HCU | HafenCity Universität Hamburg

MASTERARBEIT

Bauingenieurwesen – Architectural Engineering

Nichtlineare Optimierung im Bauingenieurwesen

Entwicklung der n-1-dimensionalen Gruppierung als Methode zur Visualisierung von mehrdimensionalen Daten

Eingereicht an der HafenCity Universität Hamburg
zur Erlangung des akademischen Grades
Master of Science

Vorgelegt von
Simon Tabarelli
am 05.11.2018

Referent: Prof. Dr.-Ing. Annette Bögle
Korreferent: Kai Schramme M.Sc.

Abstract

In Zeiten, in denen der Klimawandel und der Naturschutz allgegenwärtige Themen sind, muss sich das Bauwesen die Frage stellen, durch welche Maßnahmen die Bauwerke der Zukunft ressourcenschonend und nachhaltig gebaut werden können. In dieser Hinsicht besitzen einige Disziplinen des Bauingenieurwesens, wie beispielsweise die Bauphysik und die Materialforschung, offensichtliches Optimierungspotential. Der konstruktive Ingenieurbau besitzt bei dem Entwurf und der Bemessung von Tragwerken ebenfalls ein großes Potential zur Optimierung, allerdings sind hier die Optimierungsmethoden hochgradig nichtlinear, komplex und zeitintensiv, weshalb sie nur selten angewendet werden. An dieser Problemstellung setzt das Thema dieser Arbeit an. Folgende These fasst den zentralen Lösungsansatz zusammen:

> *Visualisierung als Bindeglied zwischen dem komplexen Verfahren der nichtlinearen Optimierung und dessen Anwendung auf Probleme im konstruktiven Ingenieurbau*.

Hierfür wird im Rahmen dieser Arbeit eine Methode entwickelt, mit der die zu optimierenden Problemstellungen visuell dargestellt werden können. Die Visualisierung liefert durch die räumliche Darstellung nichtlinearer Zusammenhänge neue Erkenntnisse, die bei der Einordnung und Verifizierung von Optimierungsergebnissen helfen. Auch für das Problem der teilweise zeitintensiven Rechenprozesse bietet die neu entwickelte Methode eine Lösung. So lassen sich Wertebereiche von Variablenräumen bereits vor einem Optimierungsprozess eingrenzen, wodurch sich die Problemgröße und der damit verbundene Rechenaufwand deutlich reduzieren lassen.

Die Methode zur Visualisierung wird anhand von drei Benchmark Testproblemen getestet und erweist sich als leistungsstarkes Tool zur schnelleren und unkomplizierteren Lösung von nichtlinearen Optimierungsproblemen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation und Ziel der Arbeit

Optimierungsprozesse spielen in vielen Bereichen der Technik seit jeher eine große Rolle. Sie werden angewendet, um Rohstoffe ressourcenschonend und nachhaltig einzusetzen, Bauteilgeometrien entsprechend den Anforderungen optimal anzupassen und komplexe Prozesse und Arbeitsabläufe effizient zu gestalten. Gerade aus dem Bauingenieurwesen sind Optimierungsprozesse nicht mehr wegzudenken. Genau genommen sind die in den aktuellen Normen und Regelwerken festgelegten Bemessungskonzepte nichts anderes als verallgemeinerte Methoden zur Bauteiloptimierung. Eines der einfachsten Beispiele, und dennoch der in der Praxis mit am häufigsten angewendete Fall, ist die Bemessung eines Balkens. Hier fungieren die Querschnittsabmessungen als Variable und die Grenzen der Tragfähigkeit und Gebrauchstauglichkeit als einzuhaltende Bedingungen. Die Aufgabe des Ingenieurs ist, unter Wahrung aller Bedingungen, die Querschnittsabmessungen so zu wählen, dass der Balken mit einem möglichst geringen Materialaufwand realisiert werden kann.

Die Anwendung der bereits erwähnten verallgemeinerten Bemessungkonzepte garantiert für einfache Standardsysteme wie das des Balkens eine optimale Bemessung. Für komplexere Systeme oder Fragestellungen, die nicht durch eine Norm abgedeckt sind, gilt das allerdings nicht. Hier müssen mathematische Optimierungsverfahren angewendet werden, um die optimale Lösung einer komplexen Fragestellung innerhalb der Grenzen des Problems zu finden. Im Bauingenieurwesen treten jedoch häufig mehrdimensionale und hochgradig nichtlineare Fragestellungen auf, die auch mit den klassischen mathematischen Optimierungsverfahren nicht zufriedenstellend gelöst werden können. Für die Optimierung derartiger Probleme müssen nichtlineare Optimierungsmethoden, auch Black-Box Optimierung genannt, angewendet werden. Nichtlineare Optimierungen müssen bei der Zielwertsuche mit einem Minimum an Funktionswerten auskommen. Die Bewertung des Problems findet nur auf Grundlage der verfügbaren Informationen aus bereits bekannten Lösungen statt. Optimierungsergebnisse basieren also nicht auf analytisch lösbaren Funktionen, weshalb die nichtlineare Optimierung etwas despektierlich als *unschön* und *unsauber* bezeichnet wird. Dabei wird außen vor gelassen, dass diese Art der Zielwertsuche völlige Freiheit bei der Modellierung geeigneter Optimierungsmodelle bietet. Dimensionen, Nichtlinearitäten und diskrete Variablen spielen für die Anwendung der Optimierungsmethode keine nennenswerte Rolle.

Die Optimierung komplexer ingenieurtechnischer Probleme mit nichtlinearen Optimierungsmethoden bietet ein großes, bislang kaum genutztes Potential, gerade im Hinblick auf ressourcenschonendes und nachhaltiges Entwerfen beziehungsweise Bauen. Selbst in der Praxis werden komplexe Tragwerkssysteme in den meisten Fällen nur auf Grundlage der verallgemeinerten Bemessungskonzepte entworfen. Das kann dazu führen, dass Strukturen überdimensioniert sind und in puncto Wirtschaftlichkeit und Nachhaltigkeit nicht dem möglichen Optimum entsprechen.

Um die Vorteile der nichtlinearen Optimierung für zukünftige Bauvorhaben stärker nutzen zu können, soll mit der vorliegenden Arbeit eine Brücke zwischen der komplexen Anwendung und der auf Zeit und Effizienz getrimmten Praxis geschlagen werden. Das zentrale Ziel ist es, dem Leser die Anwendung von nichtlinearen Optimierungen auf typische Problemstellungen im Bauingenieurwesen zugänglicher zu machen.

1.2 Problemstellung und Lösungsansätze

Für die Implementierung nichtlinearer Optimierungsprobleme müssen alle Problemparameter, dazu gehören unter anderem Variablen, Bedingungen, Grenzen und Simulationen, in eine mathematische Beziehung zueinander gesetzt werden. Dabei entstehen aufgrund der Vielzahl an Problemparametern große Datenräume, deren Strukturen meistens hochgradig nichtlinear und n-dimensional sind. Das Problem dabei ist, dass Zusammenhänge innerhalb der komplexen Datenräume nicht mehr greifbar sind. Das heißt, es kann keine Aussage darüber getroffen werden, welchen Einfluss ein bestimmter Problemparameter auf das Ergebnis der Optimierung hat. Die komplexen Datenräume sind ein Problem für die Anwendung nichtlinearer Optimierungen im Bauingenieurwesen, da Optimierungsergebnisse oft nicht nachvollziehbar und reproduzierbar sind.

Das oben beschriebene Problem ist einer der Gründe dafür, dass nichtlineare Optimierungen für die Lösung ingenieurtechnischer Probleme selten angewendet werden. Um dieses Problem zu lösen, müssen Strukturen und Zusammenhänge innerhalb des komplexen Problems erkennbar werden. Dafür wird in dieser Arbeit vorgeschlagen, die n-dimensionalen Datenräume visuell darzustellen. Visualisierungen haben den Vorteil, dass mathematische Zusammenhänge durch den räumlichen Bezug zueinander erkennbar werden. Die Schwierigkeit bei der Visualisierung der Daten ist die Dimensionsreduzierung von n-D auf 3-D. Für die Dimensionsreduzierung und anschließende Visualisierung existieren zwar einige Methoden und Techniken, allerdings eignet sich keine in vollem Umfang für die Visualisierung von Optimierungsproblemen. Aus diesem Grund wird im Rahmen dieser Arbeit eine auf

künstlichen neuronalen Netzwerken basierende Methode entwickelt, die es dem Anwender erlaubt Daten beliebig hoher Dimensionen auf die Dimension 2 oder 3 zu reduzieren und anschließend visuell darzustellen. Mit der Methode, die im Folgenden als n-1-dimensionale Gruppierung bezeichnet wird, können im Anschluss an nichtlineare Optimierungsprozesse Beziehungen zwischen Problemparametern räumlich dargestellt werden.

Während der Entwicklung und Ausarbeitung der Methode der n-1-dimensionalen Gruppierung entstand die Idee, die neue Methode nicht nur zur Visualisierung von Problemparametern einzusetzen, sondern auch zur Visualisierung von Variablenräumen. Der Kerngedanke ist, dass durch die Visualisierung Zusammenhänge innerhalb eines Variablenraums erkannt werden und so die Größe eines Problems bereits vor dem eigentlichen Optimierungsprozess reduziert werden kann. Durch diese Möglichkeit kann ein weiteres Problem der nichtlinearen Optimierungen gelöst werden: Die zum Teil langen Rechenzeiten.
Trotz leistungsstarker Computer können einzelne Optimierungsprozesse mehrere Stunden in Anspruch nehmen, was die Benutzung nichtlinearer Optimierungen für die Praxis unattraktiv macht. Durch die Visualisierung des Variablenraums mit Hilfe der n-1-dimensionalen Gruppierung soll die Größe eines Problems so weit reduziert werden, dass der Optimierungsprozess in einer angemessenen Zeit durchgeführt werden kann.

Ein drittes, eher generelles Problem ist, dass sich nur wenige wissenschaftliche Arbeiten mit der Anwendung nichtlinearer Optimierung im Ingenieurwesen beschäftigen. Dies ist der Fall, da die nichtlineare Optimierung aus dem Bereich der angewandten Mathematik stammt und vorwiegend in der Informatik und Ökonomie angewendet wird. Für diese Bereiche existieren zahlreiche Veröffentlichungen und Algorithmus-Dokumentationen, in denen die Anwendungen erläutert und Benchmark Tests für die unterschiedlichsten Algorithmen durchgeführt werden. In den meisten Fällen werden dafür standardisierte Benchmark-Funktionen verwendet, deren Eigenschaften und Extremwerte bereits bekannt sind. Mathematiker und Informatiker können auf Grundlage der Testresultate schnell und einfach die Performance ihres eigenen Algorithmus mit der Performance anderer Algorithmen vergleichen. Für Optimierungsprobleme im Ingenieurwesen haben diese Testresultate jedoch kaum eine Bedeutung. Es kann keine verbindliche Aussage darüber getroffen werden, ob ein Algorithmus, der unter idealen Testbedingungen gut performt auch für die Anwendung auf Problemstellungen im Ingenieurwesen geeignet ist. Des Weiteren ist nicht klar, welche Algorithmus-Methode für welche Problemkategorie gut geeignet ist und welche nicht. Um diese „Informationslücke“ zu schließen, werden zum einen die Grundlagen der nichtlinearen

Optimierung in dieser Arbeit ausführlich erläutert, zum anderen werden eigene Benchmark Testprobleme entwickelt, mit deren Auswertungen gleich zwei Fragen geklärt werden können:

- Welche Algorithmus-Methode ist in Abhängigkeit der Problemkategorie die geeignetste?
- Welchen Einfluss hat die Reduktion des Optimierungsproblems auf die Ergebnisse der Optimierung?

1.3 Vorgehensweise

Um das Ziel dieser Arbeit zu erreichen, gilt es die im vorherigen Absatz diskutierten Probleme und Lösungsansätze grundlegend und detailliert zu erarbeiten. Der Aufbau dieser Arbeit lässt sich grob in die Abschnitte *Grundlagen*, *Entwicklung einer Methode zur Visualisierung* und *Benchmark Tests* gliedern.

Die einleitenden Kapitel 2, 3 und 4 geben einen fundierten Überblick über die Grundlagen der *nichtlinearen Optimierung,* der *Visualisierung von Optimierungsproblemen* und der *künstlichen neuronalen Netzwerke*. Die Grundlagen dieser Themengebiete sind essentiell für die darauf folgenden Abschnitte, weshalb die Erläuterungen in diesen Kapiteln recht detailliert und umfangreich sind. So ist diese Arbeit auch für Leser von Nutzen, die bisher noch nicht mit dem Thema nichtlineare Optimierung in Berührung gekommen sind.
Die Kapitel 5 und 6 sind der Kern dieser Arbeit. Hier wird die *n-1-dimensionale Gruppierung als Methode zur Visualisierung* entwickelt und vorgestellt. Zudem wird die Grundidee, Visualisierungen nicht nur zur Darstellung von Fitnesslandschaften, sondern auch zur *Reduzierung von Optimierungsproblemen* zu nutzen, ausformuliert. Die Visualisierung als Methode wird in dieser Arbeit als zentrales Bindeglied zwischen der nichtlinearen Optimierung und deren Anwendung im Bauingenieurwesen gesehen.
Abschließend werden in Kapitel 7 sechs unterschiedliche Black-Box Algorithmen gegen drei Benchmark Probleme getestet. Die Benchmark Testprobleme werden im Rahmen dieser Arbeit entwickelt und repräsentieren typische Fragestellungen aus dem konstruktiven Ingenieurbau.

Es wird angestrebt, die komplexen Zusammenhänge und Sachverhalte der nichtlinearen Optimierung so einfach und knapp wie möglich zu erläutern. Um allerdings der Thematik gerecht zu werden, ist es an manchen Stellen notwendig, inhaltlich weiter auszuholen und tiefer ins Detail zu gehen. Um dabei die Problemstellung und den Lösungsansatz dieser Arbeit nicht aus den Augen zu verlieren, ist der Aufbau und die Gliederung in folgendem Ablaufdiagramm grafisch dargestellt.

Problemstellung

Das Verfahren der nichtlinearen Optimierung ist zu komplex und langwierig,
um es effektiv für die Lösung ingenieurtechnischer Probleme einzusetzen.

