             'lhm_kosten':'',           #x14
135.             'itk_bearb_zeit':'',       #x15
136.             'itk_kosten_kommi':'',     #x16
137.             'itk_kosten_fach':'',      #x17
138.             'itk_sonst':''             #x18
139.             }
```

Quellcode 15: Leeres Phänotyp-Dictionary

Dieses Dictionary `i` muss nun mit Hilfe des Genotypen `individual` aus den entsprechenden Phänotypen der dort eigetragenen Komponenten zusammengesetzt werden. Quellcode 16 zeigt diesen Vorgang für das FM und LM. Die Komponenten mit den jeweiligen Phänotypen wurden bei der Initialisierung des GAs importiert (hier: „fm_konfig“ und „lm_konfig“).

```
157.    #z4_FM
158.    i['fm_breite'] = (fm_konfig['fm_breite']).loc[individual[3]-1]
159.    i['fm_max_a'] = (fm_konfig['fm_max_a']).loc[individual[3]-1]
160.    i['fm_max_v'] = (fm_konfig['fm_max_v']).loc[individual[3]-1]
161.    i['fm_kosten'] = (fm_konfig['fm_kosten']).loc[individual[3]-1]
162.
163.    kapa = 'fm_kapazitaet_' + str(individual[4])
164.    i['fm_kapazitaet']= (fm_konfig[kapa]).loc[individual[3]-1]
165.
166.    #z3_LM
167.    #Maße bei lm_fachbreit sind immer die Außenmaße des Regals
168.    i['lm_fachbreite'] = (lm_konfig['lm_fachbreite']).loc[individual[2]-1]
169.    i['lm_fachtiefe'] = (lm_konfig['lm_fachtiefe']).loc[individual[2]-1]
170.    i['lm_ebenen'] = (lm_konfig['lm_ebenen']).loc[individual[2]-1]
171.    i['lm_kosten'] = (lm_konfig['lm_kosten']).loc[individual[2]-1]
172.    #Die Kapazität des Faches wird aus den LHM-Maßen berechnet
173.    x = math.floor(i['lm_fachbreite'] / i['lhm_breite'])
174.    y = math.floor(i['lm_fachtiefe'] / i['lhm_tiefe'])
175.    #Maximale Kapazität darf nicht überschritten werden
176.    i['lm_fach_kapazitaet'] = min((x*y), 10)
```

Quellcode 16: Zusammensetzen des Phänotypen – FM- und LM-Anteil

Hier werden außerdem die noch fehlenden Phänotypen berechnet. Quellcode 17, Zeile 203 zeigt die Berechnung der Modulbreite. Dieser Berechnungsschritt wurde in der Versuchsplanung (Abschnitt 3.2.1.2, S. 84) außen vor gelassen und muss daher hier bei der Verwendung des MLPs vorgeschaltet werden.

```
187.    #Wenn die Kapazität des FM kleiner ist als die üblichen Positionen
188.    #pro Auftrag, wird die Anzahl der Aufträge pro Tag um den Faktor
189.    #zwischen Positionen pro Auftrag und Kapazität angehoben.
190.    if i['fm_kapazitaet'] < i['alg_pos_auftrag']:
191.        i['alg_max_pos_auftrag'] = i['fm_kapazitaet']
192.        #Aufträge pro Tag werden um den um den Faktor der
193.        #fehlenden Kapazität des FM angehoben
194.        if i['fm_kapazitaet'] >= 1:
195.            i['alg_auftrag_tag'] = i['alg_auftrag_tag'] * (
196.                    i['alg_pos_auftrag'] / i['fm_kapazitaet']
197.                    )
198.    else:
199.        i['alg_max_pos_auftrag'] = i['alg_pos_auftrag']
200.
201.    #Die Breite eines Moduls berechnet sich aus der Fachtiefe
202.    #und der FM-Breite
203.    breite_modul = (2*i['lm_fachtiefe'] + i['fm_breite']) / 1000
204.    #Damit die Rechnung in den Grenzen des Metamodells bleibt
205.    i['alg_breite_modul'] = max(1.6, breite_modul)
206.    #Fertiges Dictionary mit Parametern wird zurückgegeben
207.    return i
```

Quellcode 17: Zusammensetzen des Phänotypen – Berechnung der Aufträge / Modulbreite

Nachdem nun alle Phänotypen des Individuums zusammengetragen bzw. berechnet wurden, kann nun seine Fitness berechnet werden. Dazu wird im ersten Schritt bestimmt, wie viele Stellplätze das Kommissioniersystem vorweisen kann (Quellcode 14, Zeile 301). Die dazugehörige Funktion ist in Quellcode 18 beschrieben.

```
108.    def calc_slots(i):
109.        #Anzahl Stellplätze (4 * Gassen * LM hintereinander pro Gasse
110.        # * LM Ebenen * Plätze pro Fach)
111.        s = (4 * i['alg_anz_gassen'] * i['lm_anz_reihe'] *
112.             i['lm_ebenen'] * i['lm_fach_kapazitaet'])
113.        #Anzahl der Stellplätze zurückgeben
114.        return s
```

Quellcode 18: Berechnung der Stellplätze des Kommissioniersystems

Zur Berechnung der eigentlichen Fitness wird zunächst geprüft, ob alle Nebenbedingungen von dem Individuum erfüllt wurden (Quellcode 14, Zeile 306-309). Erst, wenn diese alle erfüllt sind, wird die TCO mit Hilfe des MLPs berechnet. Es wird zunächst der Fall betrachtet, bei dem alle Nebenbedingungen erfüllt wurden (Quellcode 14, Zeile 351-355). Dazu wird

das Dictionary `i` mit den Phänotypen des Kommissioniersystems an die Funktion aus Quellcode 19 übergeben. Dort wird zunächst ein Prädiktor-Dictionary `j` mit den relevanten Phänotypen aus `i` befüllt. Dieser Schritt ist aus Platzgründen hier nicht abgebildet. Damit das MLP eine Schätzung über den TCO abgeben kann, benötigt es einen 18-dimensionalen Vektor $\boldsymbol{x}$ mit Werten im Intervall $[0; 1]$. Dieser wird in Zeile 279 berechnet. Da der Schätzwert des MLPs auch nur im Intervall $[0; 1]$ liegt, muss er wieder hochskaliert werden.

```
209.    def calc_tco(i, model):
210.    #Leeres Prädiktor-Dictionary j wird angelegt
211.    #und mit den relevanten Phänotypen beschrieben
212.    #...
252.        def norm_j(j):
253.            #Einträge in Dictionary j für MLP normalisieren: Intervall [0;1]
254.            j['alg_auftrag_tag'] = (j['alg_auftrag_tag'] - 50) / 7950      #x1
255.            j['alg_max_pos_auftrag'] = (j['alg_max_pos_auftrag'] - 1) / 9  #x2
256.            j['alg_pick_pos'] = (j['alg_pick_pos'] - 1) / 14               #x3
257.            j['alg_anz_gassen'] = (j['alg_anz_gassen'] - 1) / 39           #x4
258.            j['alg_breite_modul'] = (j['alg_breite_modul'] - 1.6) / 4.4    #x5
259.            j['lm_anz_reihe'] = (j['lm_anz_reihe'] - 1) / 29               #x6
260.            j['lm_fachbreite'] = (j['lm_fachbreite'] - 800) / 2000         #x7
261.            j['lm_ebenen'] = (j['lm_ebenen'] - 1) / 5                      #x8
262.            j['lm_kosten'] = (j['lm_kosten'] - 0) / 2500                   #x9
263.            j['lm_fach_kapazitaet'] = (j['lm_fach_kapazitaet'] - 1) / 9    #x10
264.            j['fm_max_a'] = (j['fm_max_a'] - 0.25) / 1.25                  #x11
265.            j['fm_max_v'] = (j['fm_max_v'] - 0.5) / 1.2                    #x12
266.            j['fm_kosten'] = (j['fm_kosten'] - 0) / 4000                   #x13
267.            j['lhm_kosten'] = (j['lhm_kosten'] - 0) / 20                   #x14
268.            j['itk_bearb_zeit'] = (j['itk_bearb_zeit'] - 3) / 37           #x15
269.            j['itk_kosten_kommi'] = (j['itk_kosten_kommi'] - 0) / 200      #x16
270.            j['itk_kosten_fach'] = (j['itk_kosten_fach'] - 0) / 100        #x17
271.            j['itk_sonst'] = (j['itk_sonst'] - 3000) / 47000               #x18
272.
273.            #Werte des Dictionaries werden in Array geschrieben,
274.            #damit das MLP damit rechnen kann
275.            return np.reshape((np.array((list(j.values())), dtype=float)),
276.                              (1, -1))
277.
278.        #Input-Vektor j für MLP-Schätzung linear transformieren [0,1]
279.        j = norm_j(j)
280.
281.        #TCO mit Hilfe des Modells berechnen und anschließend
282.        #wieder hochskalieren
283.        y_min = 30900.8
284.        y_max = 199638059.671175
285.
286.        tco = float(model.predict(j) * (y_max-y_min) + y_min)
287.        #Schätzung muss im Geltungsbereich des Modells bleiben
288.        if tco < y_min:
289.            tco = y_min
290.        elif tco > y_max:
291.            tco = y_max
292.
293.        #TCO wird zurückgegeben
294.        return tco
```

Quellcode 19: Berechnung der Fitness (TCO)

Zur Behandlung von Individuen, welche Nebenfunktionen verletzt haben, wird eine Penalty-Funktion als Mischung bzw. Modifikation aus den in Abschnitt 2.5.6 (S. 72 ff.) behandelten Ansätzen von Deb 2000 und Carlson 1995 verwendet.

Da hier mehrere Nebenbedingungen vorhanden sind, müssen diese normalisiert werden, damit die Schwere ihrer Verletzung vergleichbar ist und nicht durch unterschiedliche Einheiten verfälscht wird. Dazu wird hier die Normalisierungsfunktion nach Carlson 1995 verwendet. Die Implementierung ist in Quellcode 14, Zeile 312-314 zu sehen.

Das Problem bei der Verwendung einer Penalty-Funktion liegt in diesem Fall darin, dass im Vorfeld der Optimierung nicht klar ist, wo genau das Optimum liegt. Weiterhin ist das Optimum nicht konstant an der gleichen Stelle. Wenn man bspw. ein kleines Kommissioniersystem optimieren möchte, ist das Optimum auch sehr klein. Dementsprechend klein muss auch der Penalty-Wert sein. Möchte man allerdings ein sehr großes System optimieren, ist das Optimum und auch der nötige Penalty sehr groß. Dementsprechend schwer ist es, eine Penalty-Funktion zu finden, welche mit der Größe des Systems bzw. mit dem Wert des globalen Optimums skaliert. Es soll deshalb hier von der Idee her der Ansatz zur nicht-parametrisierten Penalty-Funktion von Deb 2000 verwendet werden. Daher wird als Selection-Operator auch die Tournament Selection verwendet.

Dieser Ansatz wurde allerdings insofern abgeändert, als dass hier nicht die Fitness des ungültigen Individuums auf die des schlechtesten Individuums gesetzt wird, sondern auf einen sehr hohen statischen Wert[63] ($399.999.999$). In Abschnitt 4.3.2 wurde angesprochen, dass jedes Individuum aus zwei Fitness-Werten besteht, welche gewichtet werden können (Quellcode 11, Zeile 13). Bei dem zweiten Fitness-Wert handelt es sich um den Penalty-Wert eines Individuums. Dadurch, dass die Fitness aller ungültigen Lösungen auf einem so hohen Wert ist, werden ihnen alle gültigen Lösungen in der Tournament Selection vorgezogen. Sind jedoch nur ungültige Lösungen in einem Tournament, wird das

[63] Dieser Wert kann durch das Metamodell nicht geschätzt werden, denn $y_{max} = 199.638.059{,}67$.

Individuum überleben, welches den niedrigsten Penalty-Wert aufweist, da ihre Fitness-Werte alle gleich hoch sind. Damit sind die drei Fälle der Selection nach Deb 2000 erfüllt.

4.3.4 Selection

Da hier eine parameter-freie Penalty-Funktion nach Deb 2000 verwendet werden soll bzw. auf Grund der Beschaffenheit des Optimierungsproblems verwendet werden muss, bleibt als Selection-Operator nur die Tournament-Selection (TS) über. Hierzu wird die in DEAP vordefinierte TS verwendet (Quellcode 20). Die Größe des Tournaments kann über den Parameter `tournsize` beliebig eingestellt werden, um die Selection-Pressure (siehe dazu Abschnitt 2.5.3, S. 68) zu erhöhen bzw. abzuschwächen.

```
583.    toolbox.register("select", tools.selTournament, tournsize=tournsize)
```

Quellcode 20: Tournament-Selection in DEAP definieren

4.3.5 Crossover

Bei der Betrachtung des Genotyps ohne die angesprochene Sortierung wird der Uniform Crossover (Abschnitt 2.5.4, S. 70) verwendet. Dies liegt vor allem daran, dass die einzelnen Gene untereinander keine Ordnung besitzen, d.h. die Wahl eines FM hat keinen direkten Einfluss auf die benachbarten Gene (z3_LM und z5_LHM). Dementsprechend fallen alle Crossover-Varianten weg, welche den Genotyp in mehrere Teile aufsplitten und diese dann unter den beiden Parents tauschen, bspw. One-Point-Crossover und verwandte Arten. Weiterhin fallen sämtliche Crossover-Varianten für Real Coded GAs ebenfalls weg, da die Gene kategorischer Natur sind (abgesehen von z1 und z2).

```
584.    toolbox.register("mate", tools.cxUniform, indpb=indpb)
```

Quellcode 21: Uniform-Crossover in DEAP definieren

Quellcode 21 zeigt den Uniform-Crossover in DEAP. Die Variable `indpb` beschreibt die Wahrscheinlichkeit, mit der ein Austausch zwischen den Genen der beiden Parents stattfindet.

Es gilt hierbei zu beachten, dass für den Crossover zwei unterschiedliche Wahrscheinlichkeiten verwendet werden: Die `cxpb` beschreibt, mit welcher

Wahrscheinlichkeit zwei Individuen zum Crossover ausgewählt werden, wohingegen die `indpb` die einzelnen Gene der Individuen behandelt.

4.3.6 Mutation

In der unsortierten Betrachtung wird, auf Grund der Zusammensetzung des Genotypen (2 numerische und 4 kategorische Gene), hier kein Standard-Mutations-Operator aus DEAP verwendet.

In der Mutation wird zwischen numerischen und kategorischen Genen unterschieden. Die numerischen Gene werden mit der Creep-Mutation modifiziert, wohingegen die kategorischen Gene mit dem Random-Resetting verändert werden. Quellcode 22 zeigt den dazugehörigen Python-Code.

Da bei kategorischen Genen keinerlei Zusammenhang zwischen deren Inhalt und der Nummerierung besteht, bleibt im Grunde genommen nur die zufällige Auswahl eines neuen Wertes innerhalb des zulässigen Bereichs. Dies geschieht durch das angesprochene Random-Resetting (Zeile 491-492). Dadurch, dass alle Werte des Gens die gleiche Wahrscheinlichkeit besitzen, bei der Mutation ausgewählt zu werden, sind die drei Bedingungen nach Kramer 2017 erfüllt (siehe Abschnitt 2.5.5, S. 70).

Man könnte diesen Operator auch für die numerischen Gene verwenden, allerdings würden dadurch vorhandene Informationen nicht genutzt werden[64]. Deshalb wird zu Mutation der numerischen Gene zusätzlich die Creep-Mutation verwendet. Diese ist in den Zeilen 458-479 zu sehen. Es wird hier eine Normalverteilung benutzt, welche als Mittelwert den aktuellen Wert des Gens enthält. Dies erfüllt die ersten beiden Bedingungen für einen guten Mutations-Operator. Zur Konformität mit dem Dritten skaliert die Standardabweichung der Normalverteilung, mit 10% der Spannweite des Gens. Insgesamt kann man den hier verwendeten Mutations-Operator als einen *Hybriden* aus Random-Resetting und Creep-Mutation bezeichnen.

[64] Wie z.B. „30 Gassen sind mehr als 22 Gassen“ oder „der nächst höhere Schritt von 3 LM pro Gasse sind 4 LM pro Gasse“.

```
452.    def mutate(individual, lower, upper):
453.            k = 0
454.            mutant = individual
455.            #Schleife über alle Gene (Index startet bei 0, deshalb <=5)
456.            while k <= 5:
457.                #Für die numerischen Gene wird die Creep-Mutation verwendet
458.                if k == 0 or k == 1:
459.                    #Feststellen, ob das Gen mutiert wird oder nicht
460.                    if random.random() < mutpb_1:
461.                        #Alter Wert wird zwischengespeichert
462.                        val = individual[k]
463.                        #Die Standardabweichung der Normalverteilung skaliert
464.                        #mit der Spannweite des Gens
465.                        sigma = math.ceil((upper[k] - lower[k]) * 0.1)
466.                        #Wenn ein Gen zur Mutation ausgewählt wurden, soll es auch
467.                        #nach der Mutation einen anderen Wert aufweisen
468.                        while val == individual[k]:
469.                            #Neuen Wert des Genotyps bestimmen (Normalverteilung)
470.                            val = round(np.random.normal(individual[k], sigma))
471.                            #Gen muss nach Mutation noch in dem zulässigen
472.                            #Bereich sein
473.                            if val < lower[k]:
474.                                val = lower[k]
475.                            elif val > upper[k]:
476.                                val = upper[k]
477.                        else:
478.                            #Neuer Wert wird in den Mutanten geschrieben
479.                            mutant[k] = val
480.                #Für die kategorischen Gene wird das Random-Resetting verwendet
481.                else:
482.                    #Feststellen, ob das Gene mutiert wird oder nicht
483.                    if random.random() < mutpb_2:
484.                        #Alter Wert wird zwischengespeichert
485.                        val = individual[k]
486.                        #Wenn ein Gen zur Mutation ausgewähl wurden, soll es auch
487.                        #nach der Mutation einen anderen Wert aufweisen
488.                        while val == individual[k]:
489.                            #Neuer Wert des Gens wird zufällig aus seiner
490.                            #Spannweite bestimmt
491.                            val = np.random.randint(lower[k], upper[k]+1)
492.                        mutant[k] = val
493.                k += 1
494.            #Das mutierte Individuum wird wieder zurückgegeben
495.            return mutant
```

Quellcode 22: Hybride-Mutation

Bei der Mutation sind insgesamt drei Wahrscheinlichkeiten einzustellen: Die `mutpb` (hier nicht zu sehen) beschreibt die Wahrscheinlichkeit, mit der ein Individuum überhaupt in die Mutationsfunktion gelangt, wohingegen die `mutpb_1` festlegt, wie wahrscheinlich es ist, dass die numerischen Gene mutiert werden. Die `mutpb_2` beschreibt das gleiche, allerdings für die kategorischen Gene. Es hat sich bei Vorversuchen herausgestellt, dass die `mutpb_1` in etwa dreimal so hoch sein sollte wie die `mutpb_2`.

4.3.7 Termination

Wie in Abschnitt 2.5.7 (S. 75 ff.) beschrieben, stehen mehrere Möglichkeiten zur Verfügung, den GA zu beenden. Da hier die Anzahl der benötigten Generationen Schwankungen unterliegt, soll keine statische maximale Anzahl von Generationen verwendet werden, sondern gemessen werden, inwiefern die Population des GA in ein bzw. das Optimum konvergiert ist und keine weitere Verbesserung mehr zu erwarten ist. Dazu wird die Funktion aus Quellcode 23 verwendet. Die Variable g gibt dabei die Nummer der aktuellen Generation an. Wenn sich die Durchschnitts-Fitness der Population über `conv_limit` Generationen nicht mehr um mindestens 2% verbessert, soll der GA abgebrochen werden. Dies hat folgende Bewandtnis: Wenn sich die durchschnittliche Fitness einer Population über mehrere Generationen hinweg nicht mehr verbessert, kann davon ausgegangen werden, dass die Population sehr homogen geworden ist. Dadurch liegen alle Individuen in demselben bzw. ähnlichen Optimum und es ist keine weitere Verbesserung der Fitness mehr zu erwarten.

```
497.    def convergence(g, conv_limit, fit_mean):
498.        #Es kann nur geprüft werden, wenn Mindestanzahl von Generationen
499.        #verstrichen sind
500.        if g >= conv_limit:
501.            #Durchschnittsfitness der aktuellen Generation (g-0) bestimmen
502.            mean_fit_0 = fit_mean[g][0]
503.            last_generations_mean = []
504.
505.            for n in range(1, conv_limit+1):
506.                last_generations_mean.append(fit_mean[g-n][0])
507.
508.            #Durchschnittliche Durchschnittsfitness der letzten Generationen
509.            average_last_generations_mean = np.mean(last_generations_mean)
510.            conv = mean_fit_0 / average_last_generations_mean
511.
512.            #Wenn sich die Durchschnitts-Fitness nicht um mindestens 2%
513.            #verbesser hat soll der GA abgebrochen werden
514.            if conv > 0.98:
515.                terminate = True
516.            else:
517.                terminate = False
518.        else:
519.            terminate = False
520.
521.        return terminate
```

Quellcode 23: Termination-Kriterium

Auf Grund der Beschaffenheit der hier verwendeten Genotypen (sehr viele kategorische Variablen) soll als Restart-Strategie lediglich das simple Neustarten des GA mit einer anderen Startpopulation angewendet werden.

4.4 Zusammengesetzter Genetischer Algorithmus

Nun muss die Hauptschleife des GAs aus den einzelnen Bestandteile zusammengesetzt werden. Der erste Teil ist in der Initialisierung (Abschnitt 4.3.2) beschrieben. In Quellcode 24 ist die Berechnung der aktuellen Generation im GA zu sehen. Diese Schleife wird solange wiederholt, bis das Abbruchkriterium (Quellcode 23) erfüllt ist. Eine Evaluation der Anfangspopulation ist dieser Schleife vorgeschaltet. Sie ist identisch zu der in Zeile 670-674.

Die Variable `calcs` hält dabei fest, wie viele Evaluationen der gesamte GA benötigt hat, wohingegen `g` die aktuelle Generation beschreibt und `g_global` die insgesamt verstrichenen Generationen für den Fall, dass Restarts verwendet werden. Diese ist jedoch nur zur späteren Auswertung des GA-Durchlaufes über einen Graphen von Relevanz.

```
640.    #Berechnen der Generationen
641.    while terminate == False:
643.        print("Generation:", g)
644.        #Auswählen der nächsten Generation
645.        offspring = toolbox.select(pop, len(pop))
646.        #Klonen der ausgewählten Individuen
647.        #muss man in Python3 als Liste schreiben
648.        offspring = list(map(toolbox.clone, offspring))
649.
650.        #Crossover
651.        for child1, child2 in zip(offspring[::2], offspring[1::2]):
652.            if random.random() < cxpb:
653.                child1, child2 = toolbox.mate(child1, child2)
654.                #Durch CX ungültig gewordene Fitness löschen
655.                del child1.fitness.values
656.                del child2.fitness.values
657.