Zentraler Lösungsansatz

Visualisierung als Bindeglied zwischen dem komplexen Verfahren der nichtlinearen Optimierung
– und –
dessen Anwendung auf Probleme im konstruktiven Ingenieurbau

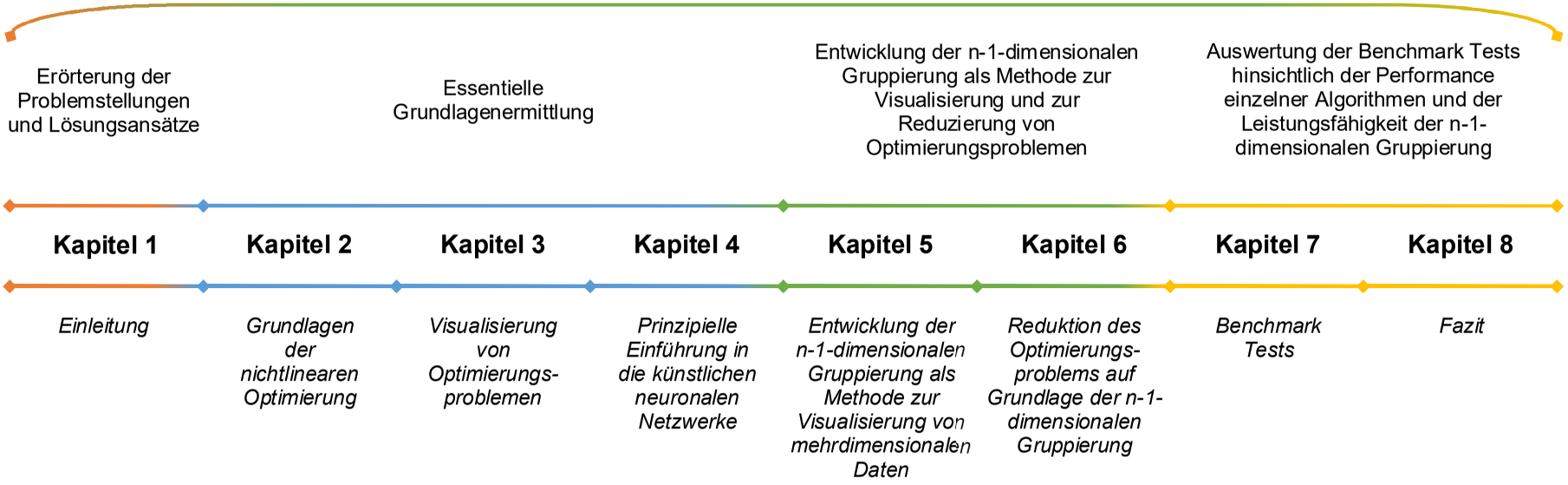


Abbildung 1: Grafische Zusammenfassung der Abschnitte und Kapitel dieser Arbeit

2 Grundlagen der nichtlinearen Optimierung

In der Mathematik, der Ökonomie und in so gut wie jeder Naturwissenschaft kommen in der einen oder anderen Art und Weise Optimierungsprozesse zum Einsatz. Jedoch sind im Ingenieurwesen die Probleme meist so komplex und hochgradig nichtlinear, dass klassische Optimierungsmethoden nicht anwendbar sind oder keine zufriedenstellenden Lösungen liefern. Aus diesem Grund kommen in dieser Arbeit nichtlineare Optimierungen zum Einsatz. Die nichtlineare Optimierung, auch Black-Box Optimierung genannt, ist eine Disziplin der angewandten Mathematik. Im Grundsatz geht es darum, das Optimum eines nichtlinearen Problems beziehungsweise einer nichtlinearen Zielfunktion zu bestimmen. Das Optimum entspricht meistens einem minimalen oder maximalen Extremwert, wobei aber auch hinsichtlich eines vorgegebenen Zielwerts optimiert werden kann. Eine Einführung in das Thema der Black-Box Optimierung ist in Absatz 2.3 zu finden.

Spricht man von unterschiedlichen Optimierungsmethoden, ist eigentlich die jeweilige Arbeitsweise des Optimierungsalgorithmus gemeint. Der grundlegende Aufbau einer Optimierung ist dabei unabhängig von der verwendeten Methode oder dem verwendeten Algorithmus. In den folgenden Absätzen 2.1 und 2.2 werden zunächst die grundlegenden Begriffe und anschließend der Optimierungsprozess im Allgemeinen vorgestellt.

2.1 Begriffsdefinitionen

Optimierungsproblem

Als Optimierungsproblem bezeichnet man eine Funktion oder Simulation, deren Optimum zunächst nicht bekannt ist. Die einfachsten Optimierungsprobleme werden durch analytische Funktionen beschrieben, die durch das Bestimmen der Nullstellen der ersten Ableitung gelöst werden können. Nichtlineare Optimierungsprobleme sind auf diese Weise nicht mehr zu lösen, weshalb sie auch ableitungsfreie Optimierungen genannt werden.

Das Optimum ist ein Extremwert der Fitnessfunktion, wobei das Minimum der Fitnessfunktion f gleich dem Maximum von $-f$ ist. Aus diesem Grund ist es zweitrangig, ob das gesuchte Optimum ein Maximum oder Minimum ist.

Variablen

Eine Struktur kann durch die unterschiedlichsten Variablen beschrieben werden. Während des Optimierungsprozesses werden diese Variablen von dem Optimierungsalgorithmus verändert und angepasst. Nicht zu verwechseln sind die Variablen mit den Parametern, denn Parameter sind fest definierte Werte, die durch die Optimierung nicht verändert werden. Ein Balken wird beispielsweise durch seine Länge, Breite und Höhe definiert. Soll der Querschnitt des Balkens optimiert werden, ist die Länge ein Parameter und die Breite und Höhe sind Variablen.

Stetige und diskrete Variablen

Aus mathematischer Sicht können Variablen in diskrete und stetige Variablen unterteilt werden. Ein Variable gilt dann als diskret, wenn sie nur eine abzählbare Anzahl an Ausprägungen einnehmen kann. Das bedeutet, die Menge der möglichen Ausprägungen ist endlich. Bei der Anzahl der Druckstreben eines Fachwerks oder Seilen einer Schrägseilbrücke ist das beispielsweise der Fall. Oft ist die Anzahl der möglichen Ausprägungen nicht nur abzählbar, sondern lässt sich vorab sogar vollständig eingrenzen. Für den Optimierungsalgorithmus bedeuten diskrete Variablen meistens einen Sprung in den Funktionswerten (Fitnesswerten) beziehungsweise in der Fitnesslandschaft. Es soll beispielsweise ein Fachwerkträger mit einer variablen Anzahl an Druckstreben (zwischen 6 und 10) hinsichtlich Gewicht und Verformung optimiert werden. Springt die Anzahl an Druckstreben von 7 auf 8, gibt es eine abrupte Veränderung in den Fitnesswerten. An entsprechender Stelle weist die Fitnesslandschaft einen Sprung auf.

Das Gegenteil von diskreten Variablen sind stetige Variablen. Diese sind dadurch definiert, dass sie unendlich viele Ausprägungen annehmen können. Mathematisch bedeutet das, dass sich zwischen zwei Ausprägungen immer eine weitere Ausprägung befinden kann. Dies ist beispielsweise bei Längenangaben der Fall. Zwischen *5,1 mm* und *5,2 mm* befindet sich *5,15 mm*. Die Variablen parametrisierter Entwürfe sind allerdings nie komplett stetig. Auch die Variablen von Längeneinheiten haben immer eine Grenze, meistens einen Minimal- und Maximalwert. Spricht man also von stetigen Variablen, sind eigentlich quasi-stetige Variablen gemeint. Per Definition sind quasi-stetige Variablen diskret und können nur eine abzählbare Menge an Ausprägungen annehmen, wobei die Menge der möglichen Ausprägungen so groß ist, dass sie sich wie stetige Variablen verhalten.

Ein Beispiel ist der Durchmesser eines Rohres. Der Durchmesser kann zwischen *10,0 mm* und *100,0 mm* in *0,1 mm* Schritten variieren. Die möglichen Ausprägungen sind begrenzt und

auch abzählbar, jedoch ist aufgrund der geringen Schrittweite die Menge der möglichen Durchmesser so groß, dass der Optimierungsalgorithmus beim Variieren des Durchmessers keine Sprünge in der Fitnesslandschaft wahrnimmt.

Dominierende und untergeordnete Variablen

Als dominierend wird eine Variable bezeichnet, die unabhängig von allen anderen Variablen den Fitnesswert beeinflusst. Das Gegenteil sind untergeordnete Variablen. Sie haben nur einen geringen Einfluss auf den Fitnesswert und können von anderen Variablen dominiert werden. Beispielsweise hat auf das Flächenträgheitsmoment eines Doppel-T-Profils die Profilhöhe einen deutlich größeren Einfluss als die Stegbreite. In diesem Fall wird die Variable der Stegbreite von der Variable der Profilhöhe dominiert und könnte somit von dem Optimierungsprozess ausgeschlossen werden.

Variablensets

Ein vollständiges Variablenset enthält von jeder Variable genau einen Wert. Das heißt, jedes Variablenset repräsentiert eine mögliche Lösung des Optimierungsproblems. Soll der Rechteckquerschnitt eines Biegebalkens über die Variation der Höhe und Breite optimiert werden, besteht das dazugehörige Variablenset aus zwei Werten. Der Höhe und der Breite.

Dimensionalität

Die Dimensionalität eines Optimierungsproblems wird durch die Anzahl der Variablen bestimmt. Generell gilt, je höher die Dimensionalität eines Optimierungsproblems desto aufwendiger und komplexer ist dessen Lösung. Um ein Problem möglichst klein und einfach zu halten, werden unwichtige Variablen von dem Optimierungsprozess ausgeschlossen. Dazu muss eine Kategorisierung in dominierende und untergeordnete Variablen vorgenommen werden.

Lokale und globale Optimierung

Als Analogie für die Menge der Lösungen eines Optimierungsproblems wird eine Gebirgslandschaft genutzt. Dabei sind Berge als Maxima und Täler als Minima der Fitnessfunktion zu verstehen. Ist es also die Aufgabe, von einem beliebigen Startpunkt aus den höchsten Berg in der unmittelbaren Nachbarschaft zu finden, spricht man von einer lokalen Optimierung. Soll jedoch der höchste Berg in der gesamten Landschaft, sozusagen der Gipfel des Gebirgsmassivs gefunden werden, spricht man von einer globalen Optimierung. Ob ein lokales oder globales Optimierungsproblem vorliegt, hängt sowohl vom

Optimierungsziel, als auch von der Kategorie des Optimierungsproblems ab. Für manche Probleme existiert lediglich eine globale Lösung, das heißt, neben diesem absoluten Optimum existieren keine weiteren Lösungen, die als gut bewertet werden können. Andere Optimierungsprobleme besitzen eine Vielzahl an ähnlich guten oder auch gleich guten Lösungen. In diesen Fällen ist es schwer, ein absolutes Optimum zu bestimmen.

Grenzen und Bedingungen

Alle Optimierungsprobleme sind in irgendeiner Art und Weise begrenzt oder gegebenen Bedingungen unterworfen. In der Praxis entsprechen die Grenzen meistens den geometrischen Randbedingungen eines Entwurfs, wie beispielsweise den minimalen und maximalen Längen, Breiten und Höhen. Die Grenzwerte werden jeweils über den entsprechenden Variablenraum gesteuert. Eine Grenze ist immer statisch, sie wird zu Beginn festgelegt und ändert sich während des Optimierungsprozesses nicht. Je enger die Grenzen gefasst sind, desto weniger mögliche Lösungen gibt es beziehungsweise desto kleiner wird das Problem.

Im Gegensatz zu Grenzen können Bedingungen sowohl statisch als auch dynamisch sein. Bedingungen können nicht über den Variablenraum zu Beginn festgelegt werden, sondern müssen über eine Penalty-Funktion in die Fitnessfunktion eingebettet werden. Ein Beispiel für eine statische Bedingung ist die Vorgabe einer maximalen Verformung. Übersteigt die Verformung den Grenzwert, wird der Fitnesswert durch die Penalty-Funktion künstlich erhöht und verschlechtert somit das Ergebnis. Dynamische Bedingungen sind Abhängigkeiten der Variablen untereinander. Folgende Abhängigkeit ist eine dynamische Bedingung:

Variablen	Grenze	dynamische Bedingung
h .. Höhe b .. Breite	$[10\ cm; 100\ cm]$ $[10\ cm; 100\ cm]$	wenn $h \geq 50\ cm$; dann $b + 0{,}5 \cdot h$; sonst b

Solange Variable h kleiner als 50 cm ist, kann Variable b einen freien Wert zwischen 10 cm und 100 cm einnehmen. Ist h allerdings größer gleich 50 cm, ist b von h durch $(+ 0{,}5 \cdot h)$ abhängig. Abhängigkeiten können entweder in Form einer Bestrafung durch die Penalty-Funktion berücksichtigt werden, oder sie werden als mathematische Bedingung direkt in der Fitnessfunktion definiert, sodass keine Variablenkombinationen zugelassen werden, die nicht der Bedingung entsprechen.

Grenzen und Bedingungen sind unerlässlich für die Formulierung von Optimierungsproblemen. Allerdings muss darauf geachtet werden, dass ein Problem nicht

überbestimmt ist. Das ist dann der Fall, wenn sich unterschiedliche Grenzen und Bedingungen gegenseitig ausschließen.

In dem oben eingeführten Beispiel wäre das der Fall, wenn $h = 100\ cm$ und $b = 75\ cm$ ist. Laut der Bedingung müsste b dann den Wert $b = 75 + 0.5 \cdot 100 = 125\ cm$ einnehmen, wobei der Variablenraum von b auf 100 cm begrenzt ist. Durch solche sich ausschließenden Grenzen und Bedingungen entstehen Lücken oder nicht definierte Bereiche, in denen die Penalty-Funktion den Fitnesswert künstlich erhöhen muss. Existieren zu viele Lücken, ist ein Problem stark überbestimmt und es kann eventuell kein Optimum gefunden werden.

Einkriterielle und multikriterielle Optimierungsprobleme

Als einkriterielles Problem oder auch Einzielproblem bezeichnet man Probleme, die hinsichtlich nur eines Kriteriums optimiert werden sollen, beispielsweise der minimalen Verformung. Allerdings ist vor allem im Bauingenieurwesen die Optimierung hinsichtlich eines Kriteriums oft nicht zielführend, da das Optimum eines Kriteriums meistens nur zu Lasten eines anderen Kriteriums erreicht werden kann.

Aus diesem Grund gibt es multikriterielle Optimierungsprobleme, auch Mehrzielprobleme genannt. Ein Beispiel ist die Optimierung hinsichtlich Verformung und Konstruktionseigengewicht. Hier ist klar, dass nicht das absolute Optimum in beiden Kriterien gleichzeitig erreicht werden kann. Ist die Konstruktion möglichst leicht, weist sie vermutlich eine große Verformung auf. Ist die Verformung minimal, wird das Eigengewicht größer sein. In diesem Fall spricht man von gegenläufigen Zielen, bei denen die optimale Lösung einen Kompromiss zwischen beiden Zielen darstellt.

In der Literatur wird oft von einkriteriellen Optimierungsproblemen mit Nebenbedingungen gesprochen. Genau genommen sind das auch multikriterielle Probleme, allerdings wird hier eine Unterscheidung zwischen dem Hauptziel und Nebenzielen gemacht. Nach dem jeweiligen Hauptziel wird optimiert, wobei die Nebenziele (Nebenbedingungen) eingehalten werden sollen.

Multikriterielle Optimierungsprobleme benötigen spezielle Optimierungsalgorithmen, die die Ziele gegeneinander abwägen und die besten Kompromisse als Lösungen präsentieren. Da jedoch die Implementierung und Anwendung solcher Algorithmen recht komplex ist, werden in dieser Arbeit alle multikriteriellen Probleme in einkriterielle umgewandelt. Hierzu wird entweder die Methode der Penalty-Funktion verwendet, oder die Fitnesswerte werden je Ziel einzeln berechnet und anschließend gewichtet miteinander addiert.

Fitnessfunktion

Das Optimierungsproblem ist im Allgemeinen durch die Fitnessfunktion oder auch Zielfunktion definiert, wobei der Begriff Funktion nicht in allen Fällen zutreffend ist. Während einfache, lineare Problemstellungen durch eine herkömmliche Funktion beschrieben werden, benötigen nichtlineare Probleme zur Implementierung oft komplexe Computersimulationen. Fitnessfunktionen nichtlinearer Probleme können in zwei Teilbereiche unterteilt werden: Den speziellen Solver und die Penalty-Funktion.

Spezieller Solver

Die Aufgabe im Optimierungsprozess verleiht dem speziellen Solver seinen Namen. Während der Optimierungsalgorithmus problemunabhängig arbeitet, kann der spezielle Solver jeweils nur eine spezielle Aufgabe lösen. Im Bauingenieurwesen sind das meistens Finite-Element-Berechnungen zur Bestimmung von Querschnittswerten, Verformungen oder Massen. Im Bereich der Bauphysik können aber auch Tageslicht-, Solarstahlen-, und Wärmebrücken-Simulationen durchgeführt werden. Für jede dieser Aufgaben wird ein spezieller Solver benötigt. In dieser Arbeit wird für die Finite-Element-Berechnung die Grasshopper Komponente Karamba 3D verwendet. Mehr dazu in Absatz 2.5 *Software*.

Penalty-Funktion

Die Verwendung von Penalty-Funktionen ist eine Methode zur Umwandlung von Optimierungsproblemen mit Nebenbedingung in ein einkriterielles Problem. Die Idee dabei ist, die Verletzung von Nebenbedingungen durch die Erhöhung des Fitnesswerts zu bestrafen. Dazu wird durch die Penalty-Funktion in Abhängigkeit der Höhe der Verletzung ein Penaltyfaktor ermittelt, mit dem der Fitnesswert multipliziert wird. Anschließend kann die Optimierung als ein einkriterielles Optimierungsproblem behandelt werden, bis die Verletzung der Nebenbedingung klein genug ist. Die Implementierung einer Penalty-Funktion könnte wie folgt aussehen:

$$\text{Fitnesswert} = f(x) \cdot p$$

mit: $f(x)$ Funktionswert der Fitnessfunktion

$n_b(x)$ Nebenbedingung

p Penaltyfaktor $p = \begin{cases} n_b(x) \; wenn \; n_b(x) \geq 1 \\ sonst \; 1 \end{cases}$

Fitnesswert

Als Fitnesswert bezeichnet man den Funktionswert der Fitnessfunktion. Anhand des Fitnesswertes wird die Performance eines Problems gemessen. Nimmt der Fitnesswert einen Extremwert ein, entspricht dieser der optimalen Lösung des Problems.

Fitnesslandschaft

Im Allgemeinen wird der Funktionsgraph der Fitnessfunktion als Fitnesslandschaft bezeichnet. Wie auch bei allgemeinen Funktionen, wird die Variable x mit den dazugehörigen Funktionswerten $f(x)$ dargestellt, wobei man im Bereich der Optimierung nicht von Funktionswerten, sondern von Fitnesswerten spricht. In Anlehnung an eine Landschaft in der Natur lassen sich die Eigenschaften einer Fitnesslandschaft recht anschaulich durch Berge, Täler, Ebenen oder auch Schluchten und Klippen beschreiben. Wie bereits beschrieben, erhält man durch die Darstellung von Fitnesslandschaften eine Vielzahl an Informationen über die Art des Optimierungsproblems beziehungsweise über die verwendete Fitnessfunktion.

In untenstehender Abbildung ist beispielhaft die Fitnesslandschaft eines eindimensionalen und zweidimensionalen Optimierungsproblems dargestellt, wobei die Landschaft eines eindimensionalen Problems durch eine einfache Kurve repräsentiert wird. Die zu sehenden Hoch- und Tiefpunkte sind die Berge und Täler der Fitnesslandschaft und markieren die Extremwerte der Fitnessfunktion. Das Ziel jeder Optimierung ist es, diese Extremwerte in der Fitnesslandschaft zu finden. Bei den untenstehenden Beispielen ist diese Aufgabe sogar für den Menschen schnell zu lösen, bei komplexeren, mehrdimensionalen Problemen ist das allerdings nicht mehr möglich.

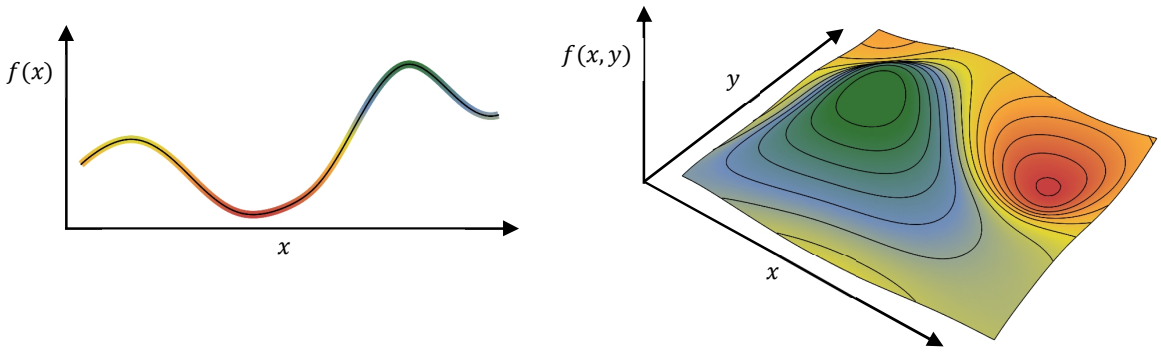


Abbildung 2: Fitnesslandschaft eines 1-D (links) und 2-D Problems (rechts)

An dieser Stelle muss erwähnt werden, dass die vollständige Darstellung einer Fitnesslandschaft, bei einer ausreichenden Größe des Optimierungsproblems, nahezu unmöglich ist. Damit eine vollständige und korrekte Landschaft dargestellt werden kann,

müssten alle Lösungen des entsprechenden Problems bekannt sein. Hinzu kommt, dass die meisten Optimierungsprobleme multidimensional sind. Die entsprechenden Landschaften können weder dargestellt werden, noch sind sie für den Betrachter nachvollziehbar. Probleme mit mehr als zwei Dimensionen müssen so bearbeitet, komprimiert oder skaliert werden, dass eine Landschaft mit maximal drei Dimensionen erstellt werden kann. Dieser Vorgang wird im Allgemeinen als Dimensionsreduktion bezeichnet. Hierzu steht eine Vielzahl an unterschiedlichen mathematischen Ansätzen zur Verfügung, wobei immer in irgendeiner Art und Weise ein Datenverlust in Kauf genommen werden muss.

In Kapitel 3 *Visualisierung von Optimierungsproblemen* werden mehrere Prozesse zur Visualisierung von mehrdimensionalen Problemen vorgestellt.

2.2 Optimierungsprozess

Der hier beschriebene Optimierungsprozess ist unabhängig von dem gewählten Optimierungsalgorithmus allgemeingültig. Die Variablen und die Fitnessfunktion, bestehend aus dem speziellen Solver und der Penalty-Funktion, bilden zusammen das Optimierungsproblem. Präsentiert man dem Problem entsprechende Variablenwerte als Input, wird der dazugehörige Fitnesswert berechnet. Wo der berechnete Fitnesswert in Bezug auf Maxima und Minima einzuordnen ist, kann nicht bestimmt werden. Für diese Aufgabe ist der Optimierungsalgorithmus zuständig. Er analysiert den Fitnesswert und verändert die Variablen entsprechend seiner Optimierungsmethode so lange, bis ein Extremwert gefunden wurde oder eine Abbruchbedingung erfüllt ist. Die Methode zur Entscheidung, ob ein Extremwert gefunden wurde, hängt von dem verwendeten Optimierungsalgorithmus ab. In der Regel gilt, dass ein Extremwert gefunden wurde, wenn sich der Fitnesswert über mehrere Iterationsschritte nur noch minimal verändert hat. Der Optimierungsalgorithmus gilt dann als konvergiert.

Falls kein Extremwert gefunden wird, dienen Abbruchbedingungen als Ausweg aus einer Endlosschleife. Eine Obergrenze der Rechenzeit oder der Anzahl der maximalen Iterationsschritte kann beispielsweise als Abbruchbedingung dienen.

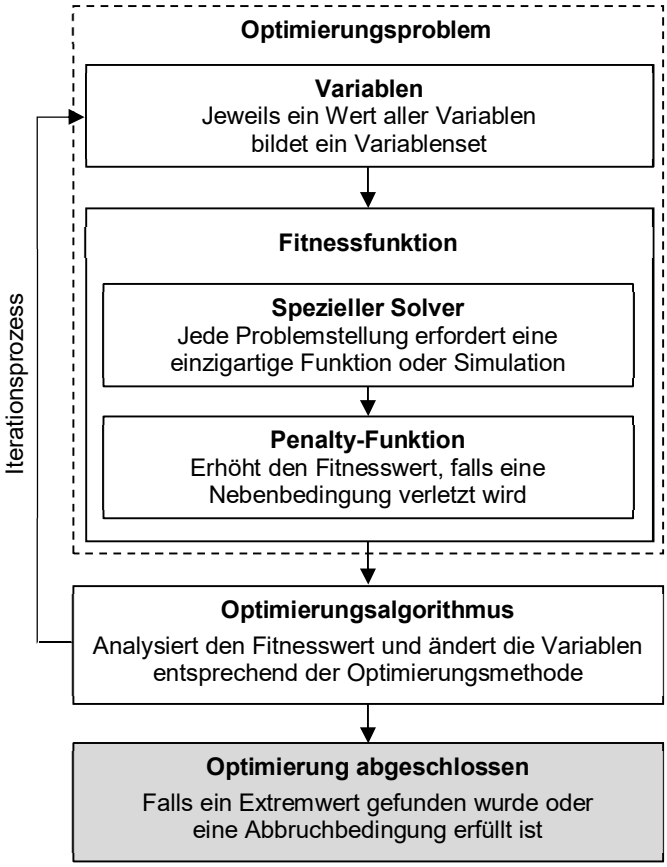


Abbildung 3: Optimierungsprozess

2.3 Grundlagen zur Differenzierung von Optimierungsproblemen

Optimierungsprobleme können sich in vielerlei Hinsicht voneinander unterscheiden. Einige Eigenschaften sind direkt zu bestimmen oder werden durch den Anwender vorgegeben, andere sind nur indirekt über die Visualisierung der Fitnesslandschaft zu bestimmen. Zur Einordnung der Optimierungsergebnisse ist es hilfreich, die Eigenschaften des Problems zu kennen. Auch die Wahl eines geeigneten Optimierungsalgorithmus hängt von der Art des Problems ab, wobei hierfür nur die Eigenschaften betrachtet werden können, die bereits vor dem Optimierungsprozess bekannt sind.

Unterschiedliche Landschaftstypologien und deren Merkmale werden in dem folgenden Absatz 2.3.1 dargestellt, gefolgt von dem Absatz 2.3.2, in dem weitere Merkmale vorgestellt werden, die nicht anhand der Fitnesslandschaft zu bestimmen sind.

2.3.1 Differenzierung anhand der Typologie der Fitnesslandschaft

Berg oder Gebirge

Aufgrund der Anzahl an Gipfeln und Tälern in einer Landschaft lässt sich ein Optimierungsproblem als lokal oder global einordnen. Für den Tragwerksplaner ist das eine sehr wichtige Information, da ein Problem mit mehreren lokalen Extremwerten einen gewissen Entscheidungsspielraum bei der Wahl der Lösung zulässt. Probleme mit nur einem globalen Extremwert geben die Lösung quasi vor.

Wird von Extremwerten gesprochen, sind Minima und Maxima in der Fitnesslandschaft gemeint. Je nach Optimierungsziel wird ein Minimum oder Maximum gesucht, wobei das Minimum von $f(x)$ dem Maximum von $-f(x)$ entspricht. Bleibt man in der Analogie der Landschaft, weist die Fitnesslandschaft mit globalen Extremwerten einen Berg und ein Tal auf. Sind mehrere Berge zu erkennen, muss der Gipfel eines Berges eindeutig als der höchste zu identifizieren sein. Die Fitnesslandschaft eines Problems mit lokalen Extremwerten gleicht eher einem Gebirgsmassiv, wobei auf den ersten Blick kein höchster Gipfel auszumachen ist. Mehrere Berge haben eine ähnliche Höhe, beziehungsweise mehrere Variablensets haben einen ähnlichen Fitnesswert, siehe Abbildung 4.

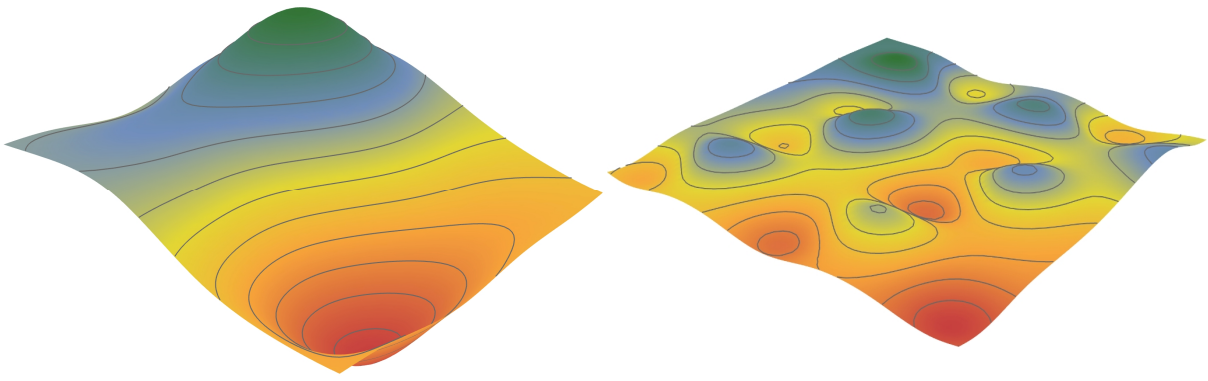


Abbildung 4: Globale Extremwerte (links), lokale Extremwerte (rechts)

Auch für die Wahl des Optimierungsalgorithmus ist die Unterscheidung zwischen lokalen und globalen Extremwerten nicht unerheblich. Einige Algorithmen sind darauf spezialisiert, lokale Extremwerte zu finden, was dazu führen kann, dass andere, vielleicht sogar bessere Lösungen nicht gefunden werden. Existieren mehrere ähnlich gute Lösungen, kann es passieren, dass ein global suchender Algorithmus nicht konvergiert und schließlich keine Lösung gefunden wird. Oft kann erst nach dem Optimierungsprozess die Art der Extremwerte

anhand der Fitnesslandschaft bestimmt werden. Stellt man fest, dass der gewählte Optimierungsalgorithmus nicht der richtige war, muss der Optimierungsprozess noch einmal neu gestartet werden.

Sprünge und Klippen

Stetige Optimierungsprobleme stellen in der Regel keine Probleme für die Optimierungsalgorithmen dar, da die Fitnesslandschaften keine Sprünge aufweisen. Im Gegensatz dazu sind auf der Fitnesslandschaft diskreter Probleme häufig Sprünge oder undefinierte Bereiche zu finden. Sprünge bedeuten einen abrupten Wechsel der Steigung, was zumindest für Algorithmen der Downhill-Simplex Methode ein Problem darstellt. Sie versuchen durch permanentes bergauf Laufen so schnell wie möglich den Gipfel zu finden. Geraten sie von oben an einen Sprung, laufen sie nicht weiter, obwohl es nach dem Sprung eventuell wieder bergauf geht. Befindet sich der Gipfel hinter einem Sprung, wird er nicht gefunden.

Abbildung 4 auf Seite 24 zeigt die Fitnesslandschaft eines stetigen Optimierungsproblems. In Abbildung 5 ist die Fitnesslandschaft eines diskreten Problems dargestellt, wobei links der Sprung als nicht definierter Bereich zu sehen ist. Da in dieser Arbeit alle Darstellungen von Fitnesslandschaften durch NURBS-Flächen erzeugt werden, können Sprünge nicht als tatsächliche Lücken oder undefinierte Bereiche dargestellt werden. NURBS-Flächen haben die Eigenschaft, jede 3-D Geometrie als eine stetige Fläche abzubilden. Aus diesem Grund wird anstelle eines Sprungs eine steile Klippe dargestellt, siehe Abbildung 5 rechts.