658.        #Mutation
659.        for mutant in offspring:
660.            if random.random() < mutpb:
661.                mutant = mutate(mutant, lower, upper)
662.                #Durch Mutation ungültig gewordene Fitness löschen
663.                del mutant.fitness.values
664.
665.        #Bestimmen, welche neuen/unbekannten Individuen dazugekommen sind
666.        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
667.
668.        #Fehlende Fitness-Werte werden ausgerechnet
669.        #Für die Mutanten und Children aus Crossover
670.        for ind in invalid_ind:
671.            calcs += 1
672.            ind.fitness.values = evaluate_individual(
673.                    ind, lm_konfig, fm_konfig,
674.                    lhm_konfig, itk_konfig, slots_plan)
675.
676.        #Alte Generation wird durch die neue Generation abgelöst
677.        pop[:] = offspring
678.        #Hall of Fame wird mit neuer Generation geupdatet
679.        hof.update(pop)
680.
681.        <<<Hier wird eine zweite Population "feasible_pop" erstellt,
682.        diese dient nur zur Bildung der statistischen Werte im logbook.
683.        Sie enthält alle gültigen Individuen>>>
684.        record = stats.compile(pop)
695.        #Logbook updaten
696.        logbook.record(gen=g_global, calcs=calcs,
697.                       feasible=feasible, **record)
698.
699.        #Durchschnittsfitness der bisherigen Generationen abrufen
700.        fit_mean = logbook.select("mean")
701.        #Termination-Kriterium erfüllt? (True/False)
702.        terminate = convergence(g, conv_limit, fit_mean)
703.        if terminate != True:
704.            g += 1
705.            #Nur für die Statistik (wenn Restarts vorhanden sind)
706.            g_global += 1
```

Quellcode 24: Hauptschleife des GA

4.5 Testkonfigurationen

Um ein gutes Bild über die Leistungsfähigkeit des GAs zu bekommen, sollen fünf unterschiedliche Kommissioniersysteme getestet werden. Dabei unterschieden sie sich über die geforderte Mindestanzahl an Stellplätzen, in der Lagergröße sowie über die Leistungsanforderung, über die Anzahl der Aufträge pro Tag. Da die Rechenzeiten recht kurz sind, werden statt der üblichen 50 GA-Durchläufe 100 Optimierungsdurchläufe pro Kommissioniersystem durchgeführt, um ein besseres stochastisches Maß zur Leistung des GAs zu erlangen. Die Ergebnisse dieser Tests sind in Tabelle 21 zu sehen.

Dadurch, dass hier 50 LM, 40 FM, 5 LHM und 10 unterschiedliche ITK zur Verfügung stehen, ergeben sich (zusammen mit der Anzahl der Gassen und LM in Reihe) insgesamt $40 \cdot 30 \cdot 50 \cdot 40 \cdot 5 \cdot 10 = 1{,}2 \cdot 10^8$ Kombinationsmöglichkeiten.

Die Parameter des GA wurden relativ grob durch Expertenwissen und einige Vorversuche weitestgehend angepasst. Da es sich bei der Optimierung von Kommissioniersystemen um ein one-off Problem handelt, wurde auf eine tiefergehende Optimierung der Parameter verzichtet.

Anmerkung: Es wurden für die hier aufgeführten Kommissioniersysteme die globalen Optima nicht explizit ausgerechnet. Es kann nur davon ausgegangen werden, dass es sich bei der Min. Fitness um das globale Optimum für die jeweilige Konfiguration handelt, da in 100 Optimierungsdurchläufen keine bessere Fitness gefunden wurde. Um das globale Optimum mit 100%iger Gewissheit bestimmen zu können, müssten für jede Konfiguration alle $1{,}2 \cdot 10^8$ Kombinationsmöglichkeiten getestet werden, was jedoch einen nicht unerheblichen Zeitaufwand darstellt.

Tabelle 21: Testergebnisse der Kommissioniersysteme

Lagerdaten					
Lagergröße	**sehr klein**	**klein**	**mittel**	**groß**	**sehr groß**
Min. Anzahl Stellplätze	37.500	50.000	100.000	150.000	200.000
Anzahl Aufträge pro Tag	1.500	2.000	4.000	6.000	8.000
Pos/Auftrag	5				
Picks/Pos	10				
GA-Konfiguration					
n_pop	3000				
tournsize	4				
cxpb	0,75				
indpb	0,275				
mutpb	0,15				
mutpb_1	0,725				
mutpb_2	0,175				
conv_limit	10				
restarts	0				
Testergebnisse nach 100 Durchläufen					
Min. Fitness	6.533.278,34	9.770.342,39	25.020.474,36	41.292.567,84	57.880.585,58
Max. Fitness	6.660.248,56	9.920.212,44	25.449.487,57	41.445.822,50	57.880.585,58
Ø Fitness	6.537.869,11	9.854.567,14	25.243.235,08	41.294.100,39	57.880.585,58
Trefferquote für Min. Fitness	96%	32%	41%	99%	100%
Min. Evaluationen	54.711	50.216	47.817	50.012	50.320
Max. Evaluationen	78.466	73.900	71.407	64.921	64.308
Ø Evaluationen	60.307,7	59.508,8	56.379,2	55.552,9	56.432,4
Min. Generationen	22	20	19	20	20
Max. Generationen	32	30	29	26	26
Ø Generationen	24,3	23,9	22,6	22,2	22,6
Min. Rechenzeit [s]	36,5	34,5	28,7	34,6	30,5
Max. Rechenzeit [s]	58,0	57,4	59,3	48,1	42,5
Ø Rechenzeit	42,1	44,8	41,8	39,7	36,1

Da es sich hierbei um ein one-off Problem handelt, liegt dementsprechend der Fokus bei der Beurteilung der Leistungsfähigkeit des GA auf den Fitness-Werten. Wie man in Tabelle 21 erkennen kann liegt die Max. Fitness bei 4 Systemen recht nah an der Min. Fitness. Beim mittleren System liegt sie rund 430.00 Euro über der Min. Fitness, was jedoch bei einem TCO von 25.000.000 Euro noch vertretbar ist. Sie ist lediglich 1,72% schlechter als die Min. Fitness. Abbildung 97 zeigt einen beispielhaften Fitness-Verlauf für das mittlere Kommissioniersystem. Das beste Individuum aus der Anfangspopulation weist eine Fitness von rund $4{,}6 \cdot 10^7$ auf. Diese konnte nach zwei Generationen bereits auf ca. $2{,}8 \cdot 10^7$ gesenkt werden. Ab der 10. Generation nimmt die Fitness den hier minimalen bzw.

optimalen Wert von $2{,}5 \cdot 10^7$ ein. Es sollte hierbei beachtet werden, dass die durchschnittliche Fitness nur über die zulässigen Individuen gebildet wird. Der dazugehörige Vorher/Nachher-Vergleich aus der Hall of Fame ist in Tabelle 22 zu sehen.

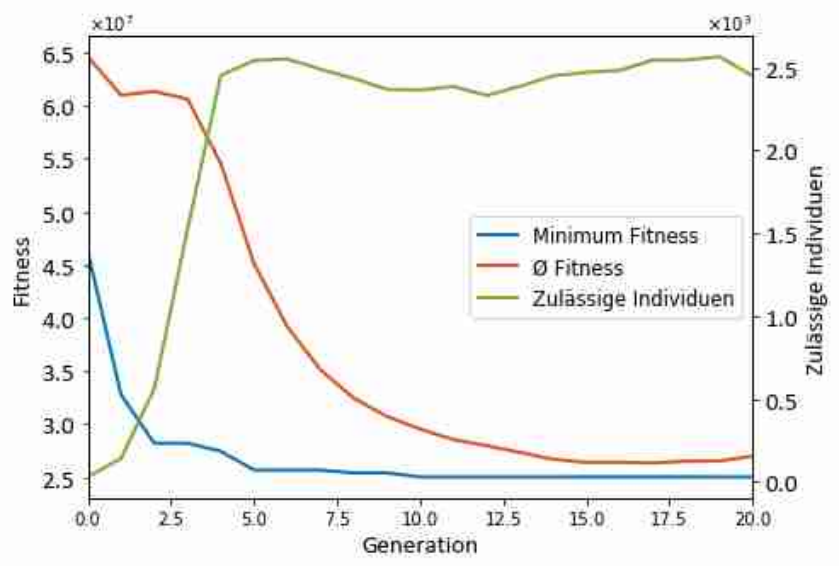


Abbildung 97: Fitness-Verlauf für mittleres Kommissioniersystem

Tabelle 22: Hall of Fame Vorher/Nachher, mittleres Kommissioniersystem

Generation 0				
Top 10	TCO	Penalty	Konfiguration	Stellplätze
1	46.250.579,22	0,0	[31, 21, 34, 14, 4, 3]	109.368
2	46.633.939,35	0,0	[34, 23, 24, 9, 4, 2]	187.680
3	46.820.907,75	0,0	[38, 16, 15, 21, 4, 6]	145.920
4	47.724.096,95	0,0	[39, 23, 32, 7, 4, 2]	179.400
5	48.844.248,37	0,0	[32, 25, 9, 21, 4, 9]	112.000
6	49.331.638,20	0,0	[25, 30, 24, 28, 4, 1]	180.000
7	51.961.093,64	0,0	[38, 19, 46, 14, 4, 4]	144.400
8	52.634.731,16	0,0	[39, 30, 43, 6, 2, 1]	112.320
9	55.819.478,23	0,0	[34, 28, 30, 19, 4, 3]	114.240
10	56.596.954,63	0,0	[27, 20, 24, 1, 4, 8]	129.600
Generation 20				
Top 10	TCO	Penalty	Konfiguration	Stellplätze
1	25.020.474,36	0,0	[14, 30, 15, 7, 4, 4]	100.800
2	25.397.147,79	0,0	[25, 20, 19, 15, 4, 2]	100.000
3	25.527.105,25	0,0	[24, 21, 19, 15, 4, 2]	100.800
4	25.660.210,47	0,0	[23, 22, 19, 15, 4, 2]	101.200
5	25.666.403,17	0,0	[25, 20, 19, 7, 4, 4]	100.000
6	25.666.536,22	0,0	[20, 25, 19, 7, 4, 4]	100.000
7	25.668.101,70	0,0	[24, 21, 19, 7, 4, 4]	100.800
8	25.711.155,10	0,0	[23, 22, 19, 7, 4, 4]	101.200
9	25.746.757,64	0,0	[21, 24, 19, 7, 4, 4]	100.800
10	25.754.523,22	0,0	[22, 23, 19, 7, 4, 4]	101.200

Es zeigt sich anhand des sehr großen Kommissioniersystems, dass der GA auch das Optimum findet, wenn kein einziges Individuum in der Anfangspopulation zulässig ist. Dies liegt hier vor allem daran, dass die Leistungsanforderungen mit 8.000 Aufträgen pro Tag sowie die geforderte Anzahl von Stellplätzen (200.000) sehr hoch sind und somit nur sehr wenige Konfigurationen diese Kriterien überhaupt erfüllen können.

Tabelle 23: Hall of Fame Vorher/Nachher, sehr großes Kommissioniersystem

Generation 0				
Top 10	TCO	Penalty	Konfiguration	Stellplätze
1	399.999.999,00	0,22222	[38, 28, 38, 14, 4, 6]	255.360
2	399.999.999,00	0,29476	[31, 30, 26, 1, 4, 1]	186.000
3	399.999.999,00	0,31427	[38, 20, 38, 14, 4, 6]	182.400
4	399.999.999,00	0,37530	[38, 30, 24, 12, 2, 9]	136.800
5	399.999.999,00	0,41295	[27, 29, 34, 32, 4, 5]	131.544
6	399.999.999,00	0,42718	[24, 27, 26, 15, 4, 7]	129.600
7	399.999.999,00	0,52532	[39, 25, 19, 21, 4, 5]	195.000
8	399.999.999,00	0,53490	[38, 16, 24, 14, 4, 2]	145.920
9	399.999.999,00	0,54777	[19, 30, 19, 15, 4, 2]	114.000
10	399.999.999,00	0,58732	[26, 21, 26, 18, 4, 4]	109.200
Generation 23				
Top 10	TCO	Penalty	Konfiguration	Stellplätze
1	57.880.585,58	0,0	[28, 30, 15, 7, 4, 2]	201.600
2	58.411.276,56	0,0	[29, 29, 15, 7, 4, 2]	201.840
3	58.696.396,67	0,0	[31, 27, 15, 7, 4, 2]	200.880
4	58.804.143,41	0,0	[30, 28, 15, 7, 4, 2]	201.600
5	59.043.458,40	0,0	[28, 30, 15, 7, 4, 1]	201.600
6	59.356.799,25	0,0	[35, 24, 15, 7, 4, 2]	201.600
7	59.500.349,57	0,0	[31, 27, 15, 7, 4, 4]	200.880
8	59.573.309,41	0,0	[29, 29, 15, 7, 4, 1]	201.840
9	59.585.031,54	0,0	[35, 24, 15, 7, 4, 4]	201.600
10	59.706.198,41	0,0	[28, 30, 15, 7, 4, 4]	201.600

In Tabelle 23 sieht man, dass zunächst alle Individuen der Hall of Fame ungültig sind und nur auf Grund ihres Penalty-Wertes die Rangfolge beeinflussen. Nach 23 Generationen wurden jedoch mindesten 10 Konfigurationen gefunden, welche die Kriterien erfüllen können. Dementsprechend zeigt sich, dass der hier gewählte Ansatz, der nicht parametrisierten Penalty-Funktion, auch unter schweren Bedingungen zu funktionieren scheint. Abbildung 98 zeigt den dazugehörigen Fitness-Verlauf. Da hier in den ersten Generationen keine zulässigen Individuen existieren, liegt die durchschnittliche Fitness über alle Individuen dementsprechend bei 399.999.999.

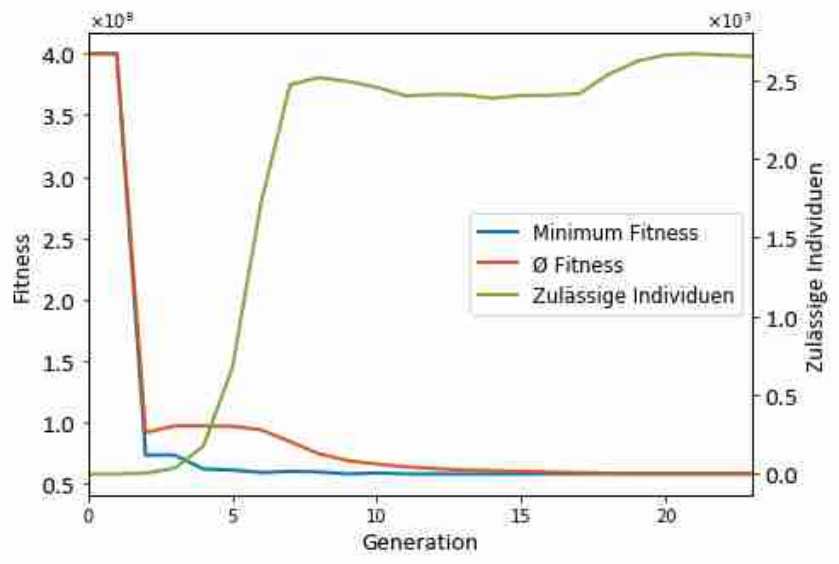


Abbildung 98: Fitness-Verlauf für sehr großes Kommissioniersystem

Überraschend ist jedoch, dass die Trefferquote für die Min. Fitness sehr unterschiedlich ist. Beim sehr kleinen, großen und sehr großen Kommissioniersystem wird die Min. Fitness (fast) immer gefunden, wohingegen die Trefferquote beim kleinen und mittleren System nur bei 32% bzw. 41% liegt. Da die Max. Fitness trotzdem recht nah an der Min. Fitness liegt, kann hier davon ausgegangen werden, dass der GA relativ oft in das zweitbeste Optimum konvergiert. Man könnte nun damit zufrieden sein, dass hier der GA trotzdem immer ein sehr gutes Ergebnis findet oder aus perfektionistischer Sicht die Trefferquote für das globale Optimum weiter steigern.

Um die Trefferquote für das globale Optimum zu erhöhen, können hier mehrere Restarts verwendet werden (siehe Abschnitt 2.5.8, S.76 f.). Dadurch steigt zwar die benötigte Rechenleistung, allerdings wird es auch wahrscheinlicher, dass der GA das globale Optimum findet. Da die Trefferquote für das sehr kleine, große und sehr große Kommissioniersystem ohnehin schon fast perfekt ist (100%), wird hier nur das kleine und mittlere System betrachtet. Im vorliegenden Versuch wurde mit 4 Restarts gearbeitet, dementsprechend wird der GA insgesamt fünf Mal durchlaufen. Die Tournament Size wurde leicht erhöht sowie das Konvergenz-Limit abgesenkt. Dadurch konvergiert der GA etwas schneller. Dies sollte jedoch kein Problem darstellen, da die vorigen Verläufe gezeigt haben, dass in den letzten Generationen des GA ohnehin keine Verbesserung in der Min. Fitness zu verzeichnen ist. Es wurden auch hier wiederum 100 Durchläufe ausgewertet. Die

Ergebnisse sind in Tabelle 24 zu sehen. Die farbigen Zahlen beschreiben die Veränderungen im Vergleich zur GA-Konfiguration ohne Restarts.

Tabelle 24: Testergebnisse der Kommissioniersysteme mit Restarts

Lagerdaten				
Lagergröße	**klein**		**mittel**	
Min. Anzahl Stellplätze	50.000		100.000	
Anzahl Aufträge pro Tag	2.000		4.000	
Pos/Auftrag	5			
Picks/Pos	10			
GA-Konfiguration				
n_pop	3000			
tournsize	5			
cxpb	0,75			
indpb	0,275			
mutpb	0,15			
mutpb_1	0,725			
mutpb_2	0,175			
conv_limit	7			
restarts	4			
Testergebnisse nach 100 Durchläufen				
Min. Fitness	9.770.342,39	-	25.020.474,36	-
Max. Fitness	9.916.673,39	**-0,04%**	25.379.147,79	**-0,28%**
Ø Fitness	9.787.051,97	**-0,70%**	25.061.908,44	**-0,72%**
Trefferquote für Min. Fitness	76%	**+137,50%**	89%	**+117,07%**
Min. Evaluationen	203.578	**+305,40%**	191.855	**+301,23%**
Max. Evaluationen	310.593	**+320,29%**	286.873	**+301,74%**
Ø Evaluationen	241.618,5	**+306,02%**	227.736,5	**+303,94%**
Min. Generationen	76	**+280,00%**	71	**+273,68%**
Max. Generationen	121	**+303,33%**	111	**+282,76%**
Ø Generationen	91,9	**+284,36%**	86,1	**+280,58%**
Min. Rechenzeit [s]	136,8	**+296,98%**	117,3	**+308,07%**
Max. Rechenzeit [s]	216,3	**+276,66%**	179,9	**+203,58%**
Ø Rechenzeit	166,6	**+271,99%**	140,8	**+236,56%**

Es zeigt sich, dass die Verwendung von 4 Restarts zu einem starken Anstieg in der Trefferquote des GA führt. Dementsprechend sinkt auch die durchschnittliche Fitness. Eine 100%ige Trefferquote konnte jedoch bei beiden Systemen nicht erreicht werden. Man müsste hier wohl die Anzahl der Restarts weiter erhöhen.

Dadurch, dass der GA hier im Endeffekt fünf Mal durchlaufen wird, ist die Anzahl der Evaluationen und die damit verbundene Rechenzeit stark angestiegen. Beim kleinen System liegen diese nun bei ca. 240.000 Berechnungen. Setzt man dies jedoch ins Verhältnis zu den $1{,}2 \cdot 10^8$ Kombinationsmöglichkeiten, sind es nur 0,2%, was für eine 76% Sicherheit das globale Optimum gefunden zu haben, recht gering sein sollte. Um sich noch sicherer zu sein das globale Optimum gefunden zu haben, könnte der GA mit Restarts auch mehrmals ausgeführt werden. Bei zweimaliger Ausführung liegt die Unsicherheit noch bei 5,76%[65]. Nach drei Durchläufen bei 1,76%[66]. Hier muss dementsprechend zwischen Unsicherheit und Rechenaufwand abgewogen werden.

Abbildung 99 zeigt den Fitness-Verlauf für das kleine Kommissioniersystem. Die Sprünge in der Fitness zeigen die einzelnen Restarts an. In Tabelle 25 ist die entsprechende Hall of Fame zu sehen.

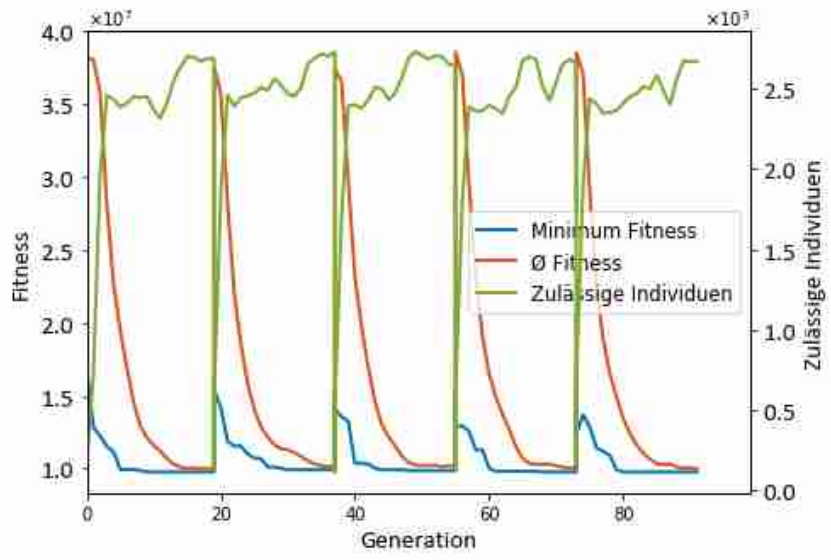


Abbildung 99: Fitness-Verlauf für kleines Kommissioniersystem mit Restarts

Die ermittelten optimalen Konfigurationen für die hier betrachteten Kommissioniersysteme sind im Anhang verzeichnet: sehr klein A-19 (S. 227), klein A-20 (S. 228), mittel A-21 (S. 229), groß A-22 (S. 230) und sehr groß A-23 (S. 231).