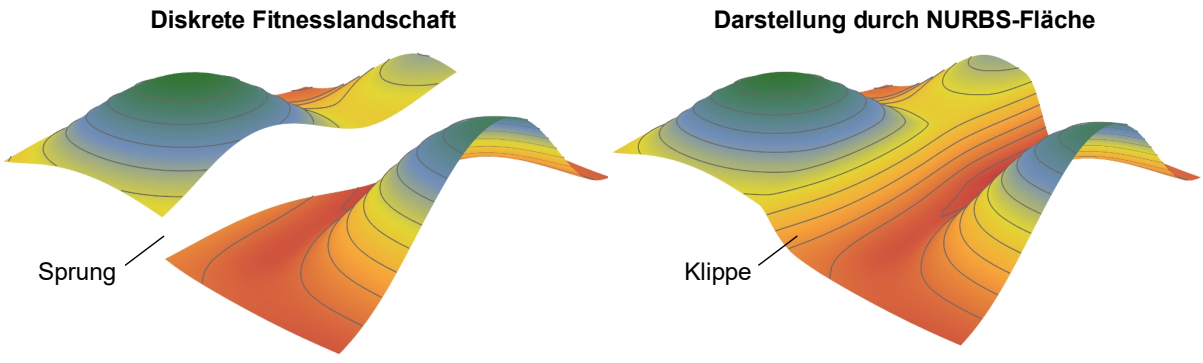


Abbildung 5: Darstellung einer diskreten Fitnesslandschaft durch NURBS-Flächen

Das bereits erwähnte Problem bei der Extremwertsuche diskreter Optimierungsprobleme wird anhand der isometrischen Darstellung einer Fitnesslandschaft in Abbildung 6 verdeutlicht. Der Weg, den der Algorithmus während des Optimierungsprozesses zurückgelegt hat, wird durch die rote Linie markiert. Punkt S ist der zufällig ausgewählte Startpunkt und die Gipfel G_1 und

G_2 markieren zwei Optima. Anhand der roten Linie ist zu erkennen, dass der Optimierungsalgorithmus, ausgehend vom Startpunkt, den schnellsten Weg bergauf sucht. In dem Bereich von Punkt K stößt er an den Rand der Klippe und kehrt um, da er sonst nicht mehr bergauf, sondern steil bergab gehen würde. In der Konsequenz wird irgendwann der Gipfel G_1 gefunden und als die optimale Lösung des Problems präsentiert. Der Gipfel G_2 wurde in dem Optimierungsprozess nicht berücksichtigt, obwohl er unter Umständen eine bessere Lösung bieten würde.

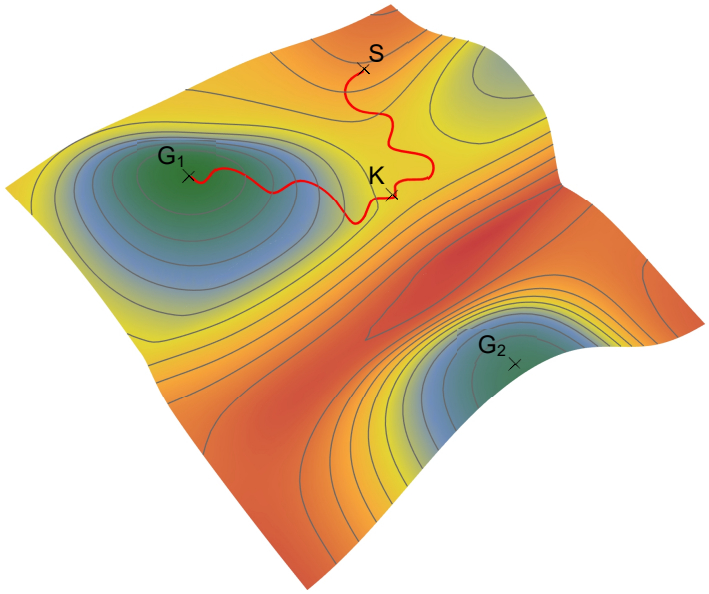


Abbildung 6: Nicht beachtete Lösungen bei diskreten Optimierungsproblemen

Ebenen

Es ist vor einer Optimierung nicht immer offensichtlich, ob alle Variablen gleichwertig sind oder ob einzelne Variablen von anderen dominiert werden. Anhand der Fitnesslandschaft kann das unter Umständen erkannt werden, denn die selbstorganisierende Karte weiß während des Visualisierungsprozesses nicht, welche Variablen dominant oder untergeordnet sind. Hier werden alle Variablen gleichbehandelt. Die Folge ist, dass die Fitnesslandschaft große Ebenen mit ähnlich guten Fitnesswerten aufweist. In der Region der Ebene ändern sich nur die untergeordneten Variablen, die dominierenden bleiben gleich. Da die Untergeordneten den Fitnesswert nur gering beeinflussen, entsteht eine Ebene, siehe folgende Abbildung 7.

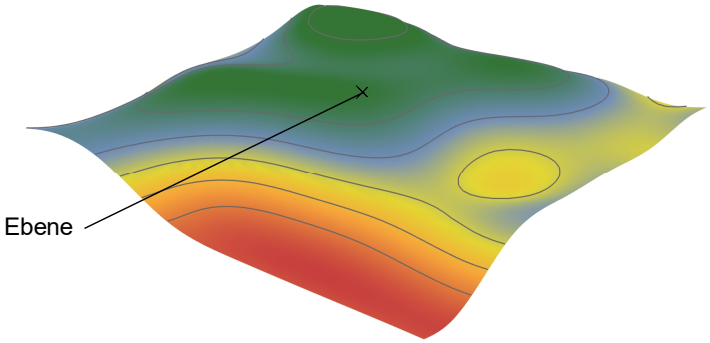


Abbildung 7: Fitnesslandschaft repräsentiert untergeordnete Variablen

An dieser Stelle muss erwähnt werden, dass die Existenz von Ebenen nicht zwangsläufig auf untergeordnete Variablen schließen lässt. Ebenen können auch bedeuten, dass gleichwertige Variablen gleichgute Lösungen liefern. Oft ist das der Fall bei symmetrischen Problemen. Dann existieren mindestens zwei unterschiedliche Variablensets mit dem gleichen Fitnesswert.

Risse und Lücken

Die Fitnesslandschaft von überbestimmten Optimierungsproblemen weist Risse oder Lücken auf, siehe Abbildung 8 links. Wie bereits beschrieben, werden die Landschaften in dieser Arbeit mit Hilfe von NURBS-Flächen visualisiert. Da NURBS-Flächen keine Diskontinuitäten zulassen, werden Risse und Lücken als tiefe Krater dargestellt, siehe Abbildung 8 rechts. Wann ein Optimierungsproblem überbestimmt ist, wird in Absatz 2.1 *Grenzen und Bedingungen* beschrieben.

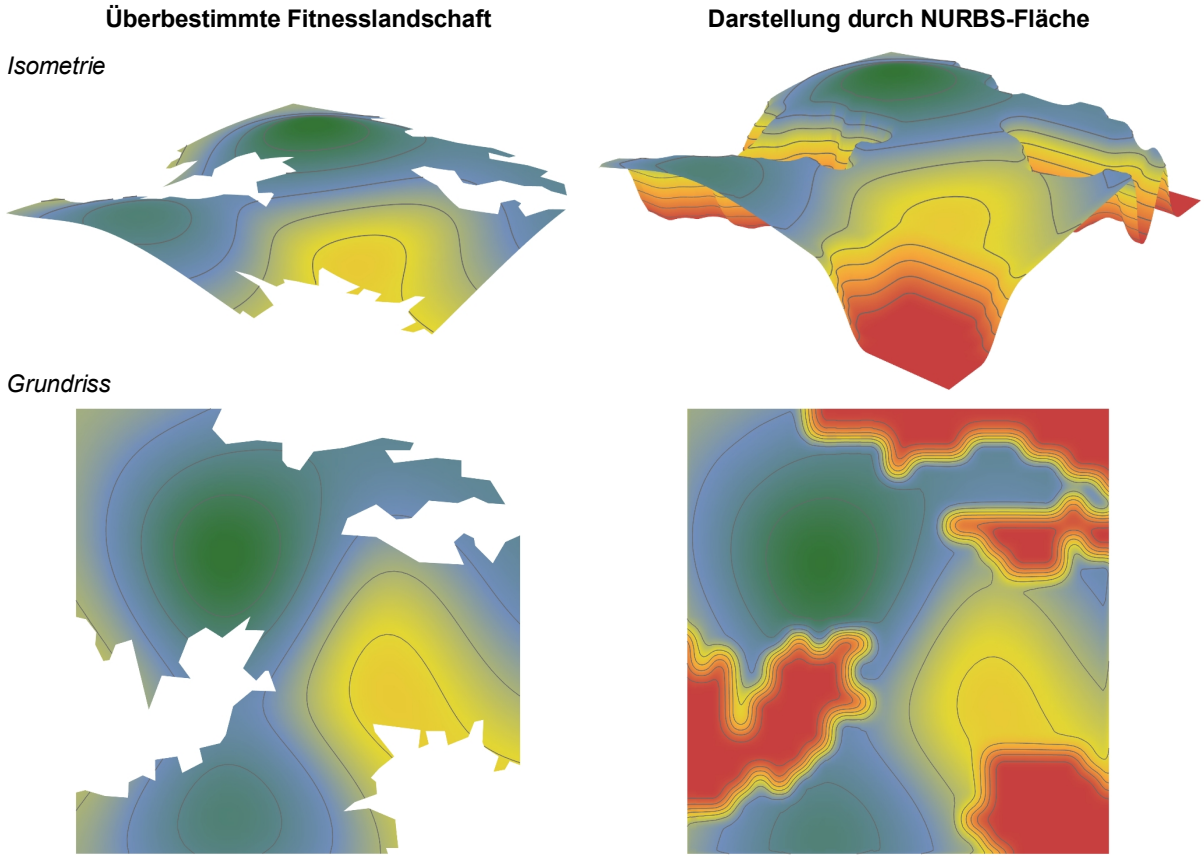


Abbildung 8: Überbestimmte Fitnesslandschaft dargestellt durch eine NURBS-Fläche

Risse und Lücken sind nicht zwangsläufig ein Indiz für ein überbestimmtes Optimierungsproblem. Durch die Verwendung von Penalty-Funktionen kann es auch dazu kommen, dass einzelne Fitnesswerte so stark erhöht werden, dass es aussieht, als ob das Problem Definitionslücken besitzt. Problematisch sind diese Risse und Lücken erst, wenn sie einen Großteil der Landschaft einnehmen und keine stetigen Bereiche mehr zu erkennen sind. In einem solchen Fall sollten die Grenzen und Bedingungen und gegebenenfalls auch die Penalty-Funktion noch einmal überdacht und gegebenenfalls angepasst werden.

2.3.2 Weitere Merkmale

Dimensionalität

Die Komplexität eines Problems steigt exponentiell mit der Dimensionalität. Aus diesem Grund spielt die Höhe der Dimension bei der Einordnung eines Problems eine große Rolle. Die Ergebnisse der jährlichen *GECCO Black Box Optimization Competition*, kurz *BBComp* zeigen, dass die Downhill-Simplex-Methode und die Ersatzmodell-Methode bei Problemen niedriger Dimensionalität vielversprechende Optimierungsergebnisse liefern. Bei Problemen

höherer Dimensionalität haben diese Methoden allerdings ihre Schwierigkeiten. Metaheuristiken scheinen hier stabiler und zuverlässiger zu funktionieren [6]. Die Downhill-Simplex-Methode, die Ersatzmodell-Methode und die Metaheuristiken werden in Absatz 2.4 vorgestellt.

Anzahl der Zielfunktionen und Nebenbedingungen

Ob ein einkriterielles oder multikriterielles Optimierungsproblem vorliegt, hat einen großen Einfluss auf die Wahl des Optimierungsalgorithmus, denn nicht alle Algorithmen können mit mehreren Zielfunktionen gleichzeitig umgehen. Wie bereits erläutert wurde, werden in dieser Arbeit multikriterielle Probleme in einkriterielle Probleme umgewandelt, indem ein Hauptkriterium festgelegt wird und alle anderen Kriterien als Nebenbedingung fungieren. Die Nebenbedingungen können dann als Penalty-Funktion in der Fitnessfunktion berücksichtigt werden.

Rechenzeit des speziellen Solvers

Optimierungen nichtlinearer Probleme sind generell sehr zeitintensiv, da einige hundert bis mehrere tausend Iterationsschritte nötig sind, für die der spezielle Solver jeweils einen Fitnesswert berechnen muss. Benötigt der spezielle Solver für die Berechnung oder Simulation eines Fitnesswertes schon mehrere Sekunden, kann die Iterationszeit des gesamten Optimierungsprozesses schnell auf mehrere Minuten oder Stunden anwachsen. Bei Optimierungsproblemen mit langen Rechenzeiten des speziellen Solvers sollte zunächst versucht werden, die Rechenzeit so weit wie möglich zu reduzieren. Ist das nur begrenzt möglich, ist es unter Umständen zielführend, einen Optimierungsalgorithmus zu wählen, der sich nur mit vergleichsweise wenigen Iterationen dem Optimum nähert.

2.4 Black-Box Optimierung

Von einer Black-Box Optimierung ist immer dann die Rede, wenn keine Informationen über die Eigenschaften der Fitnessfunktion $f(x)$ vorliegen und diese analytisch nicht zu beschreiben ist. Derartige Probleme werden in der Literatur auch ableitungsfreie Probleme genannt. Das Gegenteil ist die White-Box Optimierung von kontinuierlichen Problemen. Hier ist die Struktur von $f(x)$ bekannt und kann zur Optimierung benutzt werden. Der Gradient von $f(x)$ gibt dabei eine eindeutige Suchrichtung vor und über die Nullpunktbestimmung der partiellen Ableitung kann der Extremwert analytisch bestimmt werden.

Probleme im Ingenieurwesen fallen überwiegend in die Kategorie der Black-Box Probleme. Hinter der Berechnung der Fitnesswerte stehen meistens komplexe Simulationen oder Berechnungen, die auf nichtlinearen Werkstoffmodellen beruhen.

Die Bezeichnung Black-Box erfolgt aus der Sicht des Optimierungsalgorithmus, denn dieser soll den Extremwert einer Funktion finden, deren Eigenschaften er nicht kennt. Es können nur Informationen über die Eigenschaften der Fitnessfunktion gewonnen werden, indem die Fitnesswerte bestehender Lösungen ausgewertet werden. Das kann sich vorgestellt werden, als ob die Fitnessfunktion ein in Nebel gehüllter Berg ist, die Variablen sind die GPS-Koordinaten und der Fitnesswert entspricht der Höhe des Geländes. Nun ist es die Aufgabe, die Koordinaten des Berggipfels zu finden, wobei keine Information über die Höhe des Gipfels vorliegt. Zunächst wird an einer zufälligen Koordinate die Höhe des Geländes gemessen. Ob der gemessene Punkt nahe dem Gipfel, im Tal oder dazwischen liegt, kann nicht bestimmt werden. Alle Koordinaten zu messen, wäre zu zeitaufwändig und würde zu viele Ressourcen verbrauchen. Also werden nach einem bestimmten Muster weitere Koordinaten gemessen und anschließend die Höhen miteinander verglichen. So kann nach einer ausreichenden Anzahl an Messungen mit einer bestimmten Wahrscheinlichkeit festgestellt werden, in welchen Regionen der Landschaft sich Täler und in welcher sich Berge befinden. Ob sich unter den gemessenen Punkten der absolut höchste Gipfel befindet, kann jedoch nie mit hundertprozentiger Sicherheit bestimmt werden.

Alle Black-Box Optimierungen laufen nach diesem Schema ab. Die unterschiedlichen Optimierungsalgorithmen unterscheiden sich lediglich in der Methode und dem Muster, in dem die einzelnen Punkte auf der Landschaft untersucht werden. Black-Box Optimierungsalgorithmen lassen sich grundsätzlich in die drei folgenden Methoden einteilen: Downhill-Simplex-Methode, Ersatzmodell-Methode und Metaheuristik. Algorithmen der Downhill-Simplex-Methode und der Ersatzmodell-Methode können entweder für globale oder für lokale Suchen eingesetzt werden. Metaheuristische Algorithmen betreiben eine globale und lokale Suche. Sie starten global und schränken sich mit zunehmendem Optimierungsfortschritt auf ein lokales Gebiet ein. Alle drei Methoden werden im Folgenden vorgestellt.

Downhill-Simplex-Methode

Die Downhill-Simplex-Methode wird, angelehnt an den englischen Begriff, auch im deutschsprachigen Raum oft als Direct Search Methode bezeichnet. Die Downhill-Simplex-Methode ist die einfachste unter den drei oben genannten, weshalb eine große Anzahl an

Algorithmen auf dieser Methode basiert. Sie zeichnet aus, dass die Fitnesslandschaft nach einer deterministischen Methode untersucht wird. Das heißt, das Ergebnis einer Optimierung ist bei gleichem Startpunkt nicht nur reproduzierbar, sondern auch alle Zwischenschritte sind bei jedem Suchlauf identisch. Der Bergsteigeralgorithmus (Hillclimbing-Suchverfahren) und das Nelder-Mead-Verfahren sind die bekanntesten Algorithmen der Downhill-Simplex-Methode.

Ersatzmodell-Methode

Die Optimierung mit Hilfe der Ersatzmodell-Methode wird auch Sequentielle Optimierung genannt. Zu Beginn des Optimierungsprozesses werden einige zufällige Fitnesswerte berechnet, deren Zusammenhang der Algorithmus erkennt und daraus ein erstes Ersatzmodell des eigentlichen Optimierungsproblems erstellt. Die Implementierung eines geeigneten Ersatzmodells ist dabei rechnerisch sehr aufwendig und zeitintensiv.

Das Ersatzmodell besteht aus approximierten, differenzierbaren Funktionen, die das Verhalten der Fitnessfunktion so nah wie möglich imitieren. In jedem Iterationsschritt werden vielversprechende Lösungen am Ersatzmodell ermittelt und mit der Fitnessfunktion überprüft. Anhand der so ermittelten echten Funktionswerte wird das Ersatzmodell angepasst und verfeinert. Mit jedem Iterationsschritt verbessert sich das Ersatzmodell, was zu einer stetigen Verbesserung des Fitnesswertes führt.

Metaheuristik

Als Metaheuristik werden Algorithmen bezeichnet, die eine Problemstellung nicht exakt, sondern nur näherungsweise lösen. Im Vergleich zu der Downhill-Simplex-Methode und der Ersatzmodell-Methode basieren die metaheuristischen Optimierungsverfahren auf keinem mathematischen Konvergenzbeweis, sondern versuchen einen natürlichen Prozess so gut wie möglich nachzuahmen. Unter allen Optimierungsmethoden ist die Metaheuristik die populärste und meistverbreitete, da sie einfach zu implementieren ist und immer, unabhängig von der Art des Optimierungsproblems eine Lösung liefert. Unter Mathematikern und Informatikern ist die Metaheuristik allerdings als „method of last resort“ [3] bekannt, da keinesfalls garantiert ist, dass die gefundene Lösung einem Extremwert der Problemstellung entspricht. Benchmark Tests mit bekannten Testfunktionen zeigen, dass metaheuristische Optimierungsverfahren generell langsamer sind und eine schlechtere Performance abliefern als klassische Optimierungsverfahren. Hier gilt das No-Free-Lunch-Theorem [4], nachdem

kein allgemeines metaheuristisches Optimierungsverfahren besser ist, als alle anderen speziellen Optimierungsverfahren.

Trotz der hohen Ungenauigkeit und der schlechten Performance haben metaheuristische Optimierungsverfahren ihre Berechtigung. Wie bereits erwähnt, bieten sie sich durch ihre einfache Implementierung und die Fähigkeit, beliebig komplexe Probleme zu optimieren, an. Ein weiterer großer Vorteil der Metaheuristiken ist, dass sie sowohl eine globale als auch lokale Suche durchführen können. Somit müssen, im Gegensatz zur Optimierung mit Algorithmen der Ersatzmodell-Methode und Downhill-Simplex-Methode, keine Informationen über die Struktur des Problems bekannt sein.

Aufgrund der großen Popularität dieser Methode gibt es mittlerweile eine immense Anzahl an metaheuristischen Algorithmen. Zu den bekanntesten zählen der evolutionäre Algorithmus, die Partikelschwarmoptimierung und die simulierte Abkühlung. In Bibliotheken findet man zu fast jedem Prozess in der Natur einen entsprechenden Optimierungsalgorithmus, wie beispielsweise den Ameisenalgorithmus, den Bienen-Algorithmus oder das künstliche Immunsystem.

2.5 Software

Rhinoceros 3D

Die 3D Modellierungssoftware Rhinoceros, oft auch nur Rhino genannt, wurde von Robert McNeel & Associates entwickelt und basiert auf der Modellierung mit nicht-uniformen rationalen B-Splines, kurz NURBS-Modellierung. NURBS sind sehr genaue mathematische Nachbildungen beliebig komplexer Freiformen. Die einzige Einschränkung ist, dass alle NURBS-Geometrien durch nicht-verzweigende, stetige Linienzüge darstellbar sein müssen. Rhino wird sowohl im Ingenieurwesen als auch in der Architektur zur Darstellung von Freiformen und für das Rapid Prototyping benutzt. Mit Rhino lassen sich über vierzig Dateiformate bearbeiten, sowie über hundert Anwendungen von Drittanbietern verwenden.

Rhino ist eine nicht-parametrische Modellierungssoftware und bietet nur ein sehr kurzes, implizites Ereignisprotokoll beziehungsweise eine kurze Eingabehistorie. Das bedeutet, es kann nur eine geringe Anzahl an Eingabeschritten nachvollzogen oder rückgängig gemacht werden. Eine explizite Eingabehistorie bietet die Plug-In-Anwendung Grasshopper 3D für Rhino. Im Grunde kann mit Grasshopper eine Historienabhängigkeit in einem baumähnlichen

Datenflussdiagramm vom Benutzer erstellt werden, welches in der Rhino-Oberfläche dargestellt wird. [5]

Grasshopper 3D

Grasshopper 3D (GH) ist ein von David Rutten zusammen mit McNeel & Associates entwickelts Plug-In für Rhinoceros 3D. Die Anwendung ist frei verfügbar und lässt sich mit jeder gültigen Rhino Lizenz ohne Einschränkungen benutzen. GH ist eine grafische Programmierumgebung, für die grundsätzlich keine Programmierkenntnisse im Sinne von Code schreiben erforderlich sind. Programme werden erstellt, indem Komponenten auf die Leinwand (Canvas) gezogen werden und der Ausgang einer Komponente mit dem Eingang einer nachfolgenden Komponente durch ein Kabel verbunden wird. Die mit den Kabeln verbundenen Komponenten repräsentieren den Datenfluss des Programms. Diese grafische Art zu programmieren ist im Vergleich zu klassischen Programmiersprachen einfach und intuitiv anwendbar. Gleichzeitig ist dies aber auch der größte Nachteil von GH, da nur eine sequentielle Verarbeitung der Komponenten möglich ist. Programmschleifen oder Rechenwiederholungen sind nicht ohne weiteres möglich.

Grasshopper benutzt zur Datenverarbeitung Listen, die zu mehrdimensionalen Listen zusammengefasst werden können. In GH wird eine Liste in einer mehrdimensionalen Liste als Ast in einem Baum bezeichnet. In klassischen Programmiersprachen ist diese Art der Datenverarbeitung als mehrdimensionale Arrays bekannt.

Mit den meisten Standardkomponenten in GH können Geometrien erstellt und bearbeitet werden. Die Möglichkeiten reichen von einfachen Komponenten wie „*Erstelle eine Linie aus zwei Punkten*“ bis hin zu „*Erstelle einen 3D Körper aus einer Punktewolke*“. Wie für Rhino gibt es auch für GH eine Vielzahl an Plug-Ins, wodurch die Komponentenpalette fasst unbegrenzt erweiterbar ist. Für diese Arbeit wurden hauptsächlich die Plug-Ins *Karamba3D* zur statischen Berechnung und *Self-Organising-Map-Component* zum Erstellen von selbstorganisierenden Karten verwendet. Des Weiteren wurden die Plug-Ins *Galapagos, Goat, Silvereye und Opossum* zur Implementierung von Optimierungsalgorithmen verwendet.

Abbildung 9 zeigt auf der linken Seite das Rhino Interface und auf der rechten Seite die Grasshopper Leinwand. In dem Beispiel wurde eine parametrisierte Linie erstellt, welche durch die X-, Y- und Z-Koordinaten des Start- und Endpunktes definiert ist. Das Programm beginnt mit der Eingabe der kartesischen Koordinaten mit Hilfe der Komponenten *Number*

Slider. Diese sind durch Kabel mit den Eingängen der *Construct Point* Komponenten verbunden, deren Ausgänge wiederrum mit den Eingängen der Komponente *Line* verbunden sind. Durch die Manipulation der *Number Slider* kann die Geometrie der Linie verändert werden. Die Definition der Linie ist ausschließlich auf der Grasshopper Leinwand zu sehen, während die Linie als Live-Vorschaubild in Rhino angezeigt wird. Rhino dient in diesem Beispiel als Vorschaufenster. Es kann aber auch von GH auf Punkte, Linien, Flächen und Körper in Rhino referenziert werden. Für das untenstehende Beispiel könnten also auch Punkte in Rhino gezeichnet werden, die dann mit GH durch eine Linie verbunden werden. Andersrum können auch mit GH erzeugte Geometrien nach Rhino exportiert werden und als Standard DXF- oder DWG-Datei abgespeichert werden. So lassen sich mit GH erzeugte Geometrien in beliebigen CAD-Anwendungen weiterbearbeiten.

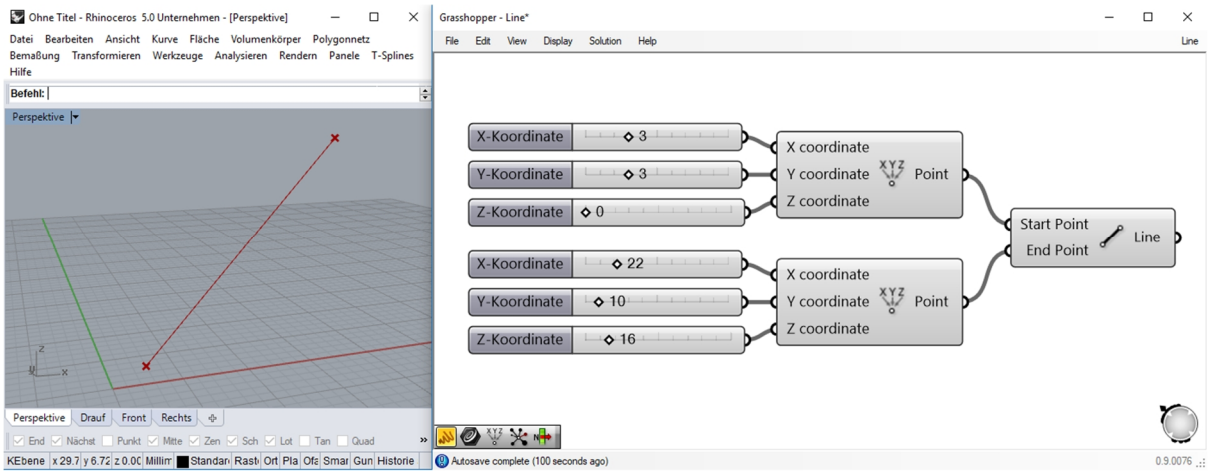


Abbildung 9: Parametrische Geometrie: Definiert in GH und angezeigt in Rhino

Nach dem dargestellten Prinzip lassen sich beliebig komplexe Geometrien erzeugen oder auch weiterbearbeiten. Aus der oben erzeugten Linie kann beispielsweise ein Biegebalken oder Fachwerkstab generiert werden, der wiederum mit dem bereits erwähnten Plug-In *Karamba 3D* statisch analysiert werden kann.

Für Ingenieure und Architekten ist die Modellierungssoftware Rhinoceros in Verbindung mit Grasshopper und den entsprechenden Plug-Ins ein leistungsstarkes Werkzeug, um Konstruktionen zu parametrisieren, zu analysieren und zu optimieren.

Karamba 3D

Karamba ist ein kostenpflichtiges Plug-In für Grasshopper, mit dem sich parametrisch konstruierte Balken, Fachwerke, Rahmen und Schalen statisch analysieren lassen. Karamba ist in der Lage, dreidimensionale Konstruktionen mit der Finite-Element-Methode zu berechnen und Größen wie Auflagerkräfte, Schnittgrößen, Querschnittswerte und Verformungswerte direkt in Grasshopper auszugeben. Nach der Berechnung können die entsprechenden Größen wie gewohnt in GH weiterverwendet werden. Auf Grundlage der Verformung können beispielsweise parametrisierte Querschnittsabmessungen optimiert werden.

In folgender Abbildung ist die Grasshopper Definition für eine Kragarmträger Berechnung durch Karamba dargestellt. Es ist zu sehen, wie wenige Komponenten benötigt werden, um Informationen über die Querschnittswerte, Schnittkräfte, Verformung und Ausnutzung der Konstruktion zu erhalten. Durch die Parametrisierung können in kurzer Zeit die Parameter Spannweite, Lagerbedingung, Lastart, Lastgröße und Querschnittsabmessung geändert werden.

Live-Visualisierung in Rhino **Grasshopper Definition**

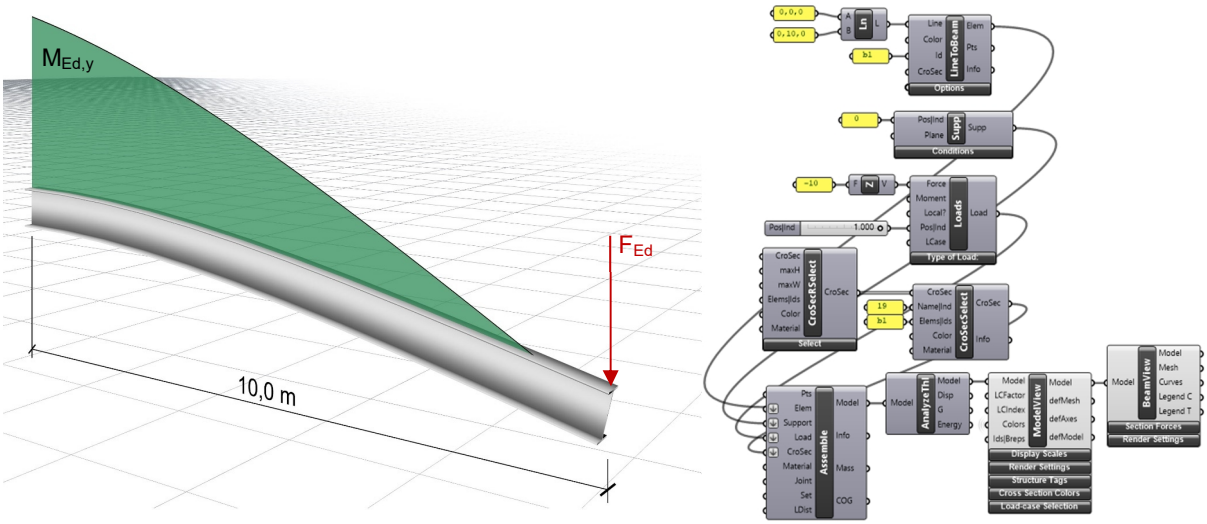


Abbildung 10: Berechnung eines Kragarmträgers mit Grasshopper und Karamba

Für immer wiederkehrende Systeme bedeutet die Möglichkeit parametrisierte Modelle berechnen zu können eine enorme Erleichterung. Änderungen am System können durch den Tragwerksplaner schnell vorgenommen und analysiert werden. Auch in Entwurfsphasen bieten parametrisierte Modelle Vorteile. Ist das Modell einmal in Grasshopper implementiert, können mehrere Varianten oder kurzfristige Änderungen schnell untersucht werden. Falls

erforderlich, können im Zuge der Bemessung Optimierungsprozesse angewendet werden, um Systeme effizienter und nachhaltiger zu gestalten.