[65] $(1 - 0{,}76)^2 = 0{,}0576 \rightarrow 5{,}76\%$
[66] $(1 - 0{,}76)^3 = 0{,}0176 \rightarrow 1{,}76\%$

Tabelle 25: Hall of Fame Vorher/Nachher, kleines Kommissioniersystem mit Restarts

Generation 0				
Top 10	TCO	Penalty	Konfiguration	Stellplätze
1	16.701.733,40	0,0	[11, 24, 15, 15, 4, 6]	63.360
2	16.899.407,29	0,0	[18, 15, 26, 27, 4, 9]	54.000
3	16.973.141,84	0,0	[12, 23, 26, 32, 4, 6]	55.200
4	18.248.309,27	0,0	[21, 30, 19, 16, 3, 6]	50.400
5	18.904.088,47	0,0	[13, 30, 26, 20, 4, 3]	78.000
6	20.158.046,58	0,0	[28, 12, 24, 20, 4, 2]	80.640
7	20.234.720,56	0,0	[37, 15, 5, 25, 4, 1]	53.280
8	20.978.509,67	0,0	[37, 10, 9, 12, 4, 4]	51.800
9	21.080.194,17	0,0	[23, 29, 18, 14, 4, 3]	53.360
10	21.144.336,49	0,0	[26, 21, 2, 6, 4, 9]	52.416
Generation 92				
Top 10	TCO	Penalty	Konfiguration	Stellplätze
1	9.770.342,39	0,0	[7, 30, 24, 25, 4, 4]	50.400
2	9.814.103,40	0,0	[7, 30, 24, 18, 4, 4]	50.400
3	9.863.720,36	0,0	[7, 30, 24, 7, 4, 4]	50.400
4	9.873.029,69	0,0	[7, 30, 15, 7, 4, 4]	50.400
5	9.886.781,36	0,0	[7, 30, 24, 15, 4, 4]	50.400
6	9.916.673,28	0,0	[9, 28, 26, 7, 4, 4]	50.400
7	9.920.212,44	0,0	[9, 28, 19, 7, 4, 4]	50.400
8	9.932.225,32	0,0	[7, 30, 15, 25, 4, 4]	50.400
9	9.987.103,67	0,0	[9, 29, 26, 7, 4, 4]	52.200
10	9.993.409,97	0,0	[7, 30, 15, 15, 4, 4]	50.400

Anmerkung: Um die Leistung des GA zu verbessern, wurde versucht, die Einträge der kategorischen Gene zu ordnen: Also die kategorischen Gene in Ordinale zu transformieren. Dadurch hätte man auch die Creep Mutation für diese Gene verwenden können statt des Random Resetting. Dazu wurden die Einträge pro Gen mit Hilfe der Erkenntnisse aus der Relative Importance (Abschnitt 3.2.3, S. 130) gewichtet und daraus eine Score gebildet. Anschließend wurden die Einträge aufsteigend sortiert, sodass ein niedriger Wert des Gens auch einer niedrigen Score entspricht. Bei Versuchen hat sich jedoch herausgestellt, dass dieses Verfahren bei einigen Lagerkonfigurationen zu besseren Ergebnissen bzgl. der Trefferquote geführt hat, wohingegen bei anderen Konfigurationen die Trefferquote eingebrochen ist bzw. identisch war zur hier verwendeten „normalen“ Herangehensweise. Es war auch kein Muster erkennbar, warum es in machen Fällen funktioniert hat und in anderen nicht. Daher wurde dieser Ansatz hier nicht weiterverfolgt.

5 Fazit und Ausblick

Im ersten Teil der Arbeit hat sich gezeigt, dass für alle vier Zielgrößen bei beiden Betrachtungsweisen, in Relation zu den Spannweiten, sehr gute MLPs erstellt werden konnten. Des Weiteren konnten die wichtigsten Faktoren zur Berechnung der Zielgrößen über RI und ICE-Plots identifiziert werden. Bei der Betrachtung der RI sollte jedoch beachtet werden, dass es insbesondere bei sehr kleinen Werten zu offensichtlichen Fehlinterpretationen kommen kann. Dies liegt wahrscheinlich daran, dass bei der Berechnung der RI durch den Olden Algorithmus die Bias-Werte des MLP nicht berücksichtigt werden. Eine zusätzliche Validierung durch ICE-/c-ICE-Plots sollte hier verwendet werden. Man könnte möglicherweise die Ergebnisse der RI verbessern, indem ein RI-Mittelwert über die RI der besten MLPs aus der Modellauswahl berechnet wird. Dadurch könnten etwaige Fehlinterpretationen herausgefiltert werden.

Bei den vier Zielgrößen hat sich gezeigt, dass die Investitionskosten bei beiden Betrachtungsweisen nahezu identisch sind. Unterschiede waren in den Betriebskosten und insbesondere in der Anzahl der benötigten Kommissionierer auszumachen. Die gug. benötigt wesentlich mehr Kommissionierer als die gg. Betrachtung. Für den Unterschied sorgen hier die Lagergröße sowie die Geschwindigkeit des FM. Beide Faktoren sind in der gg. Betrachtung weitestgehend irrelevant, wohingegen insbesondere die FM-Geschwindigkeit bei der gug. Betrachtung sehr großen Einfluss auf die Anzahl der Kommissionierer auweist. Die nahezu identischen Investitionskosten und die massiven Unterschiede im Personaleinsatz spielgeln sich auch im TCO wieder.

Da die Personalkosten einen hohen Anteil an den Betriebskosten haben und diese wiederum stark auf den TCO wirken, wäre es interessant, diese weiter zu untersuchen. Dazu sollten jedoch spezifischere Faktoren gewählt werden. Bspw. sollte hier die Gassenlänge direkt als Faktor einfließen und nicht über die Fachbreite und LM in Reihe abgebildet werden. Dadurch würden zum einen Prädiktoren eingespart werden und zum anderen wäre dieser eine aussagekräftiger, da er direkt das relevante Maß (Gassenlänge) darstellt. Dies war jedoch notwendig, da aus Gründen der Vergleichbarkeit alle Faktoren berücksichtigt werden mussten. Zudem sollten alle monetären Faktoren außenvor bleiben,

da bei ihnen nicht davon auszugehen ist, dass sie einen Einfluss auf die Anzahl der benötigten Kommissionierer haben.

Weiterhin wurde hier nur das konventionelle Kommissionieren betrachtet. Eine Anwendung der in dieser Arbeit dargestellten Methodik könnte ebenfalls für einen Vergleich unterschiedlicher Kommissioniersystemarten herangezogen werden.

Durch den Einsatz von Metamodellen konnte die benötigte Rechenzeit im Vergleich zum Excel-Tool um den Faktor 1.756 bis 12.830 eingespart werden, dies allerdings nur zu dem Preis der Modellungenauigkeiten der MLPs.

Im zweiten Teil der Arbeit wurde ein leistungsfähiger GA vorgestellt, welcher auch nur bei sehr wenigen überhaupt zulässigen Lösungen eine optimale Konfiguration des Kommissioniersystems findet. Selbst ohne Restarts liegt die maximale Abweichung der Optimierung nur 1,94% (sehr kleines Kommissioniersystem) über dem globalen Optimum. Es hat sich jedoch auch gezeigt, dass zwischen den Konfigurationen erhebliche Unterschiede in der Trefferquote für das globale Optimum bestehten. Offenbar konvergiert der GA bei einigen Konfigurationen oftmals in das Suboptimum, wohingegen bei anderen Konfigurationen (fast) immer das globale Optimum gefunden wird. Warum es hier solche Unterschiede gibt bleibt unklar. Die Verwendung einiger Restarts kann jedoch dazu beitragen, die Trefferquote für das globale Optimum zu steigern. Eine Garantie dafür, dass das globale Optimum in jedem Fall gefunden wird, kann jedoch nur die Berechnung aller $1{,}2 \cdot 10^8$ Kombinationsmöglichkeiten liefern. Allerdings sollte man hier im Hinterkopf behalten, dass die Fitness-Werte auf der Schätzung eines MLPs beruhen, welches ebenfalls eine gewisse Ungenauigkeit aufweist (siehe Abbildung 57, S.120). Daher sollte der GA lediglich zur Vorauswahl von potenziell optimalen Kommissioniersystemkonfigurationen eingesetzt werden. Diese müssten anschließend z.B. durch Simulationen tiefergehend untersucht und verglichen werden.

Als nächster Schritt könnten sämtliche Funktionen des Excel-Tools in Python implementiert und direkt zur Berechnung der Fitness verwendet werden. Dadurch würde zum einen der Umweg über ein Metamodell und die damit verbundenen Modellungenauigkeiten eingespart, zum anderen könnten dadurch die in der Versuchsplanung festgelegten

kategorischen Faktoren (z.B. unterschiedliche Wegstrategien usw.) als zusätzliche Gene in die Optimierung durch einen GA mit einbezogen werden. Dadurch fänden wesentlich mehr unterschiedliche Varianten Berücksichtigung.

Weiterhin könnte untersucht werden, ob die Möglichkeit besteht, die Komponenten des Kommissioniersystem (FM, LM usw.) sinnvoll bzgl. der untersuchten Zielgröße zu sortieren und sie somit nicht als kategorische, sondern als ordinale Gene zu betrachten. Dadurch kämen andere Operatoren zum Crossover oder zur Mutation in Betracht, die möglicherweise die Leistung des GAs steigern.

6 Literaturverzeichnis

Antonov, I. A.; Saleev, V. M. (1979): An economic method of computing LPτ-sequences. Экономичный способ вычисления ЛПτ-последовательностей. In: *USSR Computational Mathematics and Mathematical Physics (Ж. вычисл. матем. и матем. физ.)* 19 (1), S. 252–256. DOI: 10.1016/0041-5553(79)90085-5.

Arumugam, M. Senthil; Rao, M.V.C. (2007): Novel Hybrid Approaches For Real Coded Genetic Algorithm To Compute The Optimal Control Of A Single Stage Hybrid Manufacturing Systems. In: *International Journal of Computer and Information Engineering* 1 (7), 2304-2321. DOI: 10.1999/1307-6892/422.

Bellman, Richard E. (1961): Computational Aspects of Dynamic Programming. In: Richard E. Bellman (Hg.): Adaptive Control Processes. A Guided Tour. Princeton: Princeton University Press (Princeton legacy library), S. 85–99.

Bergstra, James; Bengio, Yoshua (2012): Random Search for Hyper-Parameter Optimization. In: *Journal of Machine Learning Research* 13 (2), S. 281–305. Online verfügbar unter http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf.

Bichler, Klaus; Krohn, Ralf; Riedel, Guido; Schöppach, Frank (2010): Beschaffungs- und Lagerwirtschaft. Praxisorientierte Darstellung der Grundlagen, Technologien und Verfahren. 9., aktualisierte und überarbeitete Auflage. Wiesbaden: Gabler.

Bratley, Paul; Fox, Bennett L. (1988): Algorithm 659: Implementing Sobol's quasirandom sequence generator. In: *ACM Transactions on Mathematical Software (TOMS)* 14 (1), S. 88–100. DOI: 10.1145/42288.214372.

Briscoe, Erica; Feldman, Jacob (2011): Conceptual complexity and the bias/variance tradeoff. In: *Cognition* 118 (1), S. 2–16. DOI: 10.1016/j.cognition.2010.10.004.

Buscema, Massimo (1998): Back Propagation Neural Networks. In: *Substance Use & Misuse* 33 (2), S. 233–270. DOI: 10.3109/10826089809115863.

Carlson, Susan E. (1995): A General Method for Handling Constraints in Genetic Algorithms. In: Proceedings of the Second Annual Joint Conference on Information

Science. Second Annual Joint Conference on Information Science. Wrightsville Beach, NC, USA, 28. September - 01. Oktober 1995, S. 663–666. Online verfügbar unter http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.6822, zuletzt geprüft am 19.09.2018.

Chai, T.; Draxler, R. R. (2014): Root mean square error (RMSE) or mean absolute error (MAE)? Arguments against avoiding RMSE in the literature. In: *Geosci. Model Dev.* 7 (3), S. 1247–1250. DOI: 10.5194/gmd-7-1247-2014.

CIRP (2014): Statistics. Normalization. Coastal Inlets Research Program (CIRP). Online verfügbar unter https://cirpwiki.info/wiki/Statistics#Normalization, zuletzt aktualisiert am 05.06.2014, zuletzt geprüft am 14.10.2018.

Claesen, Marc; Moor, Bart de (2015): Hyperparameter Search in Machine Learning. In: *MIC 2015: The XI Metaheuristics International Conference in Agadir, Morocco* 11. Online verfügbar unter http://arxiv.org/pdf/1502.02127.

Claesen, Marc; Simm, Jaak; Popovic, Dusan; Moreau, Yves; Moor, Bart de (2014): Easy Hyperparameter Search Using Optunity, 02.12.2014. Online verfügbar unter http://arxiv.org/pdf/1412.1114.

Clune, Jeff; Goings, Sheni; Punch, Bill; Goodman, Eric (2005): Investigations in Meta-GAs. Panaceas or Pipe Dreams? In: Franz Rothlauf (Hg.): Proceedings of the 2005 workshops on Genetic and evolutionary computation - GECCO '05. the 2005 workshops. Washington, D.C. New York, New York, USA: ACM Press, S. 235–241. Online verfügbar unter http://www.evolvingai.org/clune-goings-goodman-punch-2005-investigations-meta-gas, zuletzt geprüft am 17.08.2018.

Dao, Son Duy; Abhary, Kazem; Marian, Romeo (2016): An improved structure of genetic algorithms for global optimisation. In: *Prog Artif Intell* 5 (3), S. 155–163. DOI: 10.1007/s13748-016-0091-3.

Das, Amit Kumar; Pratihar, Dilip Kumar (2018): A Novel Restart Strategy for Solving Complex Multi-modal Optimization Problems Using Real-Coded Genetic Algorithm. In: Ajith Abraham, Pranab Kr Muhuri, Azah Kamilah Muda und Niketa Gandhi (Hg.):

Intelligent systems design and applications, Bd. 736. New York NY: Springer Berlin Heidelberg (Advances in Intelligent Systems and Computing), S. 32–41.

Deb, Kalyanmoy (2000): An efficient constraint handling method for genetic algorithms. In: *Computer Methods in Applied Mechanics and Engineering* 186 (2-4), S. 311–338. DOI: 10.1016/S0045-7825(99)00389-8.

Deb, Kalyanmoy (2001): Multi-Objective Optimization Using Evolutionary Algorithms. New York, NY, USA: John Wiley & Sons, Inc. Online verfügbar unter https://books.google.de/books?id=OSTn4GSy2uQC.

Deep, Kusum; Singh, Krishna Pratap; Kansal, M. L.; Mohan, C. (2009): A real coded genetic algorithm for solving integer and mixed integer optimization problems. In: *Applied Mathematics and Computation* 212 (2), S. 505–518. DOI: 10.1016/j.amc.2009.02.044.

Dickel, Patrick (2017): Interpretationsmethoden für Machine-Learning-Algorithmen. Unveröffentlichte Studienarbeit im Master Wirtschaftsingenieurwesen. Universität Siegen, Siegen.

Duchi, John; Hazan, Elad; Singer, Yoram (2011): Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. In: *The Journal of Machine Learning Research* 12, S. 2121–2159. Online verfügbar unter http://dl.acm.org/ft_gateway.cfm?id=2021068&type=pdf.

Ebert, Tobias; Fischer, Torsten; Belz, Julian; Heinz, Tim Oliver; Kampmann, Geritt; Nelles, Oliver (2015): Extended Deterministic Local Search Algorithm for Maximin Latin Hypercube Designs. In: 2015 IEEE Symposium Series on Computational Intelligence. IEEE SSCI 2015. Cape Town, South Africa, 07.12.2015 - 10.12.2015. Institute of Electrical and Electronics Engineers (IEEE). Piscataway, NJ: IEEE, S. 375–382.

Eiben, Agoston E.; Smit, S. K. (2012): Evolutionary Algorithm Parameters and Methods to Tune Them. In: Youssef Hamadi, Eric Monfroy und Frédéric Saubion (Hg.): Autonomous Search. 2011. Aufl. Berlin: Springer Berlin, S. 15–36.

Eiben, Agoston E.; Smith, James E. (2010): Introduction to Evolutionary Computing. Berlin, Heidelberg: Springer (Natural computing series).

Fortin, Félix-Antoine; Rainville, François-Michel De; Gardner, Marc-André; Parizeau, Marc; Gagné, Christian (2012): DEAP: Evolutionary Algorithms Made Easy. In: *Journal of Machine Learning Research* 13, S. 2171–2175. Online verfügbar unter http://www.jmlr.org/papers/volume13/fortin12a/fortin12a.pdf, zuletzt geprüft am 23.09.2018.

Fortmann-Roe, Scott (2012): Understanding the Bias-Variance Tradeoff. Online verfügbar unter http://scott.fortmann-roe.com/docs/BiasVariance.html, zuletzt geprüft am 30.07.2018.

Fraunhofer-Gesellschaft (2018): TCO - Total Cost of Ownership. Unter Mitarbeit von Sven Bärenfänger-Wojciechowski. Fraunhofer-Institut für Materialfluss und Logistik IML. Online verfügbar unter https://www.iml.fraunhofer.de/de/abteilungen/b2/anlagenmanagement/tco.html, zuletzt geprüft am 12.09.2018.

Friedman, Jerome H. (2001): Greedy Function Approximation: A Gradient Boosting Machine. In: *The Annals of Statistics* 29 (5), S. 1189–1232. DOI: 10.1214/aos/1013203451.

Friedman, Jerome H.; Meulman, Jacqueline J. (2003): Multiple additive regression trees with application in epidemiology. In: *Statistics in medicine* 22 (9), S. 1365–1381. DOI: 10.1002/sim.1501.

Garud, Sushant S.; Karimi, Iftekhar A.; Kraft, Markus (2017): Design of computer experiments: A review.

Geman, Stuart; Bienenstock, Elie; Doursat, René (1992): Neural Networks and the Bias/Variance Dilemma. In: *Neural Computation* 4 (1), S. 1–58. DOI: 10.1162/neco.1992.4.1.1.

Ghannadian, Farzad; Alford, Cecil; Shonkwiler, Ron (1996): Application of random restart to genetic algorithms. In: *Information Sciences* 95 (1-2), S. 81–102. DOI: 10.1016/S0020-0255(96)00121-1.

Ghoreishi, Seyyedeh Newsha; Clausen, Anders; Joergensen, Bo Noerregaard (2017): Termination Criteria in Evolutionary Algorithms: A Survey. In: Christophe Sabourin (Hg.): IJCCI 2017. Proceedings of the 9th International Joint Conference on Computational Intelligence, November 1-3, 2017, Funchal, Madeira, Portugal: SCITEPRESS Digital Library, S. 373–384.

Gleißner, Harald; Femerling, Christian (2008): Logistik. Grundlagen - Übungen - Fallbeispiele. 1. Auflage. Wiesbaden: Gabler (Lehrbuch).

Glorot, Xavier; Bordes, Antoine; Bengio, Yoshua (2011): Deep Sparse Rectifier Neural Networks. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, S. 315–323. Online verfügbar unter https://www.researchgate.net/publication/215616967_Deep_Sparse_Rectifier_Neural_Networks, zuletzt geprüft am 31.07.2018.

Goh, Anthony T.C. (1995): Back-propagation neural networks for modeling complex systems. In: *Artificial Intelligence in Engineering* 9 (3), S. 143–151. DOI: 10.1016/0954-1810(94)00011-S.

Goldberg, David E.; Deb, Kalyanmoy (1991): A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In: Gregory J. E. Rawlins (Hg.): Foundations of Genetic Algorithms 1991 (FOGA 1),Volume 1, Bd. 1. Burlington: Elsevier Science (Foundations of Genetic Algorithms, Volume 1), S. 69–93.

Goldberg, David Edward (1989): Genetic algorithms in search, optimization, and machine learning. 2. Aufl. Reading, Massachusetts, USA: Addison-Wesley Publishing Company, Inc.

Goldstein, Alex; Kapelner, Adam; Bleich, Justin; Pitkin, Emil (2014): Peeking Inside the Black Box. Visualizing Statistical Learning With Plots of Individual Conditional

Expectation. In: *Journal of Computational and Graphical Statistics* 24 (1), S. 44–65. DOI: 10.1080/10618600.2014.907095.

Gudehus, Timm (2010): Logistik. Grundlagen - Strategien - Anwendungen. 4., aktualisierte Auflage. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg. Online verfügbar unter http://dx.doi.org/10.1007/978-3-540-89389-9.

Gudehus, Timm; Baginski, Ralf; Forstner, Michael Freiherr von (2008): Technische Logistiksysteme. Kommissioniersysteme. In: Dieter Arnold, Heinz Isermann, Axel Kuhn, Horst Tempelmeier und Kai Furmans (Hg.): Handbuch Logistik. 3., neu bearbeitete Aufl. Berlin, Heidelberg: Springer, S. 668–685.

Hagan, Martin T.; Demuth, Howard B.; Beale, Mark Hudson; Jesus, Orlando de (2014): Neural Network Design. 2nd Edition. 2. Aufl.: Martin Hagan. Online verfügbar unter http://hagan.okstate.edu/nnd.html.

Hammerstrom, Dan (1993): Working with neural networks. In: *IEEE Spectrum* 30 (7), S. 46–53. DOI: 10.1109/6.222230.

Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome H. (2017): The elements of statistical learning. Data mining, inference, and prediction. Second edition, corrected at 12th printing 2017. New York: Springer New York; Springer Berlin (Springer series in statistics). Online verfügbar unter https://web.stanford.edu/~hastie/ElemStatLearn/, zuletzt geprüft am 08.08.2018.

Herrera, F.; Lozano, M.; Verdegay, J. L. (1998): Tackling Real-Coded Genetic Algorithms. Operators and Tools for Behavioural Analysis. In: *Artificial Intelligence Review* 12 (4), S. 265–319. DOI: 10.1023/A:1006504901164.

Hinton, Geoffrey E. (1998): A Practical Guide to Training Restricted Boltzmann Machines. In: Genevieve B. Orr und Klaus-Robert Müller (Hg.): Neural Networks. Tricks of the Trade. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science, 1524), S. 599–619.

Hompel, Michael ten; Sadowsky, Voker; Beck, Maria (2011): Kommissionierung. Materialflusssysteme 2 – Planung und Berechnung der Kommissionierung in der Logistik. Berlin, Heidelberg, New York: Springer (Intralogistik).

Hompel, Michael ten; Zellerhoff, Jörg; Pelka, Michael; Crostack, Horst-Arthur; Mathis, Jonas; Strothotte, Daniel (2010): Strategien für die flexible, auftragsweise Kommissionierung mit integrierter Prüfung. Projekt-Nr. 15811 N.

IEEE 754-2008, 29.08.2008: IEEE Standard for Floating-Point Arithmetic. Online verfügbar unter https://ieeexplore.ieee.org/document/4610935/, zuletzt geprüft am 22.08.2018.

Joe, Stephen; Kuo, Frances Y. (2008): Notes on generating Sobol' sequences. Online verfügbar unter http://web.maths.unsw.edu.au/~fkuo/sobol/, zuletzt geprüft am 02.09.2018.

Johnson, M. E.; Moore, L. M.; Ylvisaker, D. (1990): Minimax and maximin distance designs. In: *Journal of Statistical Planning and Inference* 26 (2), S. 131–148. DOI: 10.1016/0378-3758(90)90122-B.

Kingma, Diederik P.; Ba, Jimmy Lei (2015): Adam: A Method for Stochastic Optimization. In: *Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, USA, 2015*. Online verfügbar unter http://arxiv.org/pdf/1412.6980.

Kleppmann, Wilhelm (2013): Versuchsplanung. Produkte und Prozesse optimieren. 8., überarbeitete Auflage. München, Wien: Carl Hanser Verlag (Praxisreihe Qualitätswissen).

VDI-Richtlinie VDI 3590 Blatt 1, April 1994: Kommissioniersysteme - Grundlagen.

Kramer, Oliver (2017): Genetic Algorithm Essentials. Cham, Switzerland: Springer International Publishing AG (Studies in Computational Intelligence, Volume 679).

Kramer, Oliver; Schlachter, Uli; Spreckels, Valentin (2013): An adaptive penalty function with meta-modeling for constrained problems. In: IEEE Congress on Evolutionary Computation (CEC). 20 - 23 June 2013, Cancún, México. Institute of Electrical and

Electronics Engineers; IEEE Computational Intelligence Society; IEEE Congress on Evolutionary Computation; CEC. Piscataway, NJ: IEEE, S. 1350–1354.

LeCun, Yann; Bottou, Leon; Orr, Genevieve B.; Müller, Klaus-Robert (1998): Efficient BackProp. In: Genevieve B. Orr und Klaus-Robert Müller (Hg.): Neural Networks. Tricks of the Trade, Bd. 1524. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science, 1524), S. 9–50.

Lim, Siew Mooi; Sultan, Abu Bakar Md.; Sulaiman, Md. Nasir; Mustapha, Aida; Leong, K. Y. (2017): Crossover and Mutation Operators of Genetic Algorithms. In: *International Journal of Machine Learning and Computing (IJMLC)* 7 (1), S. 9–12. DOI: 10.18178/ijmlc.2017.7.1.611.

Loshchilov, Ilya (2013): CMA-ES with restarts for solving CEC 2013 benchmark problems. In: IEEE Congress on Evolutionary Computation (CEC). 20 - 23 June 2013, Cancún, México. Institute of Electrical and Electronics Engineers; IEEE Computational Intelligence Society; IEEE Congress on Evolutionary Computation; CEC. Piscataway, NJ: IEEE, S. 369–376. Online verfügbar unter https://hal.inria.fr/hal-00823880, zuletzt geprüft am 18.08.2018.

Maas, Andrew L.; Hannun, Awni Y.; Ng, Andrew Y. (2013): Rectifier Nonlinearities Improve Neural Network Acoustic Models. Computer Science Department, Stanford University. Stanford, CA 94305 USA. Online verfügbar unter https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf, zuletzt geprüft am 02.08.2018.

Maclaurin, Dougal; Duvenaud, David; Adams, Ryan P. (2015): Gradient-based Hyperparameter Optimization through Reversible Learning. In: *Proceeding ICML'15 Proceedings of the 32nd International Conference on International Conference on Machine Learning* 37. Online verfügbar unter http://arxiv.org/pdf/1502.03492.

Martin, Heinrich (2016): Transport- und Lagerlogistik. Systematik, Planung, Einsatz und Wirtschaftlichkeit. 10. Aufl. Wiesbaden: Springer Fachmedien Wiesbaden GmbH.

Mendelsohn, Lou (2018): Preprocessing Data for Neural Networks. Market Technologies, LLC. Online verfügbar unter https://www.vantagepointsoftware.com/mendelsohn/preprocessing-data-neural-networks/, zuletzt aktualisiert am 10.04.2018, zuletzt geprüft am 11.04.2018.

Michalewicz, Zbigniew (1996): Genetic Algorithms + Data Structures. Evolution Programs. Third, Revised and Extended Edition. Berlin, Heidelberg: Springer. Online verfügbar unter http://dx.doi.org/10.1007/978-3-662-03315-9.

Miller, Brad L.; Goldberg, David E. (1995): Genetic Algorithms, Tournament Selection, and the Effects of Noise. In: *Complex Systems* 9 (3), S. 193–212.

Nielsen, Michael A. (2015): Neural Networks and Deep Learning. Determination Press. Online verfügbar unter http://neuralnetworksanddeeplearning.com, zuletzt geprüft am 05.08.2018.

Olden, Julian D.; Jackson, Donald A. (2002): Illuminating the “black box”. A randomization approach for understanding variable contributions in artificial neural networks. In: *Ecological Modelling* 154 (1-2), S. 135–150. DOI: 10.1016/S0304-3800(02)00064-9.

Olden, Julian D.; Joy, Michael K.; Death, Russell G. (2004): An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. In: *Ecological Modelling* 178 (3-4), S. 389–397. DOI: 10.1016/j.ecolmodel.2004.03.013.

Paliwal, Mukta; Kumar, Usha A. (2011): Assessing the contribution of variables in feed forward neural network. In: *Applied Soft Computing* 11 (4), S. 3690–3696. DOI: 10.1016/j.asoc.2011.01.040.

Pedregosa, Fabian; Varoquaux, Gaël; Gramfort, Alexandre; Michel, Vincent; Thirion, Bertrand; Grisel, Olivier et al. (2011): Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12, S. 2825–2830. Online verfügbar unter http://scikit-learn.org/stable.

Prechelt, Lutz (1998): Early Stopping - But When? In: Genevieve B. Orr und Klaus-Robert Müller (Hg.): Neural Networks. Tricks of the Trade, Bd. 1524. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science, 1524), S. 55–69.

Riedmiller, Martin (1994): Rprop - Description and Implementation Details. Technical Report. University of Karlsruhe. Karlsruhe.