2.6 Verwendete Plug-Ins und Optimierungsalgorithmen

In freien Onlinebibliotheken ist eine immense Anzahl an Black-Box Optimierungsalgorithmen zu finden, die für die unterschiedlichsten Aufgaben und Verwendungen entworfen wurden. Bei der Auswahl geeigneter Algorithmen wurde darauf Wert gelegt, dass möglichst alle Methoden und Eigenschaften zwischen denen unterschieden werden kann, vertreten sind. Für die Implementierung von Optimierungsalgorithmen in die Grasshopper Entwicklungsumgebung werden Plug-Ins benötigt, die den entsprechenden Algorithmus in Form einer DLL-Datei auf dem Computer installieren oder auf eine Onlinebibliothek wie beispielsweise NLopt [7] zugreifen.

In dieser Arbeit kommen vier Plug-Ins und insgesamt sechs Algorithmen zum Einsatz:

Galapagos

Galapagos ist seit der Grasshopper Version 0.7.0053 fest in Grasshopper implementiert und muss nicht zusätzlich installiert werden. Galapagos richtet sich mit der einfachen Bedienung und Handhabung genauso wie Grasshopper an alle Nicht-Programmierer, und ist deshalb ein populäres Optimierungstool unter Ingenieuren und Architekten. Entwickelt wurde Galapagos von David Rutten, dem Programmierer von Grasshopper. Nach seiner Aussage entstand Galapagos aus einer nicht wissenschaftlichen und nicht professionellen Motivation. Vermutlich existiert aus diesem Grund auch keine Dokumentation über die genaue Arbeitsweise der Algorithmen. Auch Benchmark Tests mit bekannten Optimierungsproblemen sind in der Literatur nicht zu finden.

Galapagos liefert mit dem *Genetischen Algorithmus* (GA) und der *Simulated Annealing* (SA, simulierte Abkühlung) gleich zwei metaheuristische Optimierungsalgorithmen. Der GA verwendet zur Zielwertsuche, anders als die meisten Optimierungsalgorithmen, eine ganze Population anstelle eines einzelnen Suchpunktes. Der Ablauf eines GA erfolgt unabhängig vom tatsächlich verwendeten Algorithmus immer nach dem gleichen Prinzip und lässt sich in vier Schritte unterteilen:

Generation $i = 1$

1. Schritt Erzeugen der Startpopulation P_1

wiederhole

2. Schritt Bestimmung der erolgreichsten Individuen $S_i \in P_i$

3. Schritt Mutation der Individuen S_i

4. Schritt Erzeugen einer neuen Population P_{i+1} aus P_i und S_i

Generation $i = i + 1$

bis Abbruchbedingung erfüllt

Abbildung 11: Pseudo genetischer Algorithmus

Die Verwendung einer Population hat den Vorteil, dass gleichzeitig mehrere vielversprechende Regionen untersucht werden können. Dadurch wird sowohl eine lokale als auch eine globale Suche unternommen. Wird kein Zeitlimit festgelegt, minimiert sich die Gefahr, dass der Algorithmus frühzeitig an einem lokalen Optimum hängen bleibt. Ein Nachteil ist allerdings der hohe Rechenaufwand, da für jeden Iterationsschritt die Fitnesswerte der gesamten Population ermittelt werden.

Im Gegensatz zum GA ist die simulierte Abkühlung (SA) ein Einzellösungsverfahren und verwendet keine Population für die Suche des Optimums. Die Struktur des SA Algorithmus basiert grundsätzlich auf einer lokalen Suche. Da mit einer geringen Wahrscheinlichkeit auch schlechte Lösungen akzeptiert werden, kann der Algorithmus aus der lokalen Region entkommen und ein globales Optimum finden.

Die Methode der simulierten Abkühlung stammt ursprünglich aus dem Bereich der Thermodynamik. Als Grundlage für einen metaheuristischen Algorithmus dient sie erst seit den 1980er Jahren. Die Grundidee orientiert sich an dem Abkühlvorgang von Feststoffen in der Physik. Werden Festkörper zum Schmelzen gebracht und anschließend langsam abgekühlt, haben die Atome genug Zeit, sich in einer thermisch günstigen Struktur anzuordnen. So existiert für jede Temperatur ein thermisches Gleichgewicht. Übertragen auf den Optimierungsprozess bedeutet das, dass zunächst mit einem zufälligen Fitnesswert begonnen wird. Anschließend wird ein weiterer Fitnesswert in der Nachbarschaft berechnet. Ist dieser Fitnesswert besser, wird er als neue Lösung akzeptiert. Ist er schlechter, wird er nur mit einer gewissen Wahrscheinlichkeit akzeptiert. Die Wahrscheinlichkeit wird über den Faktor T, die Temperatur bestimmt. Jeder Iterationsschritt bedeutet eine Abkühlung des Systems und damit eine geringere Wahrscheinlichkeit, dass schlechte Lösungen akzeptiert werden.

Goat

Goat ist ein weiteres Plug-In und Optimierungstool für Grasshopper. Goat wurde auf der Grundlage von Galapagos entwickelt, bietet jedoch insgesamt fünf Optimierungsalgorithmen aus der NLopt Onlinebibliothek, die alle nach der Downhill-Simplex-Methode (Direct Search Methode) arbeiten. Leider verfügt Goat über keine grafische Benutzeroberfläche, die den Benutzer den Optimierungsprozess beobachten lässt. Drei der zur Verfügung stehenden Algorithmen sind für eine lokale Suche geeignet, zwei für eine globale. Für diese Arbeit wird der *DIRECT Algorithmus* und der *Sbplx Algorithmus* verwendet.

Auch für Goat existiert keine vollständige Dokumentation und Informationen über die Algorithmen sind kaum vorhanden. Aus diesem Grund wird im Folgenden nur die prinzipielle Arbeitsweise des *DIRECT* und *Sbplx Algorithmus* vorgestellt. Wie genau Goat diese Algorithmen implementiert, kann nicht gesagt werden.

DIRECT ist die Kurzform für „DIviding RECTangles“ und beschreibt das Prinzip des Algorithmus. Die Grundidee ist, das Optimierungsproblem in immer kleinere Rechtecke aufzuteilen, bis die Problemgröße so reduziert wurde, dass es direkt gelöst werden kann. *DIRECT* ist ein globaler Optimierungsalgorithmus, wobei die Maschenweite, also die Größe der Rechtecke, an jedes Optimierungsproblem angepasst werden muss. Ist die Maschenweite zu grob, können dazwischenliegende Optima nicht erkannt werden, ist sie zu eng, steigt die benötigte Rechenzeit enorm an. Dieser Effekt verstärkt sich bei Problemen höherer Dimensionalität. Deshalb wird der *DIRECT Algorithmus* vorwiegend für einkriterielle, nichtlineare Probleme niedrigerer Dimension verwendet. [8]

Anders als *DIRECT* betreibt der *Sbplx* Algorithmus eine lokale Extremwertsuche, wofür das Nelder–Mead–Verfahren angewendet wird. Bei diesem Verfahren wird zunächst eine $n+1$ dimensionale Simplex gebildet, wobei n die Dimensionalität des Problems ist. Als Simplex wird in der Geometrie ein räumliches Polygon in beliebiger Dimension bezeichnet. Die Grundidee ist, dass an den Eckpunkten des Simplex die Fitnesswerte ermittelt werden. Anschließend wird der Wert eines weiteren Punktes innerhalb des Simplex ermittelt. Ist dieser besser als der schlechteste Punkt des Simplex, werden die entsprechenden Punkte ausgetauscht und das Simplex verkleinert sich. Durch die Reduktion des Durchmessers nähert sich der Mittelpunkt des Simplex mit jedem Iterationsschritt an das Optimum an. [9]

Silvereye

Silvereye implementiert als erstes Grasshopper Plug-In die Partikelschwarmoptimierung (PSO). PSO gehört wie GA und SA zu den metaheuristischen Optimierungsmethoden. Trotzdem beschreiben die Entwickler von Silvereye PSO im Vergleich zu GA und SA als einen „schnelleren und intuitiveren“ Algorithmus [10]. Wie der Name schon sagt, wird durch den Algorithmus das natürliche Schwarmverhalten von Tieren nachgeahmt. Es wird angenommen, dass das Verhalten einzelner Partikel in einem Schwarm gewissen Gesetzmäßigkeiten folgt. Diese Gesetzmäßigkeiten dienen als Anpassungsfaktoren der Partikel im Suchraum. So besitzt ein Partikel im Suchraum Kenntnis über folgende Eigenschaften:

- Geschwindigkeitsvektor
- Bester eigener Wert und dessen Ort
- Ort und Wert des besten Partikels

Zu Beginn werden alle Partikel über den n-dimensionalen Suchraum gleichmäßig verteilt und bekommen einen initialen Geschwindigkeitsvektor zugewiesen. Bei jedem Iterationsschritt werden die Fitnesswerte jedes Partikels berechnet und der Geschwindigkeitsvektor angepasst, wobei sich die Anpassung des Geschwindigkeitsvektors aus dem Ort der global besten Lösung und dem Ort der eigenen besten Lösung zusammensetzt. Zusätzlich wird der angepasste Geschwindigkeitsvektor mit einem Zufallsfaktor manipuliert. Das Ziel ist es, alle Partikel in Richtung der besten Lösung zu führen. [10]

Opossum

Mit dem Plug-In Opossum lassen sich Optimierungen mit der Ersatzmodell-Methode durchführen. Eine Erklärung der Methode ist in Absatz 2.4 zu finden. Opossum wurde an der Universität für Technik und Design in Singapore speziell für Nicht-Programmierer entwickelt und zeichnet sich durch die „einfache und intuitive“ Handhabung aus. [11] Wie schon bei Galapagos und Goat, scheinen die Entwickler von benutzerfreundlichen Programmen keinen Bedarf darin zu sehen, ihre Arbeit zu dokumentieren. Über die tatsächlich verwendeten Algorithmen ist deshalb nur bekannt, dass sowohl der *Gutmann Algorithmus* als auch der *MSRSM Algorithmus* zur Anwendung kommen. Beide Algorithmen werden in Opossum als *RBFOpt* zusammengefasst. Sie basieren auf der radialen Basisfunktion und sind der Ersatzmodell-Methode zuzuordnen. *Gutmann* sucht den Punkt an der Stelle der größten Krümmung im Ersatzmodell, unter der

Annahme, dass durch diesen Punkt die Genauigkeit des Ersatzmodells verbessert werden kann. *MSRSM* durchsucht das Ersatzmodell nach Punkten, die aus dem Modell in irgendeiner Art und Weise herausstechen, um diese so zu verändern, dass sich die Balance des Modells verbessert. Für diese Suche können genetische Algorithmen, zufällige Stichproben oder mathematische Solver verwendet werden. Durch das Ausbalancieren verspricht man sich eine Annährung des Ersatzmodells an die tatsächliche Fitnessfunktion.

Allerdings lässt die Benutzeroberfläche von Opossum den Anwender nicht direkt zwischen den zwei Algorithmen auswählen, sondern stellt drei Auswahlmöglichkeiten zur Verfügung: Fast Search, Extensive Search und Alternative Search. Hinter der Einstellung „Fast Search“ steckt der MSRSM Algorithmus mit der Verwendung eines genetischen Algorithmus zur Suche der Punkte im Modell. „Extensive Search“ benutzt die gleichen Algorithmen, gibt dem genetischen Algorithmus jedoch mehr Zeit bei der Suche. „Alternative Search“ verwendet den *Gutmann Algorithmus*. Wenn im Rahmen dieser Arbeit auf den *RBFOpt Algorithmus* verwiesen wird, ist immer die Einstellung „Fast Search“ gemeint.

Für eine detaillierte Beschreibung des *RBFOpt Algorithmus* wird auf den Aufsatz „RBFOpt: an open-source library for black-box optimization with costly function evaluations“ von Costa und Nannicini verwiesen. [12]

Eine Zusammenfassung aller verwendeten Algorithmen ist in Kapitel 7 *Benchmark Tests* zu finden.

3 Visualisierung von Optimierungsproblemen

3.1 Grundlagen

Die Visualisierung von n-dimensionalen Datenstrukturen ist eine sehr effektive Methode, um Einblicke in die komplexen Zusammenhänge innerhalb des Datenraums zu erlangen. Das übergeordnete Ziel der Visualisierung ist es, große, n-dimensionale Daten in einen vorstellbaren und visualisierbaren 2-D oder 3-D Raum zu übersetzten. Für diese Aufgabe gibt es eine Vielzahl an Techniken und Methoden, wobei die meisten den gleichen Ansatz verfolgen. Die Dimensionen der Inputdaten werden soweit reduziert, bis sie in einem 2-D oder 3-D Raum dargestellt werden können. Der kritischste Schritt dabei ist es, die Eigenschaften der Ursprungsdaten so vollständig wie möglich zu erhalten. Dennoch gehen bei allen Visualisierungstechniken, die auf einer Dimensionsreduktion basieren, auf die eine oder andere Weise Daten verloren. Oft ist es die ursprüngliche Bedeutung oder der Datenzusammenhang, der nach der Dimensionsreduktion nicht mehr zu erkennen ist. Die Visualisierungstechniken unterscheiden sich auch in der Art der Darstellung. Oft werden unterschiedliche Farben und Formen, sowie kartesische, kreisförmige oder parallele Koordinatenachsen verwendet. Je nach Datentyp und Ziel der Visualisierung eignen sich unterschiedliche Darstellungsarten.

Spricht man von Visualisierung im Zusammenhang mit Optimierungsproblemen, ist zunächst zwischen der Visualisierung von Fitnesswerten und Fitnesslandschaften zu unterscheiden. Die Visualisierung der Fitnesswerte kann als eine Darstellung der Optimierungsergebnisse verstanden werden. Hierbei werden im Anschluss an den Optimierungsprozess die Fitnesswerte im Bezug zueinander dargestellt. Besitzt ein Optimierungsproblem keine eindeutige beste Lösung, ist die Visualisierung ein Tool zur Wahl einer geeigneten Lösung, siehe Absatz 3.2.

Bei der Visualisierung von Fitnesslandschaften geht es weniger um das Ergebnis einer Optimierung, sondern eher um die Art und Eigenschaft des Problems. Anhand der Fitnesslandschaft können Erkenntnisse gewonnen werden, die bei der Beurteilung und Einordnung des Problems und der Ergebnisse helfen. Durch den räumlichen Bezug zwischen Variablen und Fitnesswert kann auch eine Aussage über die Qualität der gewählten Fitnessfunktion beziehungsweise Penalty-Funktionen getroffen werden. Zur Visualisierung von Fitnesslandschaften siehe folgenden Absatz 3.2.

3.2 Visualisierung der Fitnesswerte

Bei der Visualisierung von Fitnesswerten wird oft von der Darstellung der Pareto-Front gesprochen. Diese Visualisierungsart ist eine recht einfache Darstellung der Optimierungsergebnisse in Abhängigkeit der Optimierungsziele. Die Darstellung der Pareto-Front ist jedoch nur dann sinnvoll und auch nur dann möglich, wenn das Optimierungsproblem zwei gegensätzliche Ziele hat. Soll zum Beispiel der Kreisquerschnitt eines Kragarms hinsichtlich des Eigengewichts und der Verformung optimiert werden, sind diese beiden Ziele gegensätzlich. Ein kleinerer Querschnittsdurchmesser resultiert in einem geringeren Eigengewicht, jedoch erhöht sich die Verformung. Soll der gleiche Kragarm hinsichtlich der Querschnittsspannung und der Verformung optimiert werden, sind die Ziele nicht gegensätzlich. Bei einem möglichst großen Querschnitt sind Spannung und Verformung am geringsten.

Bei Optimierungsproblemen mit gegensätzlichen Optimierungszielen, gibt es nicht immer die eine beste Lösung, sondern oft mehrere zulässige Lösungen. Existieren mehrere Lösungen zu einem Problem, werden diese Lösungen als pareto-optimal bezeichnet, wobei alle pareto-optimalen Lösungen zusammen die Pareto-Front bilden. Die Lösungen auf dieser Front befinden sich in einem Zustand, in dem eines der Optimierungsziele nicht verbessert werden kann, ohne das andere Ziel gleichzeitig zu verschlechtern. Anhand folgendem, in der Literatur oft verwendeten Beispiel wird das Konzept der Pareto-Front Darstellung veranschaulicht.