Ring, Christina; Ryll, Andreas; Gaus, Wilhelm (2006): Das Bestimmtheitsmaß R^2 bei linearen Regressionsmodellen mit und ohne Intercept - die Tücken der Statistikprogramme. In: *WIST* 35 (11), S. 607–612. DOI: 10.15358/0340-1650-2006-11-607.

Ruder, Sebastian (2016): An overview of gradient descent optimization algorithms. In: *CoRR* abs/1609.04747. Online verfügbar unter http://arxiv.org/pdf/1609.04747, zuletzt geprüft am 15.08.2018.

Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (1986a): Learning Internal Representations by Error Propagation. In: David E. Rumelhart und James L. McClelland (Hg.): Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations. Cambridge, MA, USA: MIT Press, S. 318–362.

Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (1986b): Learning representations by back-propagating errors. In: *Nature* 323 (6088), S. 533. DOI: 10.1038/323533a0.

Safe, Martín; Carballido, Jessica; Ponzoni, Ignacio; Brignole, Nélida (2004): On Stopping Criteria for Genetic Algorithms. In: Ana L. C. Bazzan und Sofiane Labidi (Hg.): Advances in artificial intelligence, SBIA 2004. 17th Brazilian Symposium on Artificial Intelligence, São Luis, Maranhão, Brazil, September 29 - October 1, 2004 : proceedings, Bd. 3171. Berlin, London: Springer (Lecture notes in computer science, Lecture notes in artificial intelligence 0302-9743, 3171), S. 405–413.

Santner, Thomas J.; Williams, Brian; Notz, William (2003): Design and analysis of computer experiments. New York: Springer (Springer series in statistics). Online

verfügbar unter http://www.loc.gov/catdir/enhancements/fy0817/2003045444-d.html.

Sarker, Ruhul; Runarsson, Thomas; Newton, Charles (2001): Genetic Algorithms for Solving a Class of Constrained Nonlinear Integer Programs. In: *Int Trans Operational Res* 8 (1), S. 61–74. DOI: 10.1111/1475-3995.00006.

Schwefel, Hans-Paul (1995): Evolution and Optimum Seeking. New York, NY, USA: John Wiley & Sons, Inc. (Sixth-generation computer technology series). Online verfügbar unter http://ls11-www.cs.tu-dortmund.de/lehre/wiley, zuletzt geprüft am 22.08.2018.

Shevchuk, Yurii (2015): Visualize Algorithms based on the Backpropagation. NeuPy - Neural Networks in Python. Online verfügbar unter http://neupy.com/2015/07/04/visualize_backpropagation_algorithms.html, zuletzt geprüft am 10.08.2018.

Siebertz, Karl; van Bebber, David; Hochkirchen, Thomas (2017): Statistische Versuchsplanung. Design of Experiments (DoE). 2. Aufl. 2017. Berlin, Heidelberg: Springer Berlin Heidelberg (VDI-Buch). Online verfügbar unter http://dx.doi.org/10.1007/978-3-662-55743-3.

Sobol', Ilya M. (1967): On the distribution of points in a cube and the approximate evaluation of integrals. О распределении точек в кубе и приближенном вычислении интегралов. In: *USSR Computational Mathematics and Mathematical Physics (Ж. вычисл. матем. и матем. физ.)* 7 (4), S. 86–112. DOI: 10.1016/0041-5553(67)90144-9.

Sobol', Ilya M. (1998): On quasi-Monte Carlo integrations. In: *Mathematics and Computers in Simulation* 47 (2-5), S. 103–112. DOI: 10.1016/S0378-4754(98)00096-2.

Sobol', Ilya M.; Asotsky, Danil; Kreinin, Alexander; Kucherenko, Sergei (2011): Construction and Comparison of High-Dimensional Sobol' Generators. In: *Wilmott Magazine* 2011 (56), S. 64–79. DOI: 10.1002/wilm.10056.

Thomas, Alan J.; Petridis, Miltos; Walters, Simon D.; Gheytassi, Saeed Malekshahi; Morgan, Robert E. (2015): On Predicting the Optimal Number of Hidden Nodes. In: Hamid R. Arabnia, Leonidas Deligiannidis und Quoc-Nam Tran (Hg.). 2015 International Conference on Computational Science and Computational Intelligence (CSCI). Las Vegas, NV, USA, 7.12.2015 - 9.12.2015. Computational Science & Computational Intelligence (CSCI'15). Piscataway, NJ: IEEE, S. 565–570.

Thomas, Frank; Schiffer, Ingo; Lenk, Bernhard; Sotriffer, Ingomar; Bär, Norbert; Kindermann, Thomas et al. (2008): Technische Logistiksysteme. Informationstechnik für Logistiksysteme. In: Dieter Arnold, Heinz Isermann, Axel Kuhn, Horst Tempelmeier und Kai Furmans (Hg.): Handbuch Logistik. 3., neu bearbeitete Aufl. Berlin, Heidelberg: Springer, S. 608–872.

Tieleman, T.; Hinton, G. (2012): Lecture 6.5 - RMSProp. COURSERA: Neural Networks for Machine Learning. Technical Report. University of Toronto, Toronto, Kanada.

Umbarkar, Anan J.; Sheth, Pranali D. (2015): Crossover Operators in Genetic Algorithms: A Review. In: *ICTACT Journal on Soft Computing* 06 (01), S. 1083–1092. DOI: 10.21917/ijsc.2015.0150.

Walde, Janette F. (2005): Design Künstlicher Neuronaler Netze. Ein Leitfaden zur effizienten Handhabung mehrschichtiger Perzeptrone. Dissertation Universität Innsbruck, 2000. 1. Aufl. Wiesbaden: Deutscher Universitäts-Verlag.