Optimiert werden soll ein Kragträger, welcher am freien Ende mit P belastet wird. Variabel ist die Kragträgerlänge l und der Querschnittsdurchmesser d, siehe Abbildung 12. Gesucht wird die Kombination aus Länge und Querschnittsdurchmesser, bei dem der Kragträger das geringste Eigengewicht besitzt und gleichzeitig die geringste Endverformung aufweist.

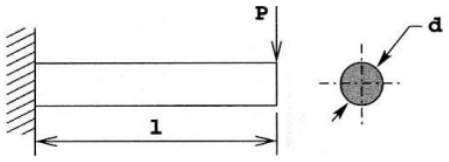


Abbildung 12: System des Kragträgers [1]

Es ist offensichtlich, dass beide Optimierungsziele, Verformung und Gewicht, gegensätzlich sind. Ein größerer Durchmesser resultiert in einer geringeren Verformung, erhöht aber das Eigengewicht, wobei ein kleinerer Durchmesser das Gegenteil bewirkt. Um realistische Optimierungsergebnisse zu erhalten, werden die zulässigen Variablen auf $10 \leq d \leq 50\,mm$ und $200 \leq l \leq 1000\,mm$ begrenzt. Gleichzeitig wird die maximale

Querschnittsspannung auf $\sigma_{max} = 300\ N/mm^2$ und die maximale Endverformung auf $w_{max} = 5\ mm$ begrenzt. Die zur Optimierung zugelassenen Variablen bilden den zulässigen Variablenraum. Das linke Diagramm der folgenden Abbildung 13 zeigt den rechteckigen zulässigen Variablenraum und den daraus gebildeten möglichen Variablenraum. Der mögliche Variablenraum kann jedoch erst nach der Optimierung definiert werden, da dieser die Variablenkombinationen abbildet, die den Spannungs- und Verformungsgrenzwert einhalten. Ein Kragträger mit einem Querschnittsdurchmesser von $d = 25\ mm$ und einer Länge von $l = 600\ mm$ liegt damit oberhalb des Grenzwertes σ_{max} und / oder w_{max}. Jede Variablenkombination im möglichen Variablenraum kann einem Ergebnis im Ergebnisraum zugeordnet werden. Allerdings erfolgt die Zuordnung rein rechnerisch, es ist nicht möglich einem beliebigen Punkt im Variablenraum einen Punkt im Ergebnisraum grafisch zuzuordnen.

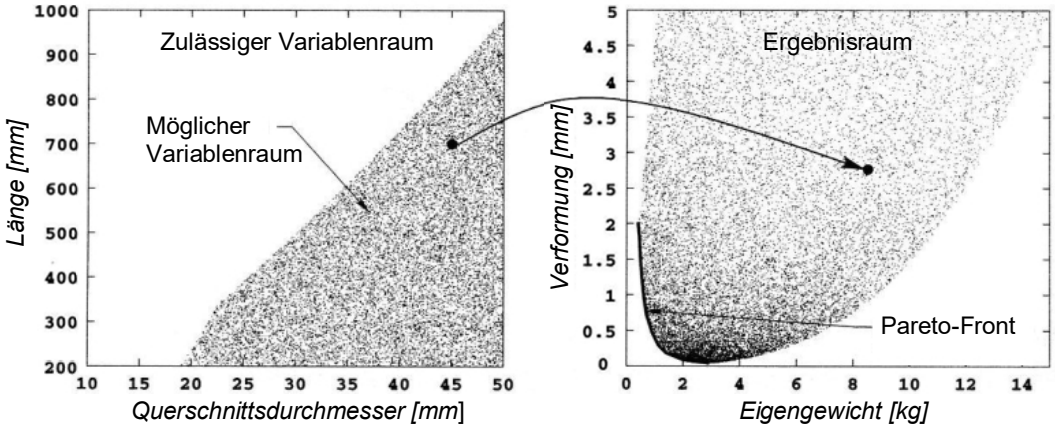


Abbildung 13: Zulässiger Variablenraum (links) und Ergebnisraum (rechts) [1]

Das rechte Diagramm stellt den Ergebnisraum der Optimierung dar, was den Fitnesswerten beider Optimierungsziele entspricht. Ziel der Optimierung war es, den Querschnitts-durchmesser und die Länge zu finden, bei der das Eigengewicht und die Endverformung des Kragträgers am kleinsten sind. Der Ergebnisraum zeigt, dass es nicht die eine optimale Lösung gibt. Die Pareto-Front markiert die Ergebnisse, die im Bezug zueinander jeweils in einem Ziel besser sind, dafür in dem anderen schlechter. Im Vergleich zu allen Ergebnissen, die nicht auf der Pareto-Front liegen, sind sie in beiden Zielen besser. Für ein besseres Verständnis ist in Abbildung 14 ein Ausschnitt der Pareto-Front mit fünf Optimierungsergebnissen, A bis E, dargestellt. Es ist offensichtlich, dass Ergebnis A mit dem geringsten Querschnittsdurchmesser auch das geringste Eigengewicht aufweist, dafür aber die größte Endverformung. Bei Ergebnis D verhält es sich umgekehrt. Dennoch kann unter Kenntnis von Ergebnis A und D keine optimale Lösung in Bezug auf beide Ziele festgelegt werden, das heißt, beide Ergebnisse sind als gleichwertig anzusehen. Analog zu Ergebnis A

und D verhält es sich mit allen anderen Ergebnissen auf der Pareto-Front. Betrachtet man ausschließlich Ergebnis E und D kann angenommen werden, dass beide Ergebnisse auf der Pareto-Front liegen. Ist zusätzlich zu E und D Ergebnis C bekannt, gilt: C ist in beiden Zielen besser als E und D ist in einem Ziel besser als C. Somit liegt Ergebnis E nicht auf der Pareto-Front und ist als Lösung des Optimierungsproblems nicht relevant.

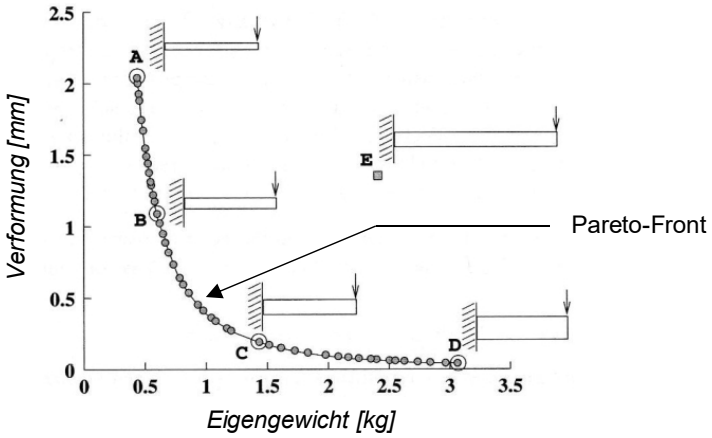


Abbildung 14: Pareto Front [1]

Allgemein gilt, mindestens ein Ergebnis auf der Pareto-Front ist in beiden Zielen besser als jedes nicht optimale Ergebnis.

Zusammenfassung

Die Darstellung der Pareto-Front ist ein nützliches Tool, um im Anschluss an einen Optimierungsprozess eine geeignete Lösung auszuwählen. Jedoch ist diese Darstellung nur begrenzt anwendbar. Wird nur ein Ziel formuliert, existiert keine Pareto-Front. Bei zwei Zielen wird die Pareto-Front durch eine Kurve und bei drei Zielen durch eine Fläche dargestellt. Bei mehr als drei Zielen können die Ergebnisse auf der Pareto-Fläche zusätzlich eingefärbt werden, allerdings leidet darunter die Lesbarkeit.

Des Weiteren liefert diese Darstellungsart keinerlei Informationen über das Optimierungsproblem oder die Fitnesslandschaft an sich. Es kann keine Aussage darüber getroffen werden, ob das Problem einen globalen oder mehrere lokale Extremwerte besitzt. Zudem wird in der Darstellung der Pareto-Front der Effekt einer Penalty-Funktion nicht sichtbar und ein Zusammenhang zwischen Variablen und Fitnesswert kann nur durch eine Rückrechnung der Ergebnisse hergestellt werden.

Um die Qualität und Eignung eines Black-Box Algorithmus bezüglich einer Problemkategorie beurteilen zu können, sind diese Informationen allerdings unerlässlich. Das reine Ergebnis

der Optimierung rückt dabei sogar in den Hintergrund, weshalb auf die Darstellung der Pareto-Front bei der Auswertung von Benchmark Tests verzichtet wird.

3.3 Visualisierung von mehrdimensionalen Daten

Im Folgenden wird gezeigt, nach welchem Prinzip mehrdimensionale Daten dargestellt werden können. Grundsätzlich stehen hierfür mehrere Methoden zur Verfügung, die meistens aus dem Bereich der Ökonomie stammen. In dieser Arbeit wird nur die Darstellung von Daten in einer 3-D Landschaft beschrieben, Darstellungsarten wie der Hypercube, Parallel-Koordinaten oder Netzwerke werden nicht weiter betrachtet.

Die relevanten Daten für eine Fitnesslandschaft bestehen aus den Variablensets der Optimierung und den daraus resultierenden Fitnesswerten. Ein n-dimensionales Optimierungsproblem besitzt also eine n+1-dimensionale Fitnesslandschaft. Um eine klassische 3-dimnesionale Karte der n+1-dimensionalen Landschaft zu erstellen, muss die Dimensionalität zwangsläufig reduziert werden. Dies wird erreicht, indem die drei Achsen des kartesischen Koordinatensystems in eine achsenlose Ebene und eine Höhenachse aufgeteilt werden. Unabhängig von der Dimensionalität werden alle Variablensets auf die achsenlose Ebene projiziert und nur die Fitnesswerte werden auf der Höhenachse dargestellt.

Das Visualisierungsprinzip wird anhand der Box-Ballon Metapher im Folgenden erläutert:

Es soll eine Fitnesslandschaft eines dreidimensionalen Optimierungsproblems mit den Variablen a_n, b_n und c_n und dem Fitnesswert $f(a_n, b_n, c_n)$ erstellt werden. Das n-te vierdimensionale Datenset ist somit $[a_n; b_n; c_n; f(a_n, b_n, c_n)]$. Die Variablen werden zunächst als Punkt auf die achsenlose Ebene projiziert. Der projizierte Punkt kann als Box mit drei Eigenschaften verstanden werden, wobei diese verschlossen ist und somit die Eigenschaften von außen nicht sichtbar sind. Es ist zwar möglich, die Box zu öffnen und sich die darin enthaltenen Eigenschaften anzuschauen, jedoch können nicht alle Boxen gleichzeitig, sondern jeweils nur eine geöffnet werden. Anders verhält es sich mit dem Fitnesswert. Dieser ist als ein mit Helium gefüllter Ballon zu verstehen, der mit einer Schnur an der Box befestigt ist, wobei die Länge der Schnur und somit die Höhe des Ballons den Absolutwert des Fitnesswerts bestimmt. Siehe Abbildung 15 auf folgender Seite.

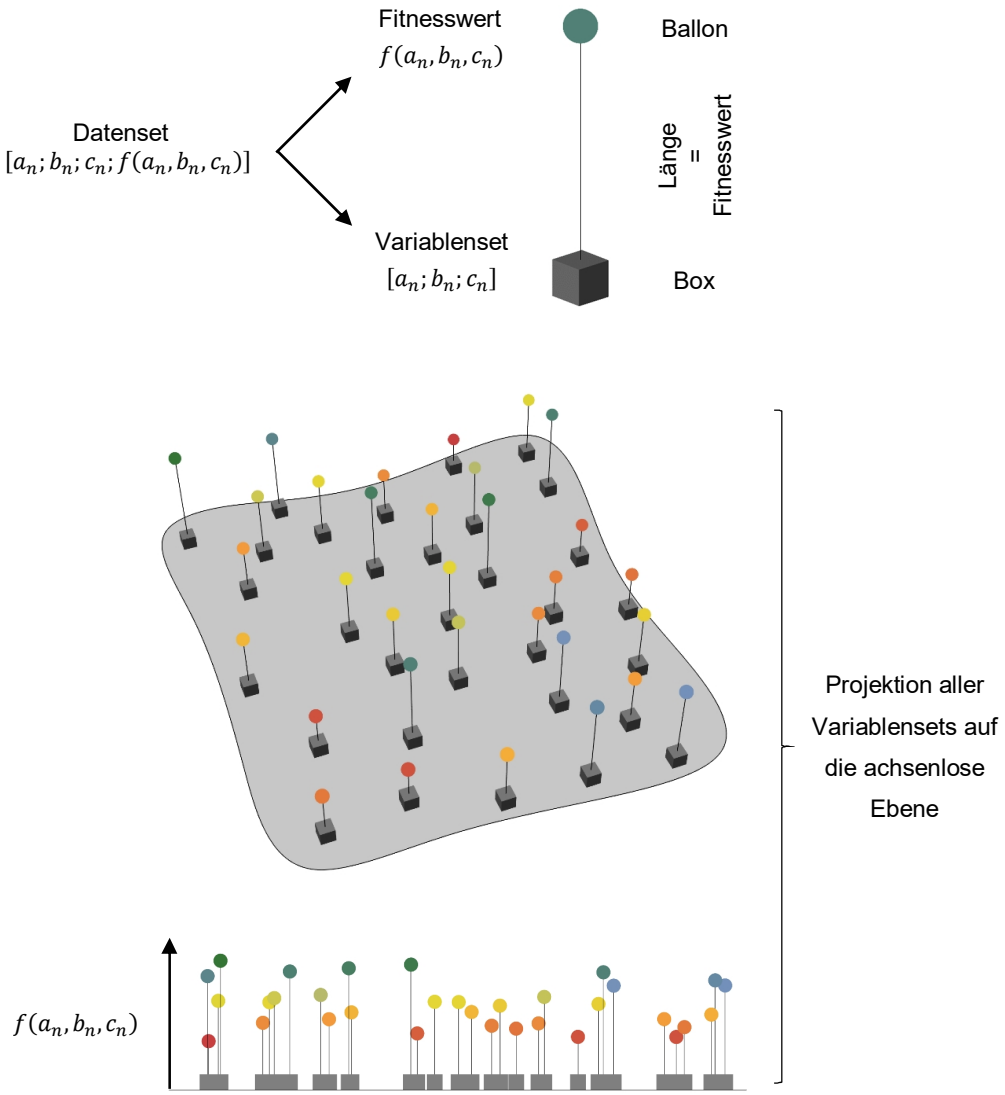


Abbildung 15: Darstellungsprinzip einer multidimensionalen Fitnesslandschaft

Jeder Ballon schwebt entsprechend seinem Fitnesswert in einer bestimmten Höhe über der Ebene. Wird über die Ballons eine Fläche aufgespannt, entsteht eine 3-D Landschaft mit Bergen, Tälern und Ebenen. Siehe Abbildung 16.

Allerdings stellt die aufgespannte Landschaft im Moment eine rein zufällige Typologie dar. Berge und Täler in der Landschaft lassen nur auf die Existenz von Extremwerten (Maxima und Minima) schließen. Andere Eigenschaften der Fitnessfunktion, wie beispielsweise lokale oder globale Extremwerte, stetiges oder diskretes Verhalten, werden nicht sichtbar.