Wang, Huanjing; Xing, Guangming; Chen, Kairui (2008): Categorical Data Transformation Methods for Neural Networks. In: Proceedings of the 2008 International Conference on Information and Knowledge Engineering. Online verfügbar unter http://works.bepress.com/huanjing_wang/5/, zuletzt geprüft am 11.04.2018.

Welling, Soeren H.; Refsgaard, Hanne H. F.; Brockhoff, Per B.; Clemmensen, Line H. (2016): Forest Floor Visualizations of Random Forests, 04.07.2016. Online verfügbar unter http://arxiv.org/pdf/1605.09196.

7 Anhang

A-1 Rechnerkonfiguration

System Information report written at: 10/11/18 14:53:24
[System Summary]

Item	Value
OS Name	Microsoft Windows 10 Pro
Version	10.0.17763 Build 17763
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Manufacturer	Gigabyte Technology Co., Ltd.
System Model	H87-HD3
System Type	x64-based PC
System SKU	To be filled by O.E.M.
Processor	Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz, 3401 Mhz, 4 Core(s), 4 Logical Processor(s)
BIOS Version/Date	American Megatrends Inc. F10, 18.08.2015
SMBIOS Version	2.7
Embedded Controller Version	255.255
BIOS Mode	UEFI
BaseBoard Manufacturer	Gigabyte Technology Co., Ltd.
BaseBoard Product	H87-HD3
BaseBoard Version	x.x
Installed Physical Memory (RAM)	8,00 GB
Total Physical Memory	7,89 GB
Available Physical Memory	3,80 GB
Total Virtual Memory	9,14 GB
Available Virtual Memory	3,54 GB

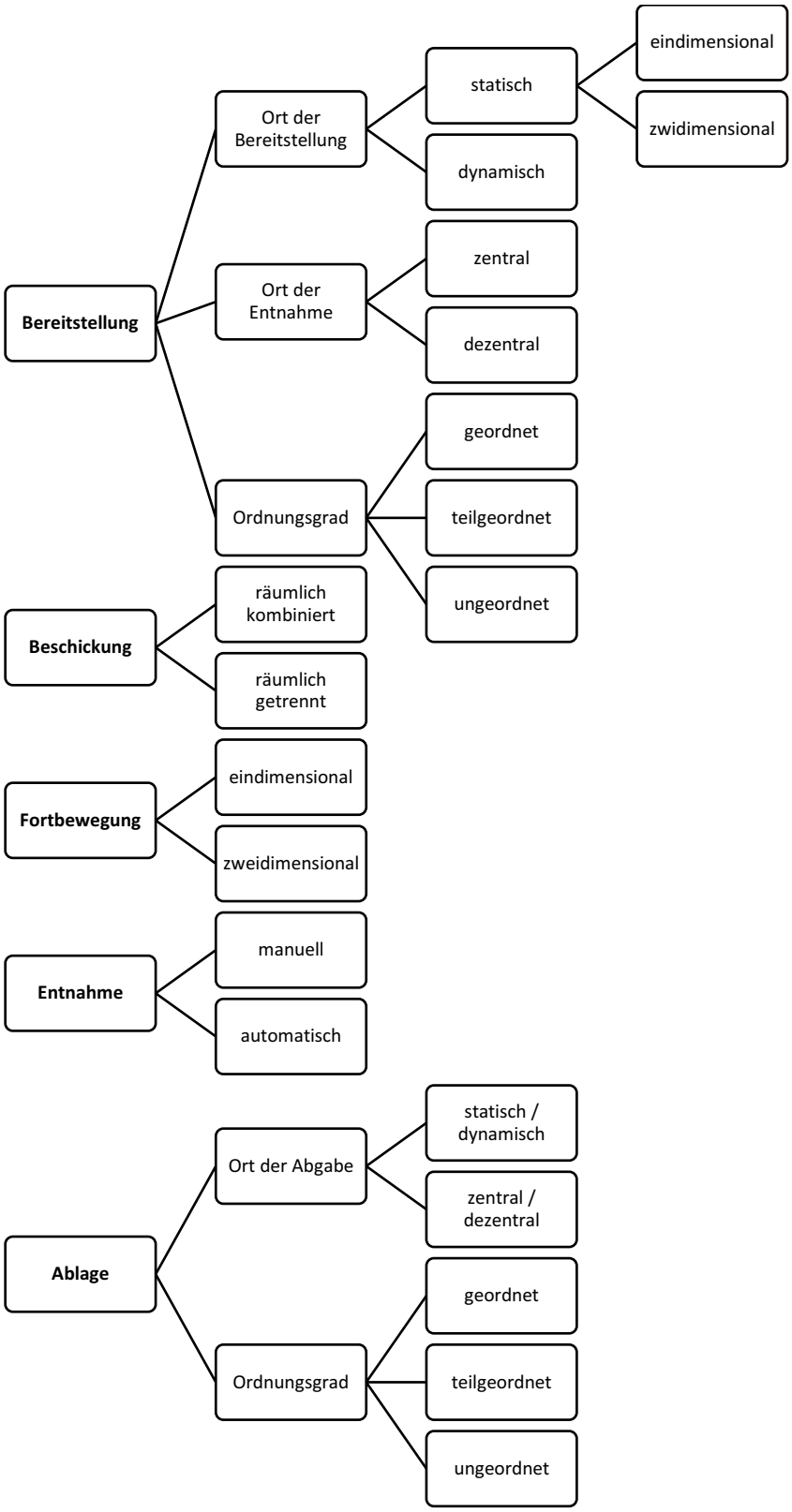
A-2 Verwendete Programme

Programm	Details
Microsoft Excel 2016	Version 1809 (Build 10827.20150) 64 Bit
MATLAB	Version R2018a (9.4.0.813654) 64-bit (win64)
Python	Anaconda 3 64 Bit Version 1.9.2 Version 3.6.5 64bits, Qt 5.9.4, PyQt5 5.9.2 on Windows In Spyder 3.3.1

A-3 Verwendete Python-Pakete

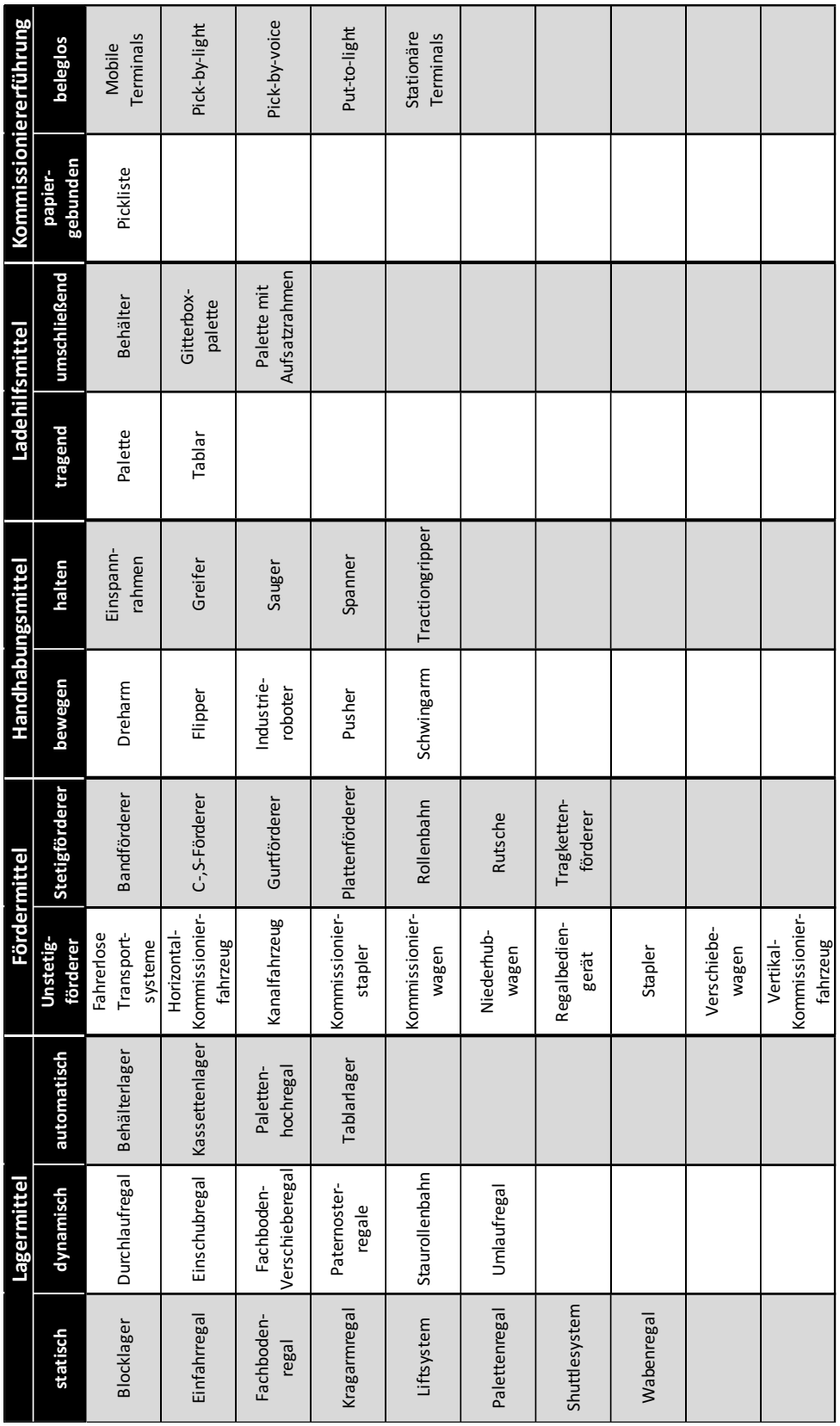
Paketname	Version	Funktion	Link zur Dokumentation
Standardpakete			
chain (itertools)	3.6	Handhabung von Daten	docs.python.org/3.6/library/itertools.html#itertools.chain
defaultdict (collections)	3.6	Handhabung von Daten	docs.python.org/3.6/library/collections.html#collections.defaultdict
math	3.6	Mathematische Funktionen	docs.python.org/3.6/library/math.html
random	3.6	Erzeugung von Zufallszahlen	docs.python.org/3.6/library/random.html
time	3.6	Umgang mit Zeiten	docs.python.org/3.6/library/time.html
NumPy	1.14.3	Datenverwaltung	numpy.org/
pandas	0.23.0	Datenverwaltung	pandas.pydata.org/
Visualisierung von Daten			
matplotlib	2.2.2	Visualisierung von Daten	matplotlib.org/
seaborn	0.8.1	Visualisierung von Daten	seaborn.pydata.org/
PDPbox	0.2.0	Erstellung PDP / ICE	pdpbox.readthedocs.io/en/latest/
Machine-Learning			
scikit-learn	0.19.1	Machine-Learning Tools	scikit-learn.org/stable/index.html
MinMaxScaler (sklearn.preprocessing)	0.19.1	Lineare Transformation	scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
train_test_split (sklearn.model_selection)	0.19.1	Split in Train-/Testdaten	scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
MLPRegressor (sklearn.neural_network)	0.19.1	MLP	scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html
mean_squared_error (sklearn.metrics)	0.19.1	Berechnung MSE	scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html
mean_absolute_error (sklearn.metrics)	0.19.1	Berechnung MAE	scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html
Genetischer Algorithmus			
DEAP	1.2.2	GA Framework	deap.readthedocs.io/en/master/

A-4 Kommissioniersystem Konfigurationen

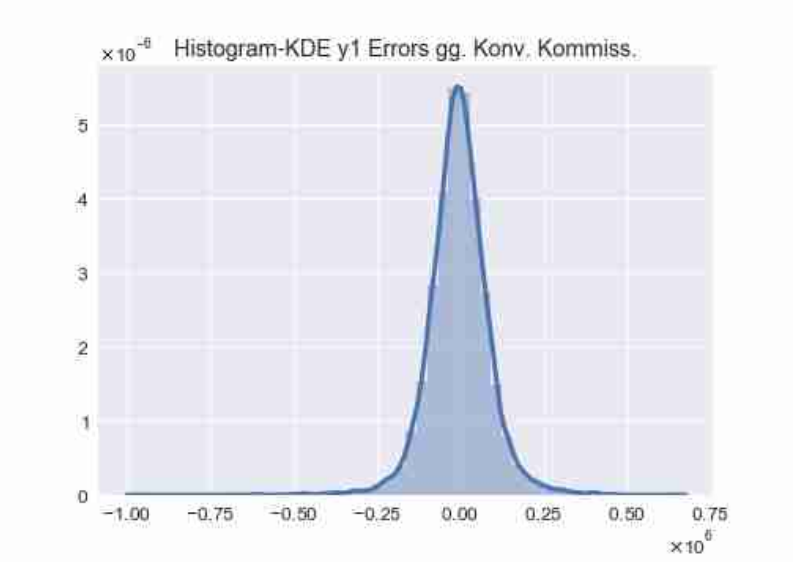


A-5 Technische Komponenten von Kommissioniersystemen

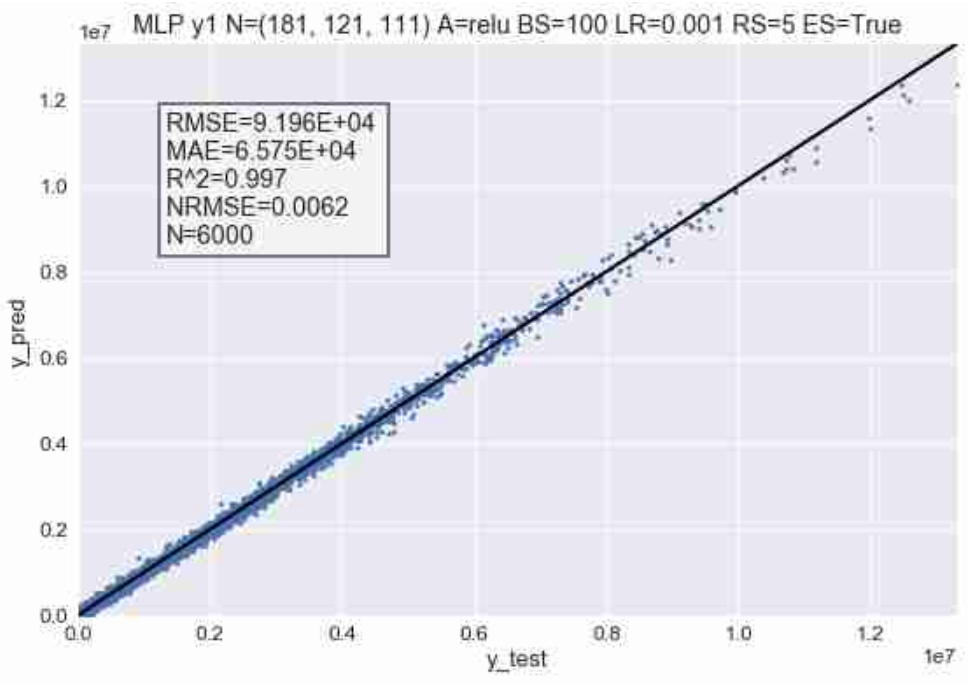
Nach Hompel et al. 2011, S. 44



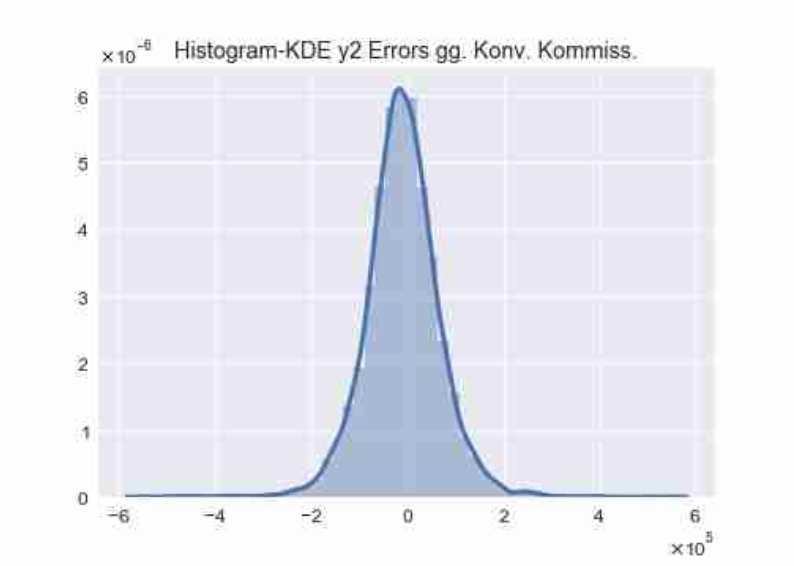
A-6 Modellauswahl Investitionskosten (gassengebunden)



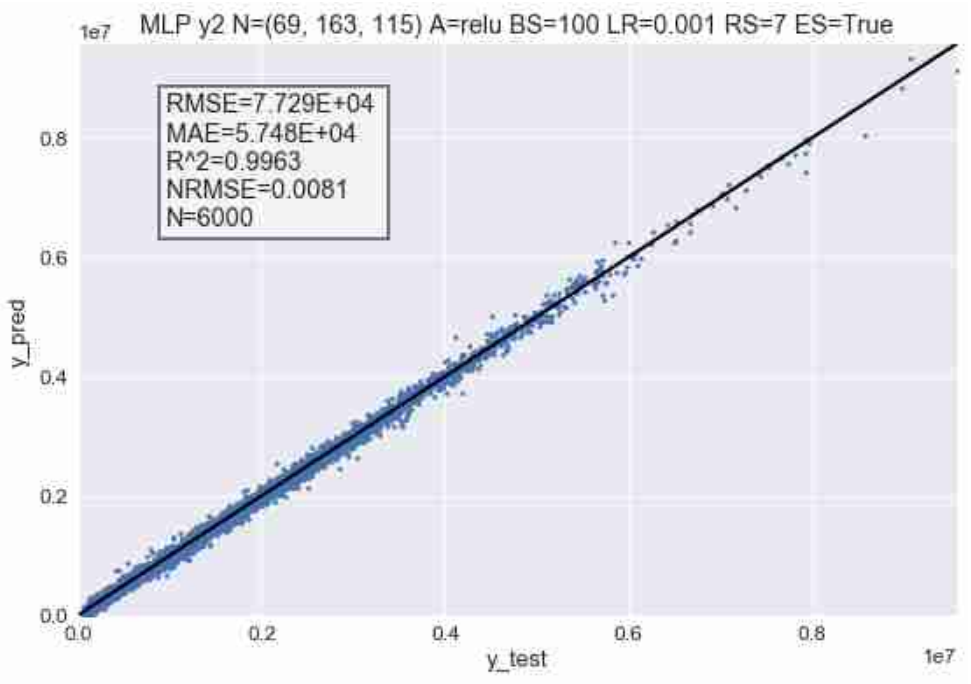
	R^2	RMSE	NRMSE	MAE	MSE	activation	batch_ size	early_ stopping	hidden_layer_ sizes	learning_ rate_init	random_ state
1	0,99691	8,835E+04	5,943E-03	6,345E+04	7,806E+09	relu	100	TRUE	(181, 121, 111)	0,001	5
2	0,99678	9,028E+04	6,073E-03	6,533E+04	8,150E+09	relu	400	TRUE	(86, 25, 112)	0,001	12
3	0,99674	9,084E+04	6,111E-03	6,254E+04	8,252E+09	relu	200	TRUE	(142, 13, 149)	0,001	8
4	0,99665	9,200E+04	6,189E-03	6,516E+04	8,465E+09	relu	100	TRUE	(81, 86, 64)	0,001	1
5	0,99652	9,375E+04	6,306E-03	6,734E+04	8,789E+09	relu	100	TRUE	(90, 141, 79)	0,001	11
6	0,99652	9,380E+04	6,310E-03	6,572E+04	8,799E+09	relu	200	TRUE	(189, 10, 182)	0,001	7
7	0,99632	9,646E+04	6,489E-03	7,056E+04	9,305E+09	relu	100	TRUE	(113, 147, 157)	0,0001	11
8	0,99612	9,899E+04	6,659E-03	7,196E+04	9,799E+09	relu	100	TRUE	(156, 194, 160)	0,001	14
9	0,99582	1,028E+05	6,912E-03	7,216E+04	1,056E+10	relu	100	TRUE	(19, 200, 138)	0,001	5
10	0,99582	1,028E+05	6,915E-03	7,427E+04	1,057E+10	relu	200	TRUE	(165, 154, 52)	0,001	2



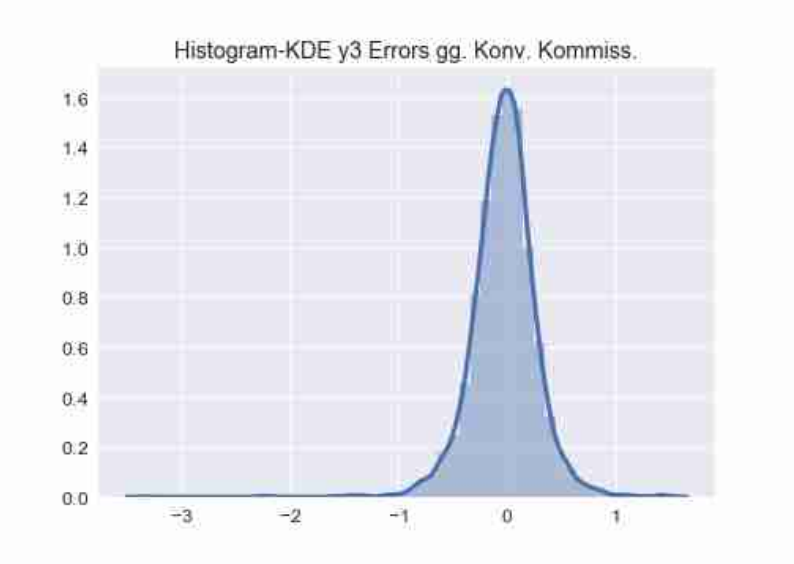
A-7 Modellauswahl Betriebskosten (gassengebunden)



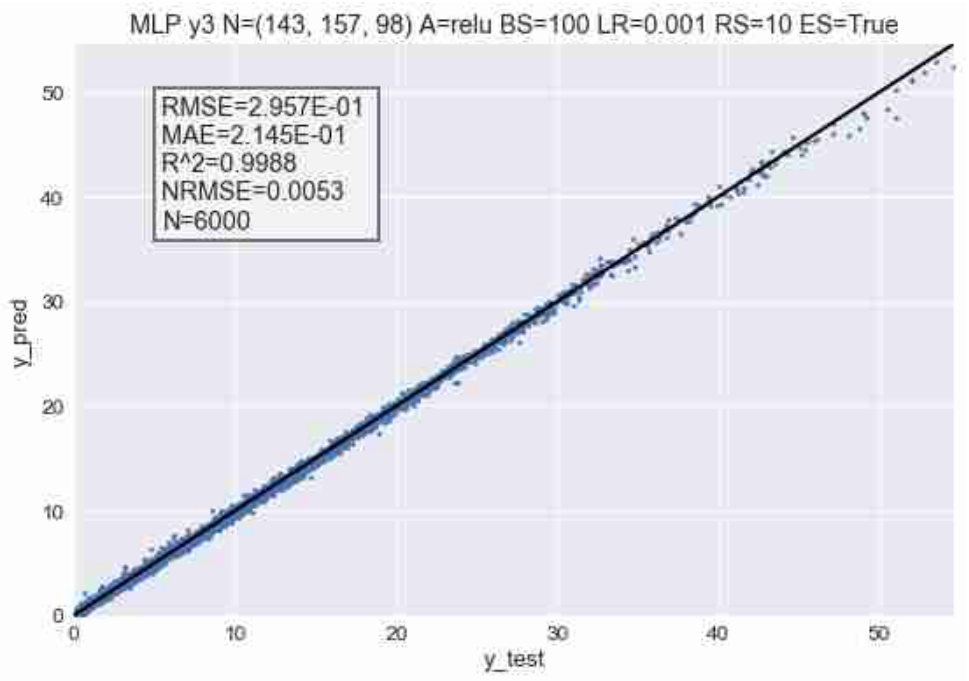
	R^2	RMSE	NRMSE	MAE	MSE	activation	batch_ size	early_ stopping	hidden_layer_ sizes	learning_ rate_init	random_ state
1	0,99656	7,382E+04	7,713E-03	5,535E+04	5,450E+09	relu	100	TRUE	(69, 163, 115)	0,001	7
2	0,99648	7,457E+04	7,792E-03	5,595E+04	5,561E+09	relu	100	TRUE	(166, 122, 137)	0,001	6
3	0,99617	7,787E+04	8,136E-03	5,805E+04	6,064E+09	relu	100	TRUE	(63, 101)	0,001	18
4	0,99617	7,789E+04	8,138E-03	5,805E+04	6,066E+09	relu	100	TRUE	(180, 139, 58)	0,01	9
5	0,99612	7,839E+04	8,191E-03	5,877E+04	6,146E+09	relu	200	TRUE	(169, 133, 137)	0,001	5
6	0,99543	8,506E+04	8,887E-03	6,533E+04	7,235E+09	relu	100	TRUE	(157, 122, 55)	0,001	11
7	0,99540	8,530E+04	8,912E-03	6,337E+04	7,276E+09	relu	100	TRUE	(88, 117, 17)	0,001	2
8	0,99509	8,810E+04	9,205E-03	6,468E+04	7,761E+09	relu	100	TRUE	(158, 20, 44)	0,001	7
9	0,99497	8,917E+04	9,316E-03	6,681E+04	7,951E+09	relu	100	TRUE	(133, 129)	0,01	8
10	0,99495	8,936E+04	9,337E-03	6,485E+04	7,985E+09	relu	100	TRUE	(145, 83)	0,01	16



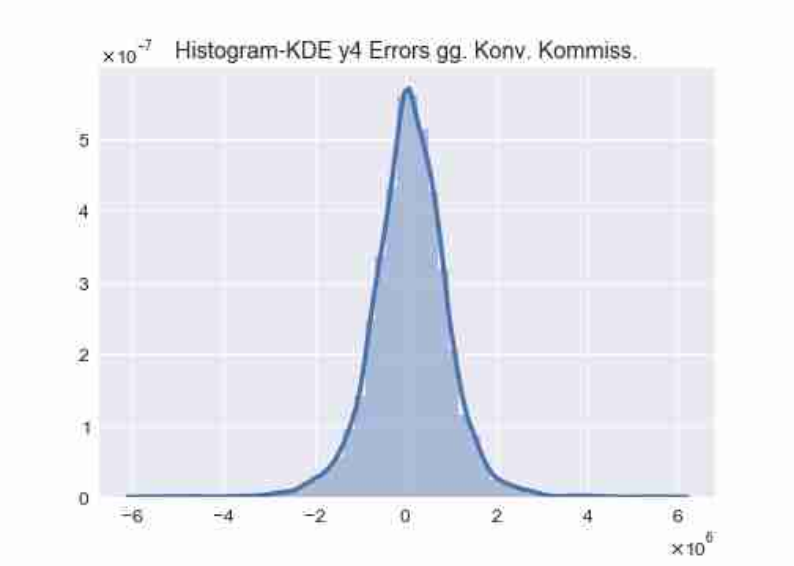
A-8 Modellauswahl Anzahl Kommissionierer (gassengebunden)



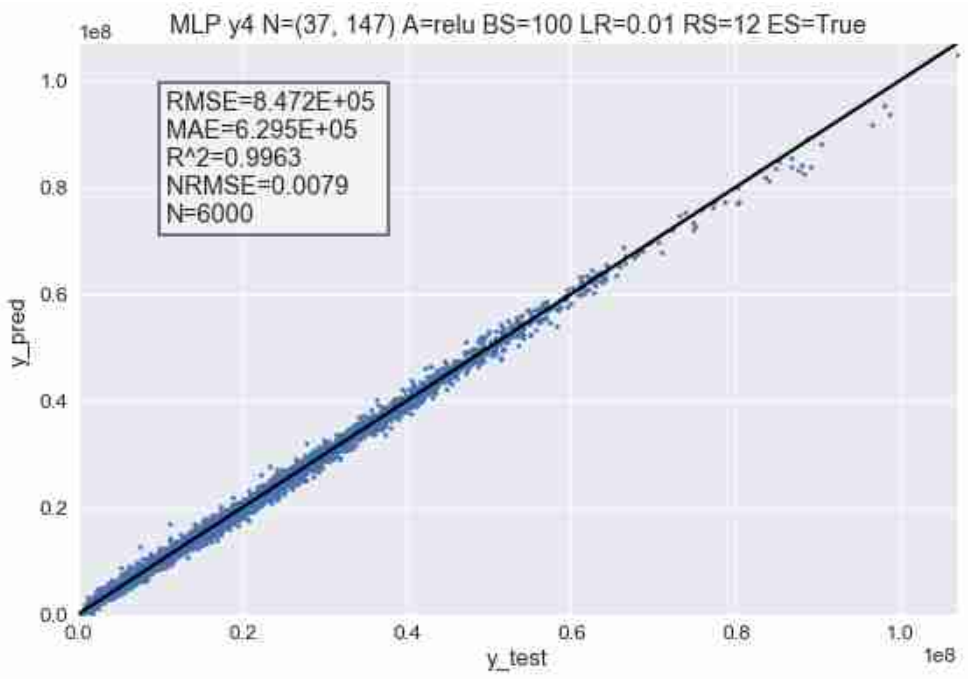
	R^2	RMSE	NRMSE	MAE	MSE	activation	batch_ size	early_ stopping	hidden_layer_ sizes	learning_ rate_init	random_ state
1	0,99892	2,916E-01	5,190E-03	2,127E-01	8,504E-02	relu	100	TRUE	(143, 157, 98)	0,001	10
2	0,99888	2,976E-01	5,296E-03	2,234E-01	8,856E-02	tanh	200	TRUE	(195, 76)	0,01	4
3	0,99884	3,018E-01	5,371E-03	2,341E-01	9,108E-02	tanh	100	TRUE	(195, 139, 20)	0,001	8
4	0,99884	3,031E-01	5,394E-03	2,271E-01	9,184E-02	relu	100	TRUE	(49, 149, 146)	0,01	6
5	0,99883	3,034E-01	5,400E-03	2,239E-01	9,206E-02	tanh	100	TRUE	(152, 21, 66)	0,001	5
6	0,99881	3,057E-01	5,441E-03	2,256E-01	9,345E-02	relu	100	TRUE	(59, 142)	0,01	5
7	0,99877	3,120E-01	5,553E-03	2,354E-01	9,734E-02	relu	100	TRUE	(65, 181, 66)	0,001	12
8	0,99876	3,127E-01	5,566E-03	2,361E-01	9,781E-02	tanh	100	TRUE	(199, 20, 104)	0,001	8
9	0,99876	3,133E-01	5,576E-03	2,295E-01	9,815E-02	relu	100	TRUE	(191, 46, 146)	0,001	9
10	0,99875	3,135E-01	5,579E-03	2,356E-01	9,827E-02	relu	100	TRUE	(115, 177, 144)	0,001	4



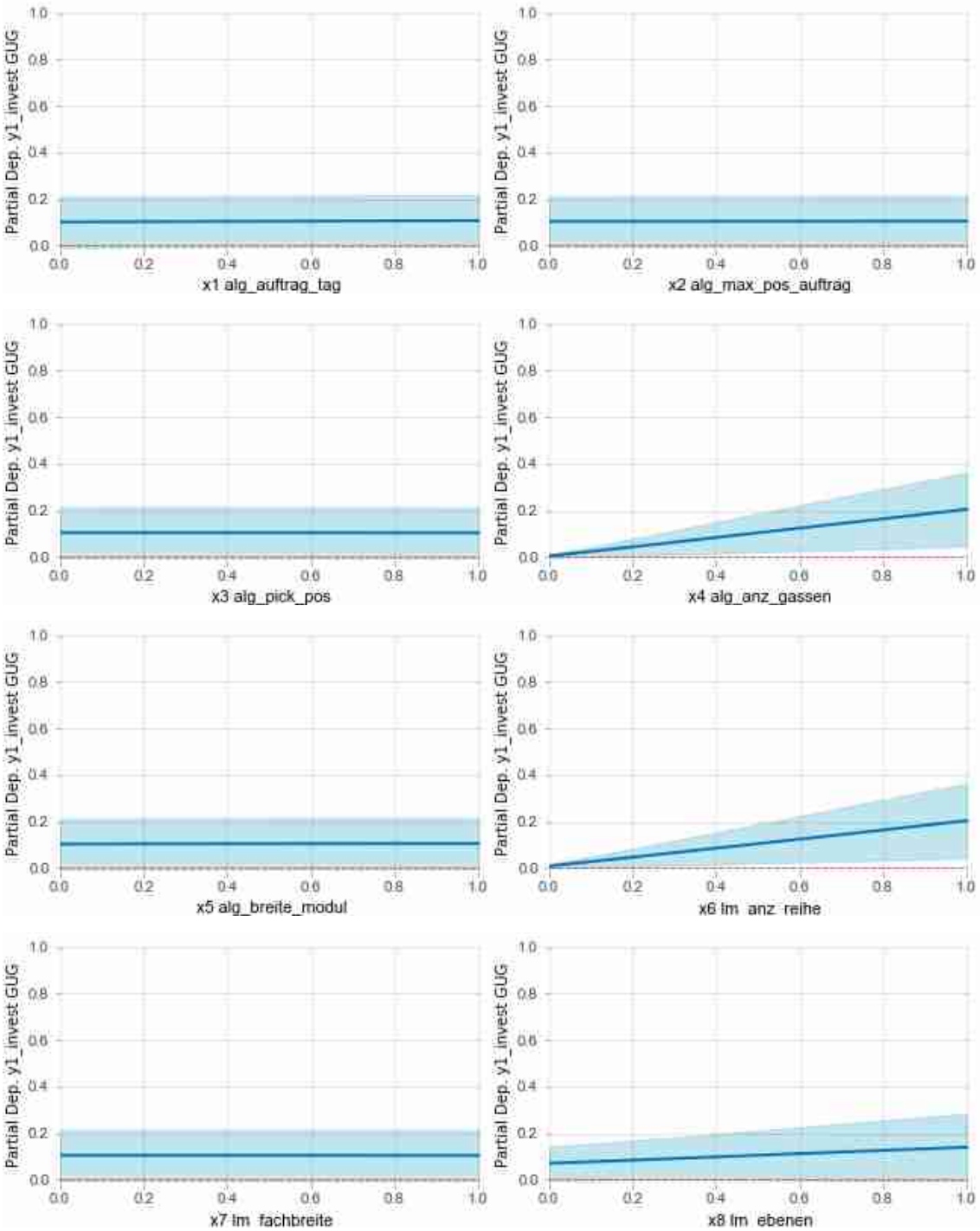
A-9 Modellauswahl TCO (gassengebunden)

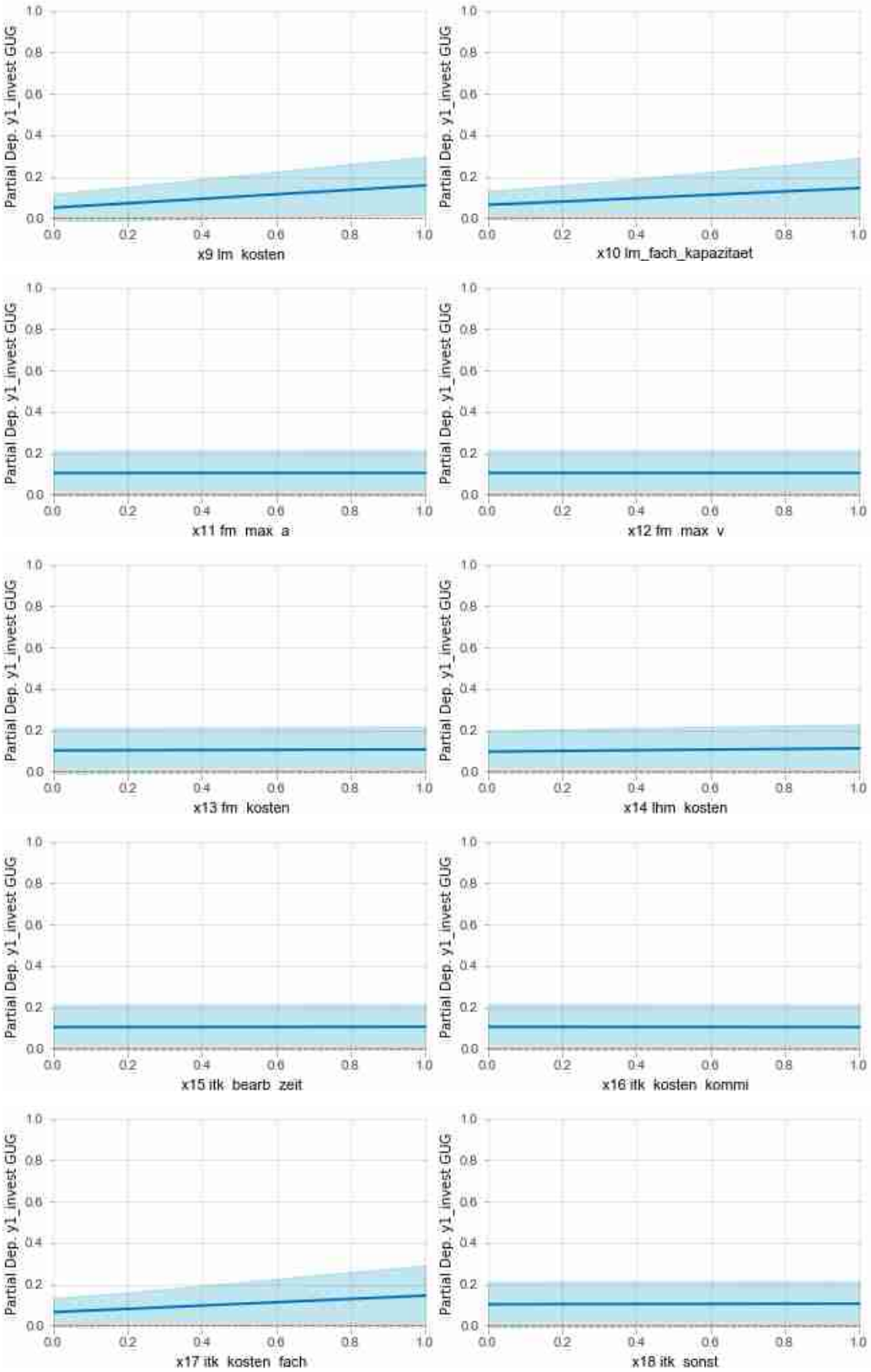


	R^2	RMSE	NRMSE	MAE	MSE	activation	batch_ size	early_ stopping	hidden_layer_ sizes	learning_ rate_init	random_ state
1	0,99627	8,386E+05	7,847E-03	6,288E+05	7,033E+11	relu	100	TRUE	(37, 147)	0,01	12
2	0,99576	8,948E+05	8,373E-03	6,710E+05	8,007E+11	relu	100	TRUE	(182, 131, 113)	0,001	17
3	0,99567	9,040E+05	8,459E-03	6,710E+05	8,172E+11	relu	100	TRUE	(114, 94, 32)	0,01	19
4	0,99559	9,125E+05	8,539E-03	6,802E+05	8,326E+11	relu	100	TRUE	(83, 158, 149)	0,001	7
5	0,99547	9,239E+05	8,645E-03	6,955E+05	8,536E+11	relu	100	TRUE	(103, 178)	0,001	8
6	0,99535	9,369E+05	8,767E-03	6,902E+05	8,778E+11	relu	100	TRUE	(112, 136)	0,001	17
7	0,99532	9,399E+05	8,796E-03	7,084E+05	8,835E+11	relu	100	TRUE	(185, 84, 21)	0,001	9
8	0,99531	9,404E+05	8,800E-03	6,959E+05	8,844E+11	relu	100	TRUE	(152, 112, 17)	0,001	8
9	0,99524	9,474E+05	8,865E-03	6,974E+05	8,975E+11	relu	100	TRUE	(112, 83)	0,01	11
10	0,99519	9,526E+05	8,914E-03	7,207E+05	9,075E+11	relu	200	TRUE	(197, 63, 104)	0,001	16

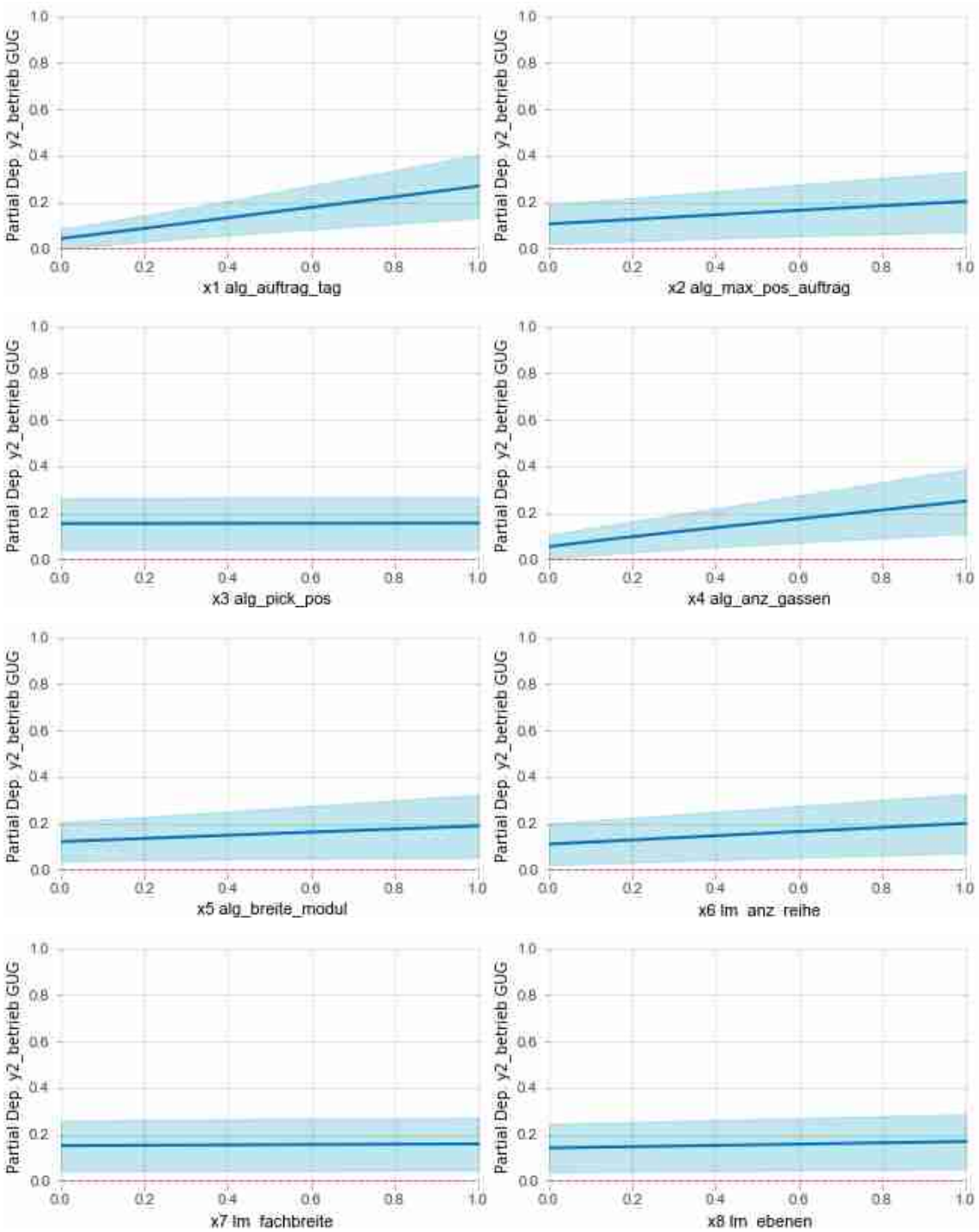


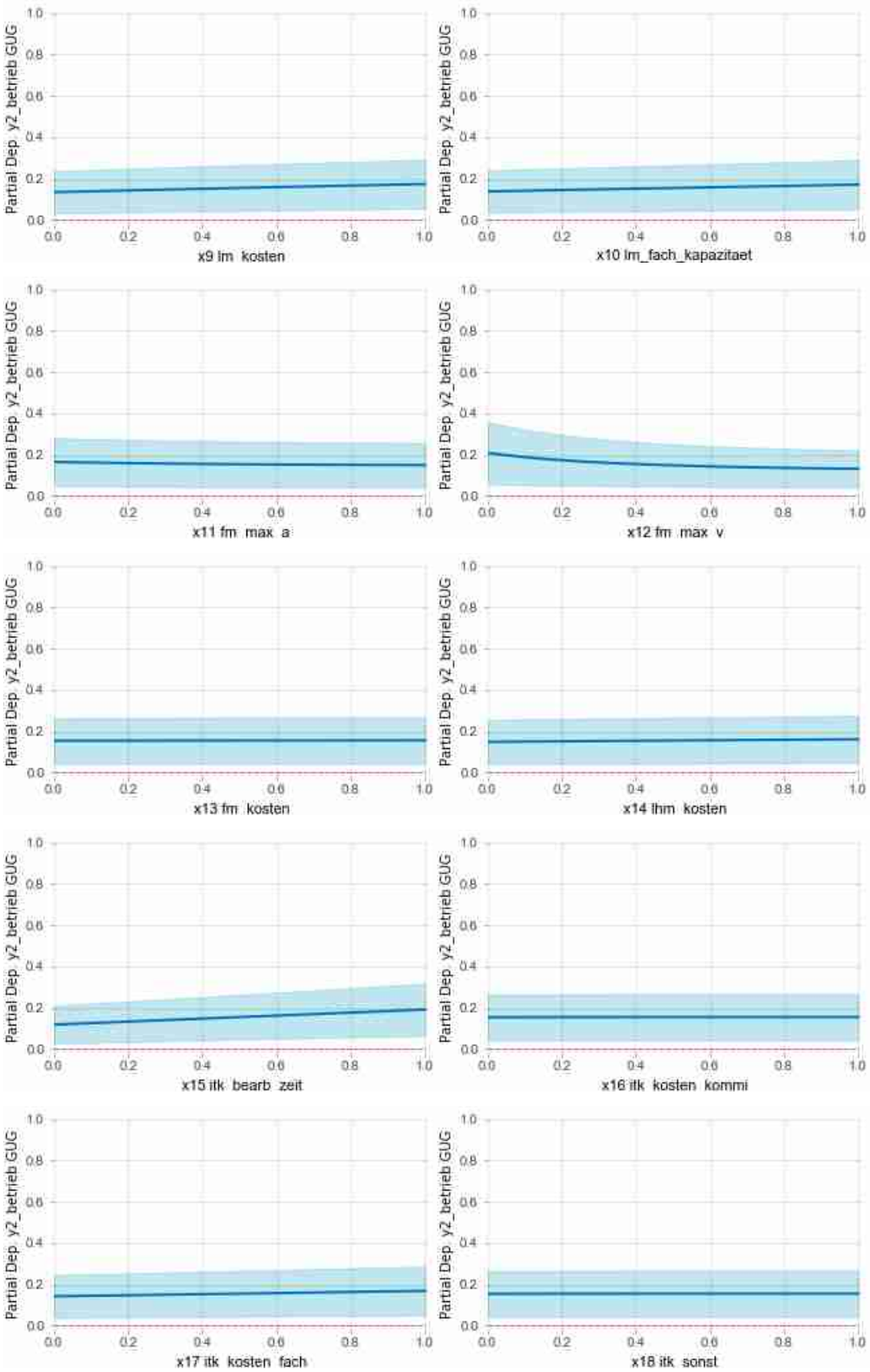
A-10 PDPs Investitionskosten (gassenungebunden)



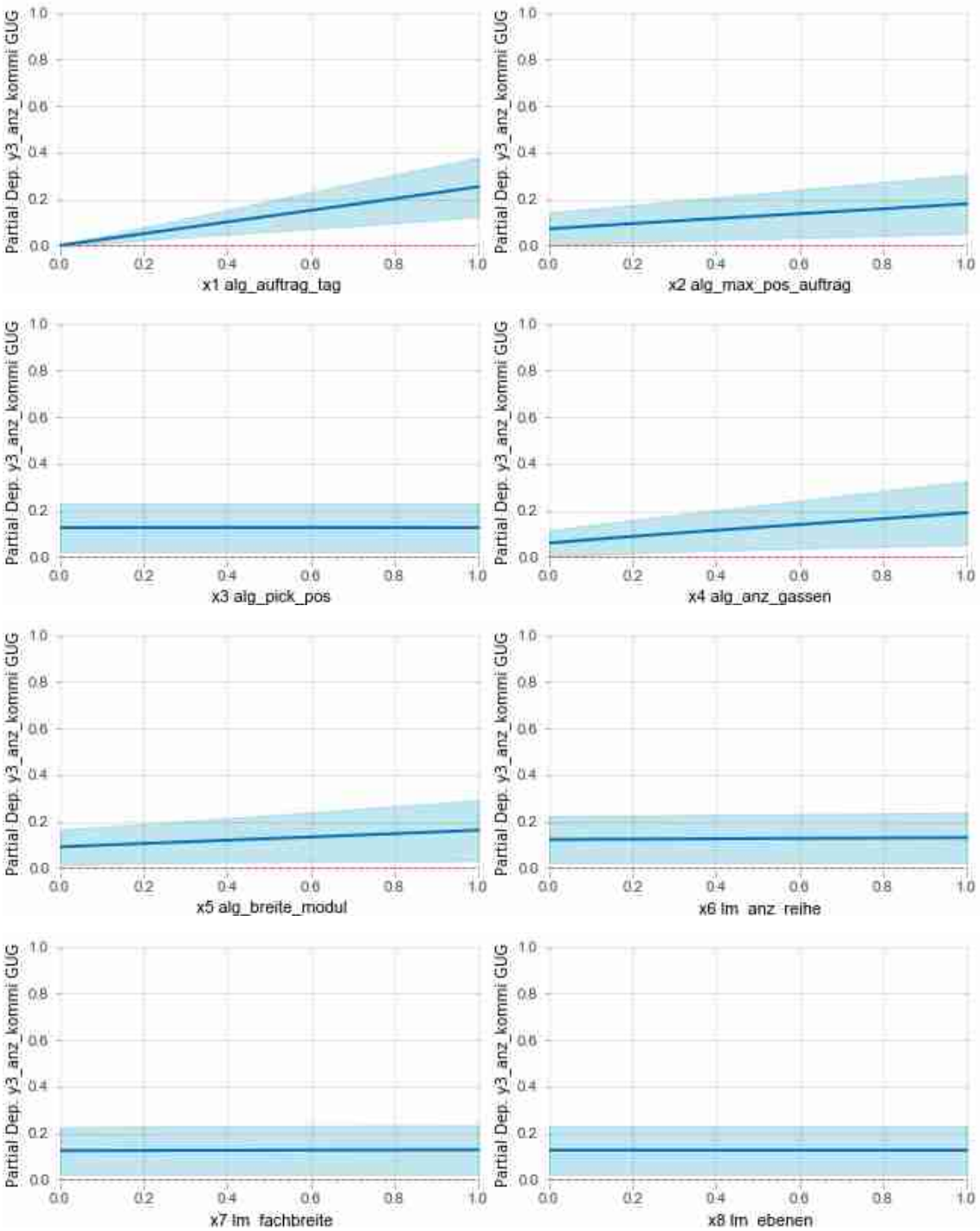

Partial Dep. y1_invest GUG
x9 lm_kosten
x10 lm_fach_kapazitaet
x11 fm_max_a
x12 fm_max_v
x13 fm_kosten
x14 lhm_kosten
x15 itk_bearb_zeit
x16 itk_kosten_kommi
x17 itk_kosten_fach
x18 itk_sonst

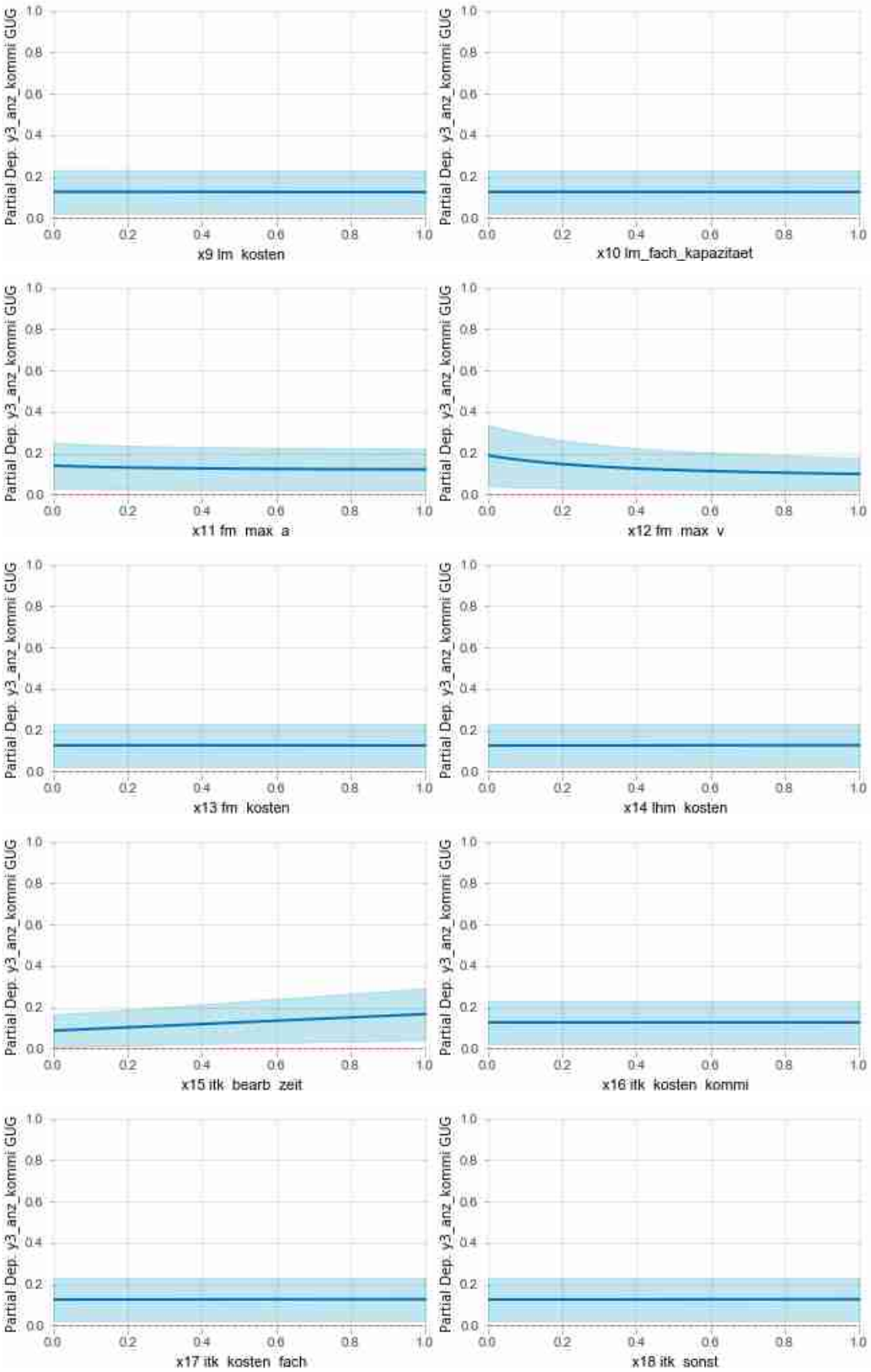
A-11 PDPs Betriebskosten (gassenungebunden)



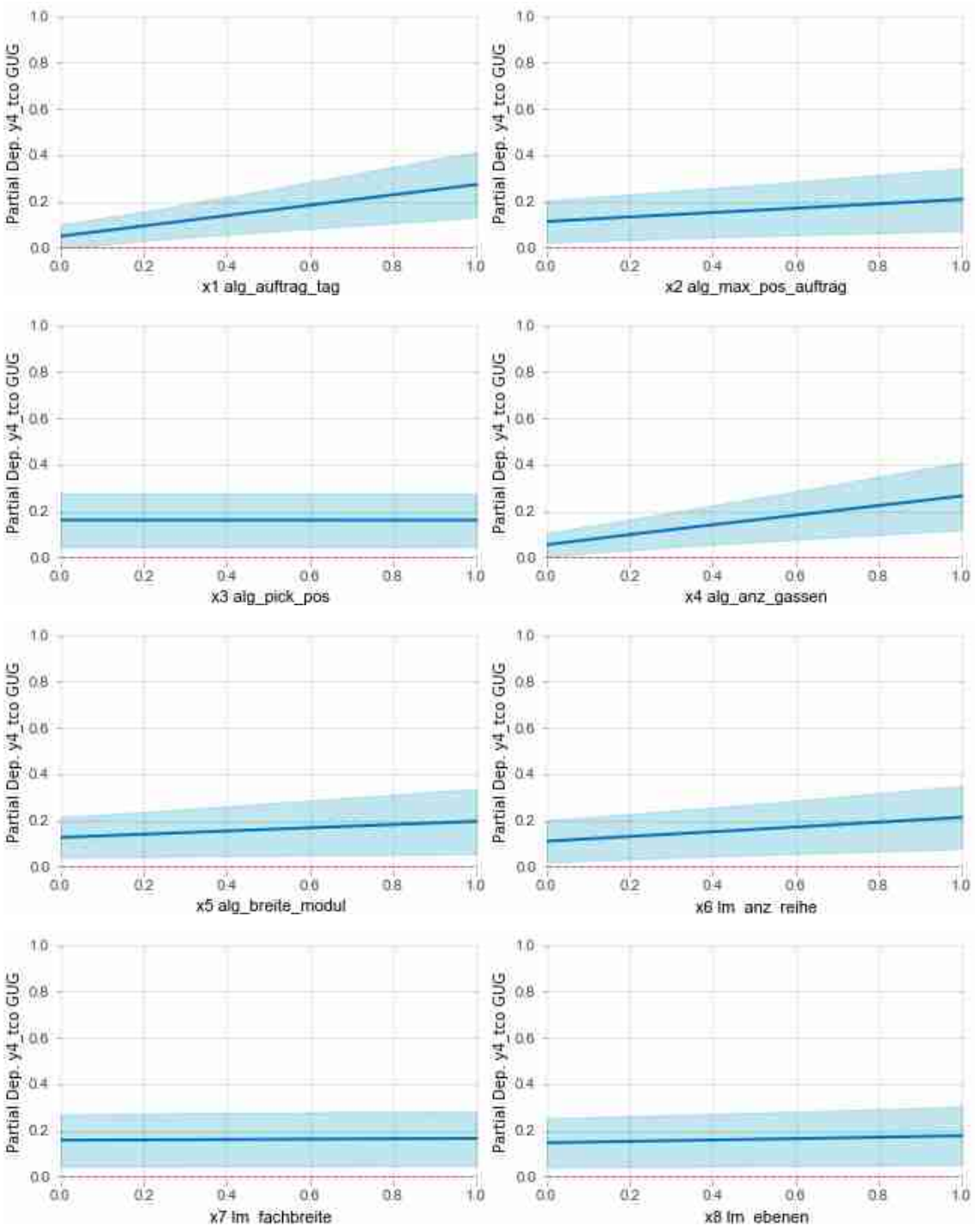

Partial Dep_y2_betrieb GUG
x9 lm_kosten
x10 lm_fach_kapazitaet
x11 fm_max_a
x12 fm_max_v
x13 fm_kosten
x14 lhm_kosten
x15 itk_bearb_zeit
x16 itk_kosten_kommi
x17 itk_kosten_fach
x18 itk_sonst

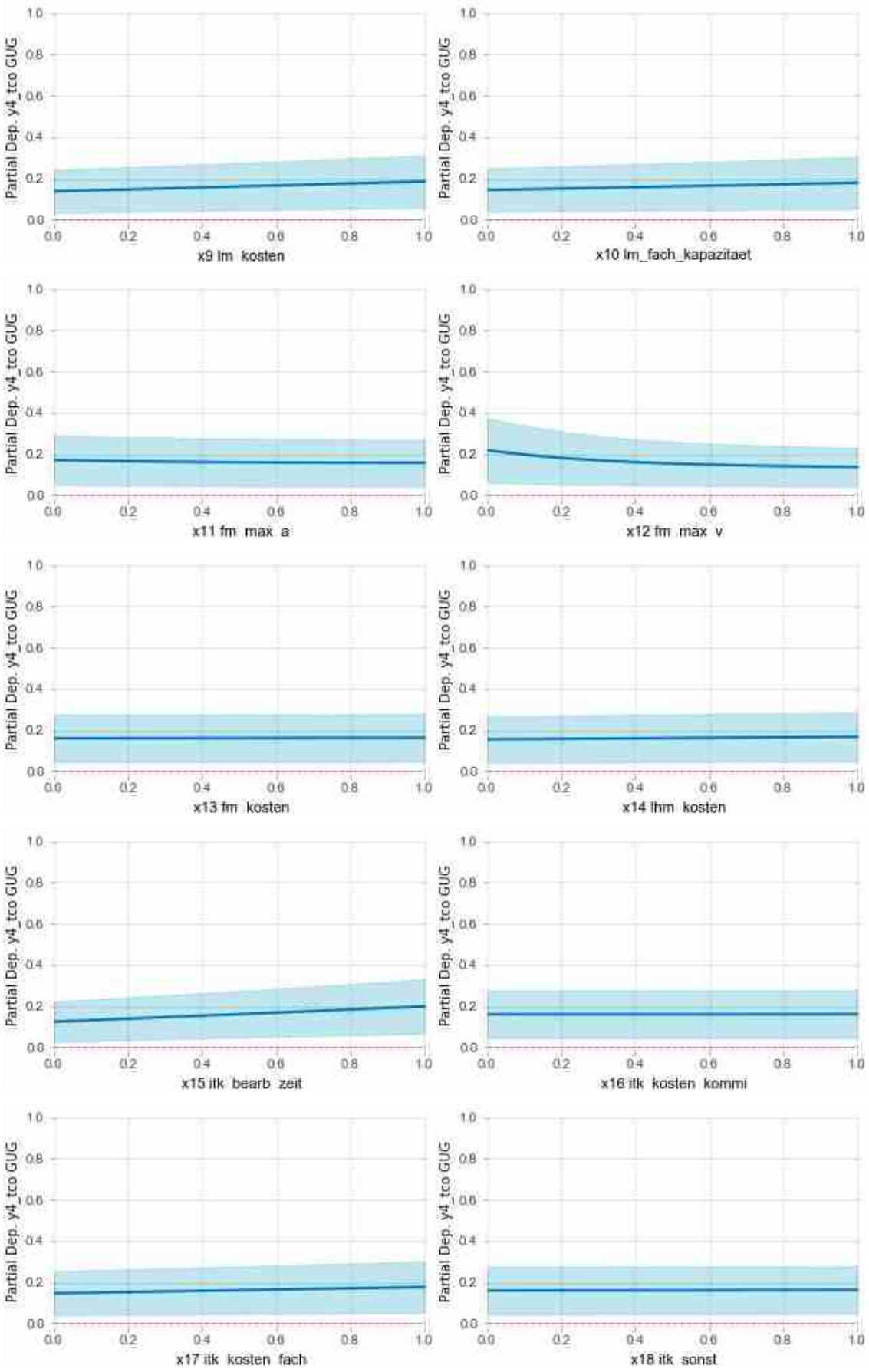
A-12 PDP Anzahl Kommissionierer (gassenungebunden)



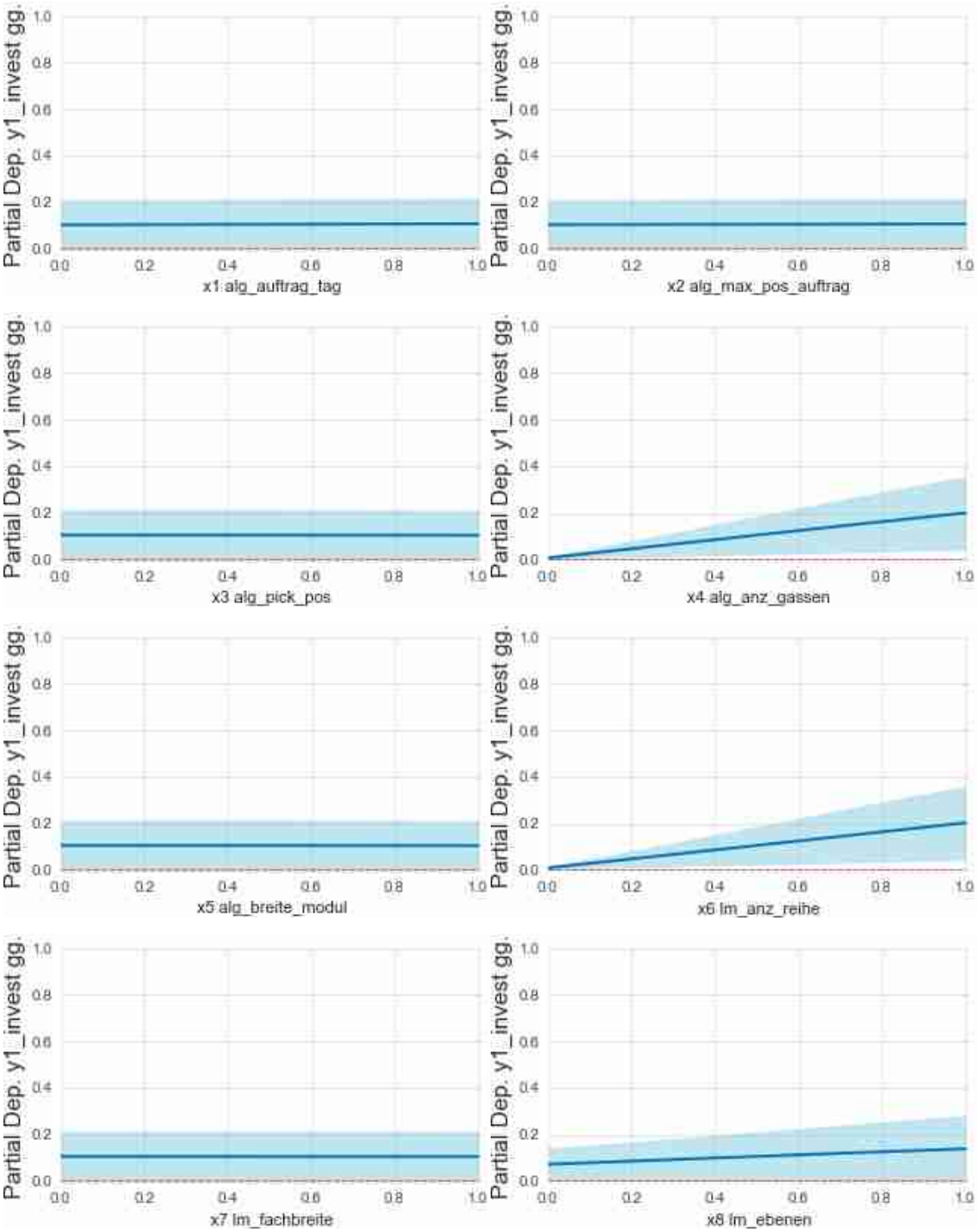

Partial Dep. y3_anz_kommi GUG
x9 lm_kosten
x10 lm_fach_kapazitaet
x11 fm_max_a
x12 fm_max_v
x13 fm_kosten
x14 lhm_kosten
x15 itk_bearb_zeit
x16 itk_kosten_kommi
x17 itk_kosten_fach
x18 itk_sonst

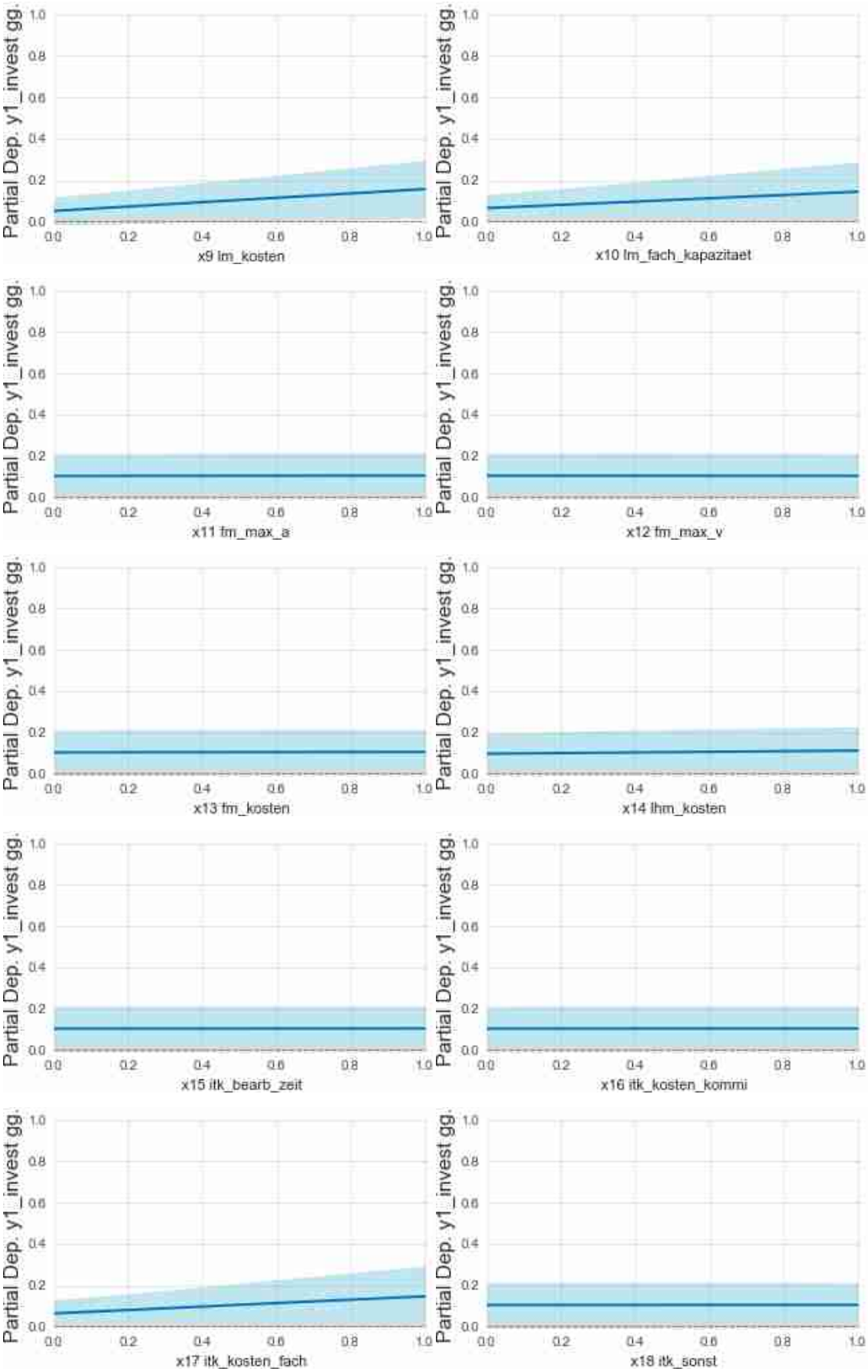
A-13 PDPs TCO (gassenungebunden)



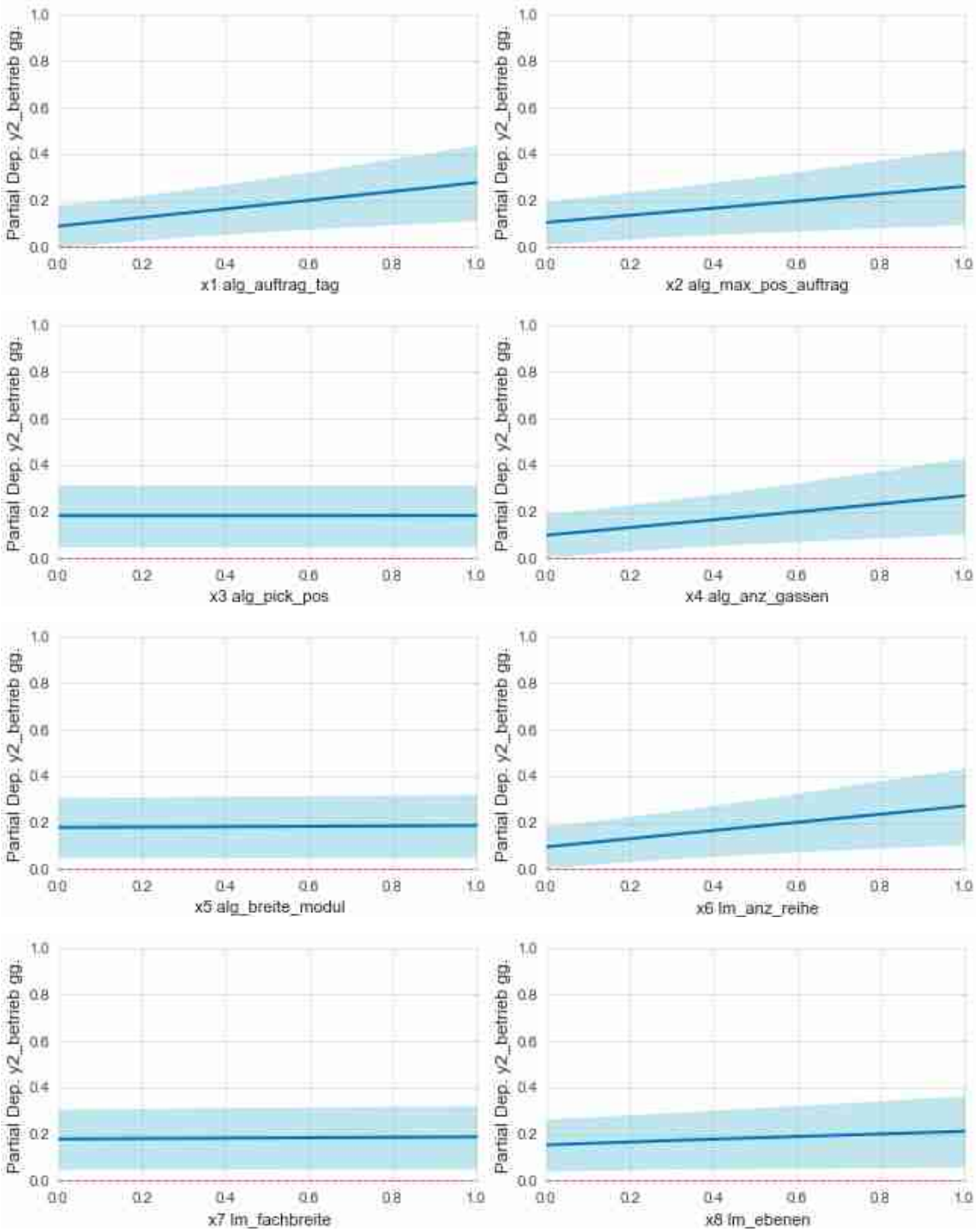

Partial Dep. y4_tco GUG
x9 lm_kosten
x10 lm_fach_kapazitaet
x11 fm_max_a
x12 fm_max_v
x13 fm_kosten
x14 lhm_kosten
x15 itk_bearb_zeit
x16 itk_kosten_kommi
x17 itk_kosten_fach
x18 itk_sonst

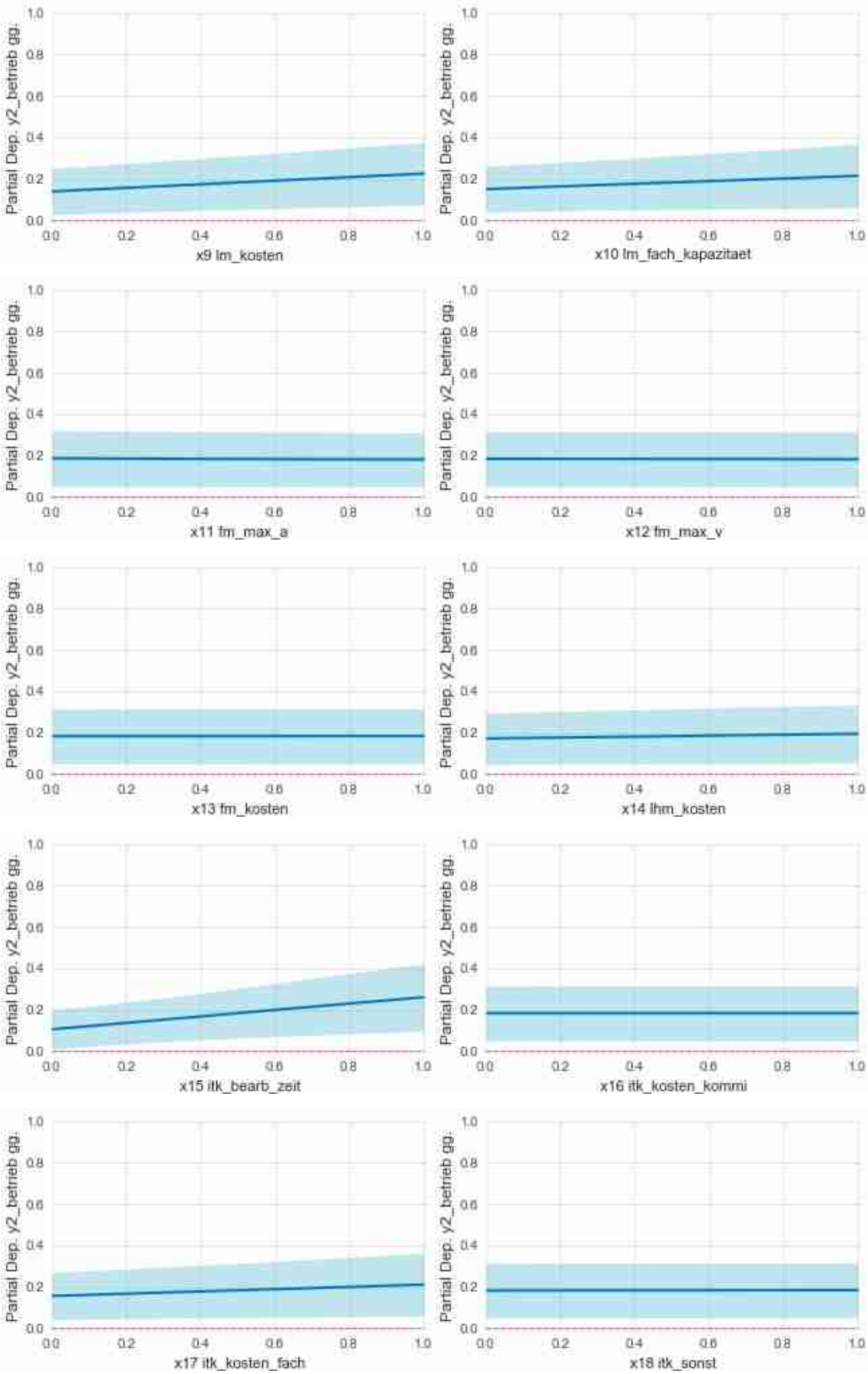
A-14 PDPs Investitionskosten (gassengebunden)



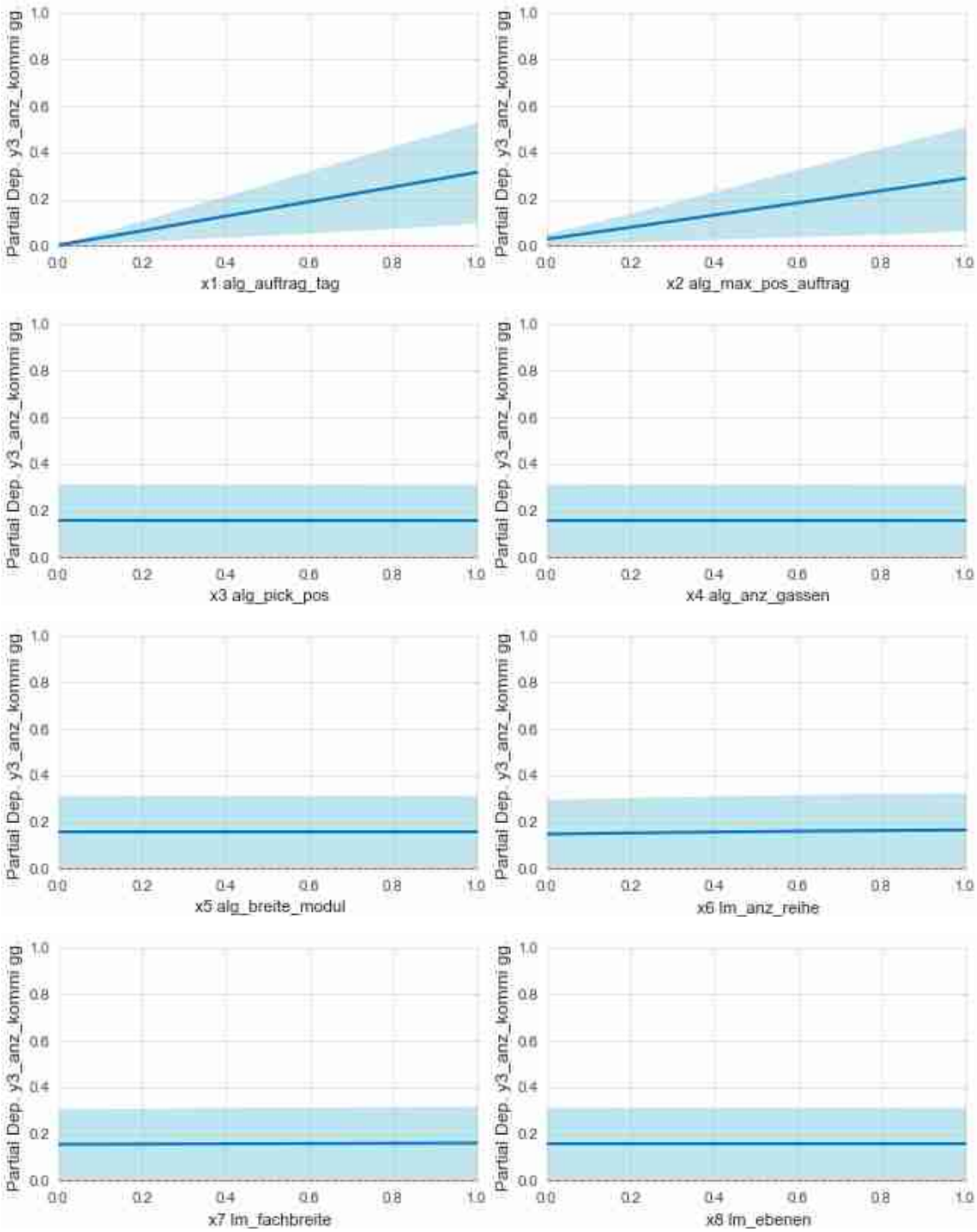

Partial Dep. y1_invest gg.
x9 lm_kosten
x10 lm_fach_kapazitaet
x11 fm_max_a
x12 fm_max_v
x13 fm_kosten
x14 lhm_kosten
x15 itk_bearb_zeit
x16 itk_kosten_kommi
x17 itk_kosten_fach
x18 itk_sonst

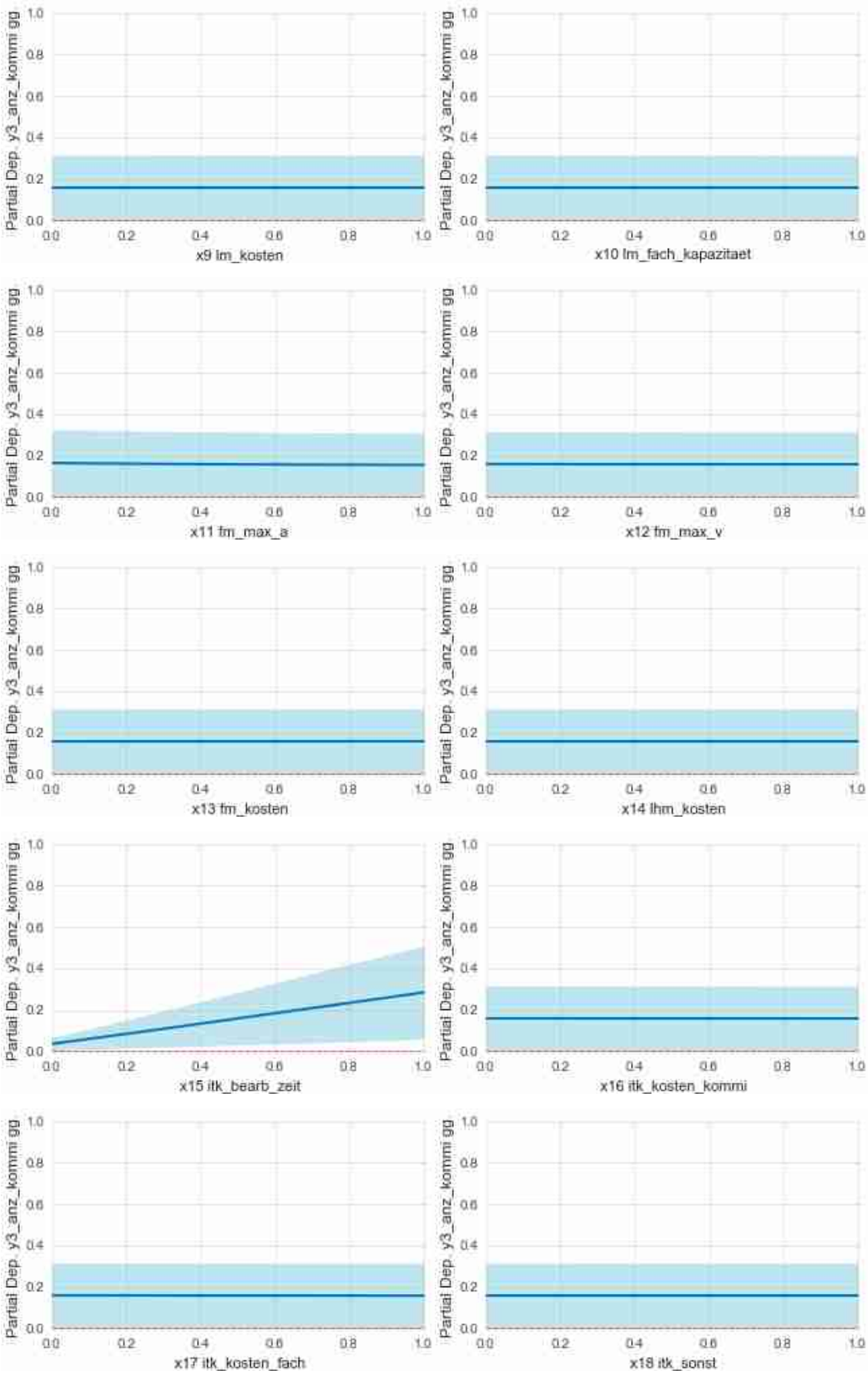
A-15 PDPs Betriebskosten (gassengebunden)



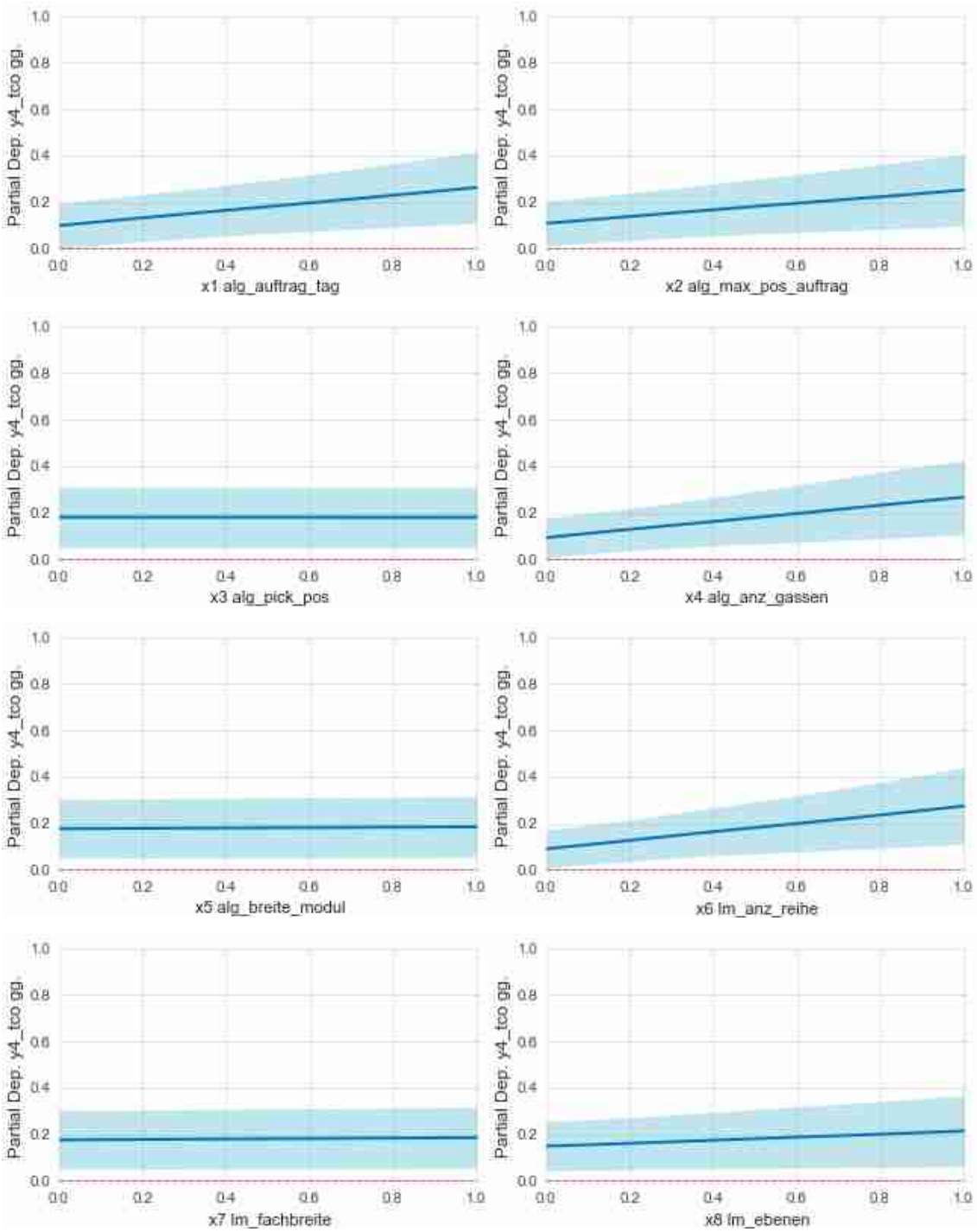

Partial Dep. y2_betrieb gg:
x9 lm_kosten
x10 lm_fach_kapazitaet
x11 fm_max_a
x12 fm_max_v
x13 fm_kosten
x14 lhm_kosten
x15 itk_bearb_zeit
x16 itk_kosten_kommi
x17 itk_kosten_fach
x18 itk_sonst

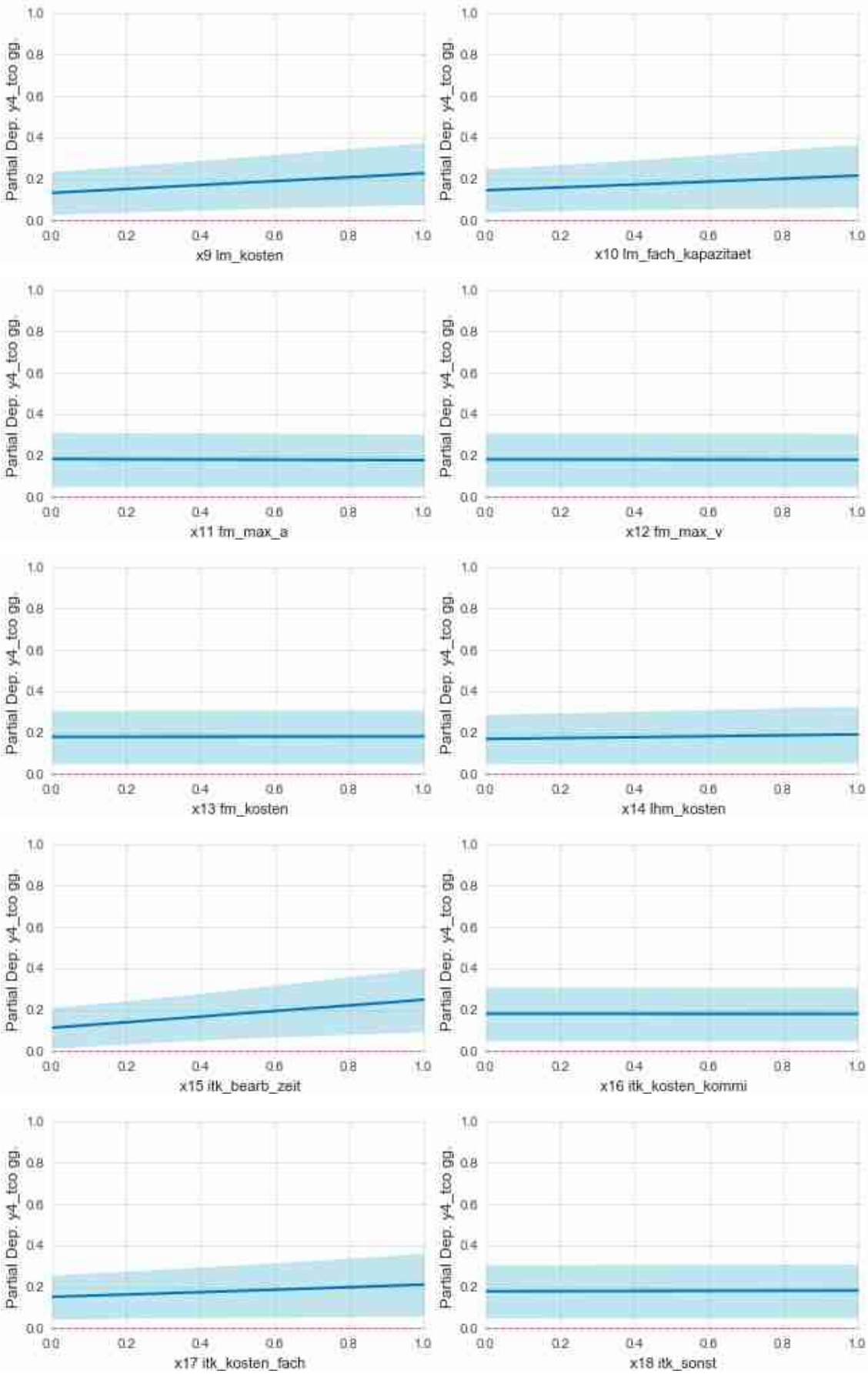
A-16 PDPs Anzahl Kommissionierer (gassengebunden)




Partial Dep. y3_anz_kommi gg
x9 lm_kosten
x10 lm_fach_kapazitaet
x11 fm_max_a
x12 fm_max_v
x13 fm_kosten
x14 lhm_kosten
x15 itk_bearb_zeit
x16 itk_kosten_kommi
x17 itk_kosten_fach
x18 itk_sonst

A-17 PDPs TCO (gassengebunden)




Partial Dep. y4_tco gg
x9 lm_kosten
x10 lm_fach_kapazitaet
x11 fm_max_a
x12 fm_max_v
x13 fm_kosten
x14 lhm_kosten
x15 itk_bearb_zeit
x16 itk_kosten_kommi
x17 itk_kosten_fach
x18 itk_sonst

A-18 Spannweiten der Komponenten-Phänotypen

Kategorie	Lagermittel (LM)				Ladehilfsmittel (LHM)				ITK		
Bezeichnung	Fachbreite [mm]	Fachtiefe [mm]	Ebenen	Kosten [€]	Breite des LHM [mm]	Tiefe des LHM [mm]	Kosten [€]	Bearbeitungszeit [s]	Kosten pro Kommissionierer [€]	Kosten pro Lagerfach [€]	Sonstige Kosten [€]
Variable	lm_fachbreite x7	lm_fachtiefe	lm_ebenen x8	lm_kosten x9	lhm_breite	lhm_tiefe	lhm_kosten x14	itk_bearb_zeit x15	itk_kosten_kommi x16	itk_kosten_fach x17	itk_sonst x18
Min	800	300	1	0	200	300	0	3	0	0	0
Max	2000	1500	6	2500	800	1200	20	40	200	100	50000

Kategorie	Fördermittel (FM)							
Bezeichnung	Breite [mm]	Beschleunigung [m/s^2]	Geschwindig-keit [m/s]	Kapazität für LHM1 [Stück]	Kapazität für LHM2 [Stück]	Kapazität für LHM3 [Stück]	Kapazität für LHM4 [Stück]	Kapazität für LHM5 [Stück]	Kosten [€]
Variable	fm_breite	fm_max_a x11	fm_max_v x12	fm_kapazitaet_1	fm_kapazitaet_2	fm_kapazitaet_3	fm_kapazitaet_4	fm_kapazitaet_5	fm_kosten x13
Min	1200	0,25	0,5	0	0	0	0	0	0
Max	3500	1,5	1,7	7	7	7	7	7	4000

A-19 Optimale Konfiguration sehr kleines Kommissioniersystem

Konfiguration		
[7, 27, 19, 20, 4, 4]		
Leistungsdaten		
Aufträge pro Tag	1.500	
Max Positionen pro Auftrag	5	
Picks pro Position	10	
Lagerdaten		
Anzahl Gassen	7	
Breite eines Moduls	2,19	m
Lagerplätze	37.800	
Lagermittel		
LM-Nummer	19	
Lagermittel in Reihe	27	
Fachbreite	1.785	mm
Fachtiefe	1.092	mm
Ebenen	5	
Kosten	168,00	€
Fachkapzität	10	LHM
Fördermittel		
FM Nummer	20	
Max. Beschleunigung	1,15	m/s²
Max. Geschwindigkeit	1,10	m/s
Breite	3,1	m
Kapazität	4	LHM
Kosten	1.020,00	€
Ladehilfsmittel		
LHM-Nummer	4	
Breite	223	mm
Tiefe	407	mm
Kosten	3.200,00	€
Informationstechnische Kommissioniererführung		
ITK-Nummer	4	
Bearbeitungszeit	22	s/pos
Kosten pro Kommissionierer	160,00	€
Kosten pro Fach	3,00	€
Sonstige Kosten	36.090,00	€
TCO		
	6.533.278,34	**€**

A-20 Optimale Konfiguration kleines Kommissioniersystem

Konfiguration		
[7, 30, 24, 25, 4, 4]		
Leistungsdaten		
Aufträge pro Tag	1.500	
Max Positionen pro Auftrag	5	
Picks pro Position	10	
Lagerdaten		
Anzahl Gassen	7	
Breite eines Moduls	1,92	m
Lagerplätze	50.400	LHM
Lagermittel		
LM-Nummer	24	
Lagermittel in Reihe	30	
Fachbreite	1.718	mm
Fachtiefe	959	mm
Ebenen	6	
Kosten	702,00	€
Fachkapzität	10	LHM
Fördermittel		
FM Nummer	25	
Max. Beschleunigung	0,75	m/s^2
Max. Geschwindigkeit	1,15	m/s
Breite	3,2	m
Kapazität	5	LHM
Kosten	3.200,00	€
Ladehilfsmittel		
LHM-Nummer	4	
Breite	223	mm
Tiefe	407	mm
Kosten	3.200,00	€
Informationstechnische Kommissioniererführung		
ITK-Nummer	4	
Bearbeitungszeit	22	s/pos
Kosten pro Kommissionierer	160,00	€
Kosten pro Fach	3,00	€
Sonstige Kosten	36.090,00	€
TCO		
	9.770.342,39	**€**

A-21 Optimale Konfiguration mittleres Kommissioniersystem

Konfiguration		
[14, 30, 15, 7, 4, 4]		
Leistungsdaten		
Aufträge pro Tag	4.000	
Max Positionen pro Auftrag	5	
Picks pro Position	10	
Lagerdaten		
Anzahl Gassen	14	
Breite eines Moduls	1,77	m
Lagerplätze	100.800	
Lagermittel		
LM-Nummer	15	
Lagermittel in Reihe	30	
Fachbreite	1.191	mm
Fachtiefe	882	mm
Ebenen	6	
Kosten	614,00	€
Fachkapzität	10	LHM
Fördermittel		
FM Nummer	7	
Max. Beschleunigung	1,05	m/s²
Max. Geschwindigkeit	1,35	m/s
Breite	1,2	m
Kapazität	5	LHM
Kosten	1.140,00	€
Ladehilfsmittel		
LHM-Nummer	4	
Breite	223	mm
Tiefe	407	mm
Kosten	3.200,00	€
Informationstechnische Kommissioniererführung		
ITK-Nummer	4	
Bearbeitungszeit	22	s/pos
Kosten pro Kommissionierer	160,00	€
Kosten pro Fach	3,00	€
Sonstige Kosten	36.090,00	€
TCO		
	25.020.474,36	**€**

A-22 Optimale Konfiguration großes Kommissioniersystem

Konfiguration		
[25, 30, 19, 7, 4, 4]		
Leistungsdaten		
Aufträge pro Tag	6.000	
Max Positionen pro Auftrag	5	
Picks pro Position	10	
Lagerdaten		
Anzahl Gassen	25	
Breite eines Moduls	2,19	m
Lagerplätze	150.000	
Lagermittel		
LM-Nummer	19	
Lagermittel in Reihe	27	
Fachbreite	1.785	mm
Fachtiefe	1.092	mm
Ebenen	5	
Kosten	168,00	€
Fachkapzität	10	LHM
Fördermittel		
FM Nummer	7	
Max. Beschleunigung	1,05	m/s^2
Max. Geschwindigkeit	1,35	m/s
Breite	1,2	m
Kapazität	5	LHM
Kosten	1.140,00	€
Ladehilfsmittel		
LHM-Nummer	4	
Breite	223	mm
Tiefe	407	mm
Kosten	3.200,00	€
Informationstechnische Kommissioniererführung		
ITK-Nummer	4	
Bearbeitungszeit	22	s/pos
Kosten pro Kommissionierer	160,00	€
Kosten pro Fach	3,00	€
Sonstige Kosten	36.090,00	€
TCO		
	41.292.567,84	**€**

A-23 Optimale Konfiguration sehr großes Kommissioniersystem

Konfiguration		
[28, 30, 15, 7, 4, 2]		
Leistungsdaten		
Aufträge pro Tag	8.000	
Max Positionen pro Auftrag	5	
Picks pro Position	10	
Lagerdaten		
Anzahl Gassen	28	
Breite eines Moduls	1,77	m
Lagerplätze	201.600	
Lagermittel		
LM-Nummer	15	
Lagermittel in Reihe	30	
Fachbreite	1.191	mm
Fachtiefe	882	mm
Ebenen	6	
Kosten	614,00	€
Fachkapzität	10	LHM
Fördermittel		
FM Nummer	7	
Max. Beschleunigung	1,05	m/s^2
Max. Geschwindigkeit	1,35	m/s
Breite	1,2	m
Kapazität	5	LHM
Kosten	1.140,00	€
Ladehilfsmittel		
LHM-Nummer	4	
Breite	223	mm
Tiefe	407	mm
Kosten	3.200,00	€
Informationstechnische Kommissioniererführung		
ITK-Nummer	2	
Bearbeitungszeit	15	s/pos
Kosten pro Kommissionierer	85,00	€
Kosten pro Fach	16,00	€
Sonstige Kosten	35.379,00	€
TCO		
	57.880.585,58	**€**

8 Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und die den verwendeten Quellen und Hilfsmitteln wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

_____ _____

Ort, Datum Unterschrift