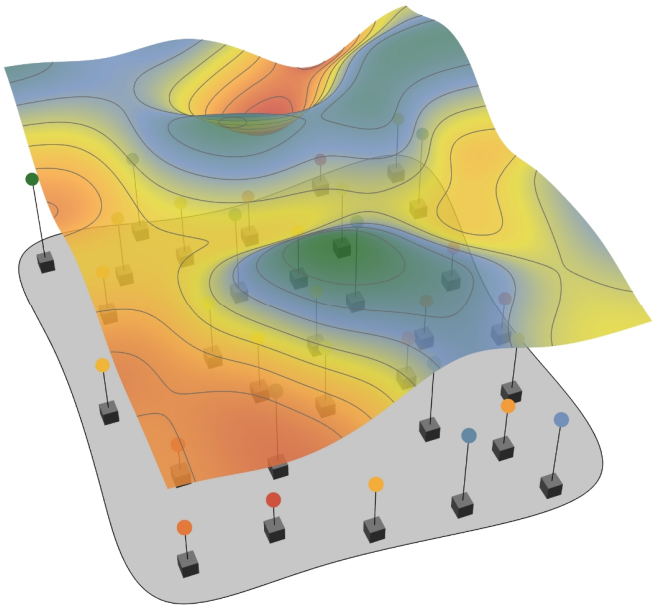


Abbildung 16: Aufgespannte Landschaft (unsortiert)

Zusammengefasst bedeutet das, dass die bisherige Typologie der Fitnesslandschaft noch keinen Mehrgewinn an Informationen über die Fitessfunktion beziehungsweise das Optimierungsproblem liefert. Der Grund dafür ist, dass die Variablensets (Boxen) ohne eine Regel oder ein System auf die achsenlose Ebene projiziert wurden. Ihre Anordnung und damit Position auf der Ebene ist rein zufällig. Um eine aussagekräftige Fitnesslandschaft zu erstellen, müssen die Variablensets sortiert und dementsprechend auf der Ebene angeordnet werden. Aufgrund der hohen Dimensionalität der Variablensets können diese nicht nach ihrem Wert sortiert werden. Auf den Wert n folgt also nicht zwangsläufig der Wert $n + 1$. Um die Sets sinnvoll auf der Ebene anzuordnen, müssen sie nach ihren Ähnlichkeiten sortiert werden. So sind sich beispielsweise die Variablensets $[1; 0; 0]$ und $[1; 1; 0]$ ähnlicher als die Variablensets $[0; 0; 0]$ und $[1; 1; 1]$. Es gilt: Sich zueinander ähnliche Variablensets werden auf der Ebene nah aneinander dargestellt. Sich weniger ähnliche oder gar fremde Variablensets haben auf der Ebene einen größeren Abstand zueinander. In Abbildung 17 sind links die sortierten Variablensets aus obigem Beispiel dargestellt und rechts die dazugehörige aufgespannte Fitnesslandschaft. Es ist wichtig zu wissen, dass die Fitnesswerte (die Höhe der Ballons) von der geänderten Anordnung nicht betroffen sind. Bleibt man in der oben eingeführten Metapher bedeutet das, alle Ballons auf der unsortierten Ebene sind auch in der sortierten Ebene wieder zu finden, es hat sich lediglich ihre Position geändert.

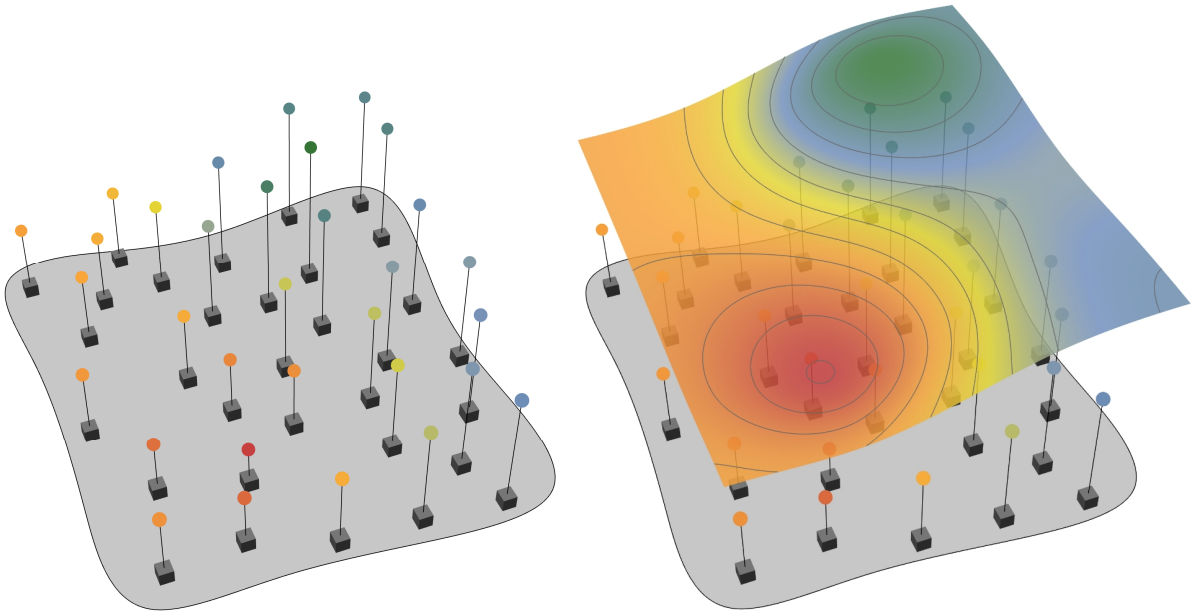


Abbildung 17: Sortierte Variablensets (links); Aufgespannte Fitnesslandschaft (rechts)

Anhand der Fitnesslandschaft in Abbildung 17 kann nun das Optimierungsproblem analysiert werden. Offensichtlich ist, dass das Problem zwei globale Extremwerte, ein Minimum und ein Maximum besitzt. Außerdem handelt es sich um ein Problem mit stetigen Variablen, da in der Landschaft keine Sprünge zu sehen sind. Sprünge werden in der Terminologie der Landschaft auch als Klippen bezeichnet.

Mehrdimensionale Variablensets entsprechend ihren Ähnlichkeiten zu sortieren, ist eine mathematisch komplexe Aufgabe, für die mehrere unterschiedliche Ansätze zur Verfügung stehen. In diesem Zusammenhang wird in der Literatur oft von multidimensionaler Skalierung (MDS) gesprochen, wobei der Begriff je nach Verfahren mathematisch nicht immer korrekt ist. MDS wird dennoch oft als Überbegriff für die Anordnung von mehrdimensionalen Daten auf einer Ebene verwendet.

Zur Bestimmung der Ähnlichkeiten der Variablensets berechnen die meisten gängigen MDS Verfahren die euklidischen Abstände aller Variablen untereinander, siehe [Gl.1] auf Seite 52. Der euklidische Abstand gibt die Distanz zwischen zwei Punkten im n-dimensionalen Raum an. Dabei gilt, je kleiner der Abstand, desto ähnlicher sind sich die Variablensets. Kennt man die Ähnlichkeiten in Form von Abständen, müssen diese auf die projizierten Variablensets auf der Ebene übertragen werden. Die Abstände der zunächst zufällig angeordneten Sets auf der Ebene können recht einfach über den Satz des Pythagoras berechnet werden. Danach beginnt der Iterationsprozess, in dem die Variablensets so lange auf der Ebene verschoben

werden, bis die zweidimensionalen Abstände mit den euklidischen Abständen im n-dimensionalen Raum übereinstimmen.

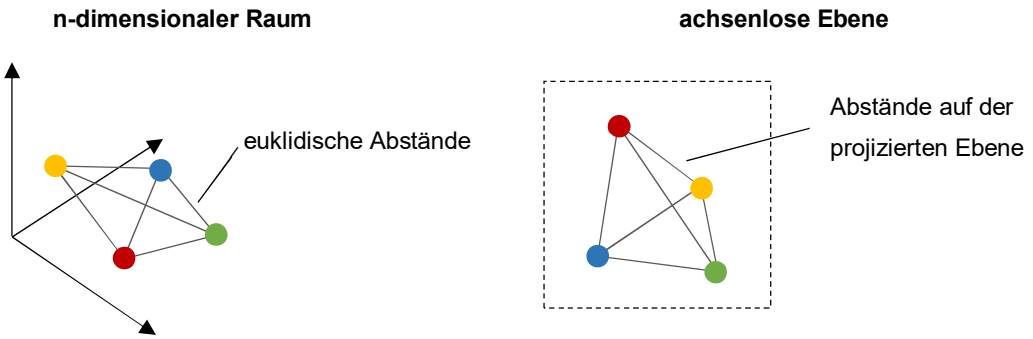


Abbildung 18: Ähnlichkeiten in Form von Abständen

Da es bei einer hinreichenden Anzahl an Variablensets nahezu unmöglich ist, den Iterationsprozess in einer angemessenen Zeit abzuschließen, gibt es einige Näherungsansätze für dieses Problem. In diesen Ansätzen unterscheiden sich auch die Methoden zur Erstellung einer Fitnesslandschaft.

Allerdings haben alle Methoden zwei Gemeinsamkeiten: Sie benötigen trotz der Näherungsansätze sehr viel Rechenleistung und damit Rechenzeit. Zudem sind für die Implementierung dieser Methoden fundierte Kenntnisse der theoretischen Mathematik notwendig, welche über die Ingenieurmathematik hinaus gehen. Beides trägt dazu bei, dass keine dieser Methoden im Ingenieuralltag Anwendung findet. Aus diesem Grund wurde im Rahmen dieser Arbeit ein Verfahren zur Visualisierung von Fitnesslandschaften entwickelt, welches alle oben genannten Aufgaben erfüllt, dafür jedoch deutlich weniger Rechenleistung benötigt und einfach anzuwenden ist. Das Verfahren beruht auf der Verwendung von künstlichen neuronalen Netzen, welche im folgenden Absatz vorgestellt werden. Die Entwicklung und der Ablauf der neuen Methode wird in Kapitel 5 vorgestellt und anhand eines Beispiels veranschaulicht.

4 Prinzipielle Einführung in die künstlichen neuronalen Netzwerke

4.1 Allgemeines

Künstliche neuronale Netze (KNN) sind von den natürlichen neuronalen Netzen im Gehirn inspirierte mathematische Modelle. Ihre Funktionsweise ähnelt der in der Natur. Synapsen feuern bei einer Anregung Impulse ab, welche von den Neuronen verarbeitet werden. Das menschliche Gehirn kann als ein großes neuronales System gesehen werden, welches die Fähigkeit besitzt, mehrdimensionale Informationen, wie die Sinneserfahrungen, dreidimensionalen Regionen im Gehirn zuzuordnen. Anders gesagt: Das Gehirn projiziert mehrdimensionale Informationen auf eine dreidimensionale Struktur. Das natürliche neuronale Netz muss über die Zeit trainiert werden, damit Informationen an die richtige Stelle im Gehirn projiziert werden. Man bezeichnet das gemeinhin als Lernen. KNNs müssen genauso trainiert werden, wobei hier zwischen zwei Trainingsarten unterschieden wird. Überwachtes und unüberwachtes Training. Das überwachte Training erfordert Informationen über den Input und Output, womit dem Netzwerk ein spezielles Verhalten antrainiert wird. Bei einem unüberwachten Netzwerk stehen keine gesonderten Trainingsdaten zur Verfügung, das Netzwerk trainiert sich mit den ihm präsentierten Informationen selbstständig.

Die selbstorganisierende Karte ist eine Art der unüberwachten künstlichen neuronalen Netze, welche im Rahmen dieser Arbeit zum Einsatz kommen.

4.2 Selbstorganisierende Karte

Eine selbstorganisierende Karte, auch Kohonen-Karte oder -Netz, ist eine von dem finnischen Ingenieur Teuvo Kohonen entwickelte Methode, n-dimensionale Daten auf eine zweidimensionale Ebene zu projizieren. Da die Literatur, die sich mit diesem Themenfeld beschäftigt, überwiegend in englischer Sprache verfasst ist, wird im Folgenden die Abkürzung des englischen Begriffs, Self-organizing map (SOM) verwendet.

Als SOM bezeichnet man künstliche neuronale Netzwerke, deren prinzipielles Ziel es ist, eine zweidimensionale Ansicht von n-dimensionalen Daten zu erstellen. Ein ähnliches Ergebnis erzielt das Verfahren der „Multidimensionalen Skalierung“, allerdings bleibt bei der SOM die ursprüngliche Datenstruktur erhalten, also die Dimensionalität, was eine weitere Verwendung der Daten ermöglicht.

Wie bereits erwähnt, wird im Unterschied zu anderen neuronalen Netzwerken das Netzwerktraining nicht überwacht. Das heißt, es stehen keine Trainingsdaten zur Verfügung, die dem Netzwerk ein bestimmtes Verhalten antrainieren. Im Umkehrschluss bedeutet das, dass der Ersteller der SOM die Anordnung der n-dimensionalen Datensets auf der 2D-Karte nicht beeinflussen kann. Die Anordnung oder auch das Aussehen der Karte hängt nur von den Inputdaten selbst ab. Unüberwachtes Lernen bedeutet aber auch, dass die Datensets alle der gleichen Klasse beziehungsweise Dimension entsprechen müssen. Datensets unterschiedlicher Dimensionen können nicht miteinander verglichen werden.

4.2.1 Parameter und Komponenten

Zur Erläuterung der Arbeitsweise eines SOM Algorithmus werden folgende Parameter und Komponenten eingeführt:

Parameter

$x_i = \vec{v}$	Knoten / Datenset	
$\vec{v} = \langle v_1, \dots, v_n \rangle$	Datenvektor eines Knotens	
v_n	Variable (Inputdaten)	
n	Dimension / Anzahl an Variablen	
$i\ [1, \dots, j]$	Index des Datensets	
j	Anzahl der zur Verfügung stehenden Datensets	
I_x	Neuron	
$w_{I_x} = \vec{w}$	Gewicht / Gewichtsvektor	
$\vec{w} = \langle w_1, \dots, w_n \rangle$	Datenvektor eines Neurons	
δ_s	Nachbarschaftsradius	
$\theta(\delta_s)$	Nachbarschaftsfunktion	Zeitabhängig bzw. abhängig vom Iterationsschritt
α_s	Lernrate	
s	Index der Epoche / Iterationsschritt	

Input-Ebene

Die Input-Ebene besteht aus einzelnen Knoten, wobei die Anzahl der Knoten der Anzahl der n-dimensionalen Datensets entspricht, die mit der SOM visualisiert werden sollen. Haben die Inputdaten n Dimensionen, besitzt jeder Knoten in der Input-Ebene n Eigenschaften, wobei die Eigenschaften jeweils ein Datenset repräsentieren. Dementsprechend kann ein Knoten als ein n-dimensionaler Vektor im n-dimensionalen Raum verstanden werden.

Beispiel: Es stehen insgesamt zehn Datensets zur Verfügung. Jedes Datenset spiegelt eine Farbe im RGB-Farbraum wieder. Da eine Farbe im RGB-Farbraum durch einen Wert im Rot,- Grün-, und Blaukanal definiert wird, besitzt jedes Datenset und damit auch jeder Knoten drei Dimensionen. Da jeder Farbkanal im RGB-Farbraum Werte im Intervall von 0-255 annehmen kann, gilt für jeden Knoten folgende Definition:

$$x_{i_{\{i=1,..,10\}}} = \begin{bmatrix} v_1 = R \\ v_2 = G \\ v_3 = B \end{bmatrix} \text{ mit } R, G, B = [0{,}255]$$

Output-Ebene

Die Output-Ebene besteht aus den Neuronen des künstlichen neuronalen Netzes, welche in Spalten und Zeilen auf einer zweidimensionalen Ebene angeordnet sind. Sie bilden die SOM, auf der nach der Iteration die n-dimensionalen Eingangsdaten dargestellt werden. Die Anzahl der Neuronen und somit die Größe der Karte hängt von der Art der darzustellenden Daten ab. Mehr hierzu in Abschnitt 4.2.2. Es kann jedem Knoten immer genau ein Neuron zugeordnet werden, aber nicht jedem Neuron ein Knoten. Es ist wichtig, dass die Neuronen auf der Output-Ebene und die Knoten auf der Input-Ebene die gleiche Dimension besitzen. Nur dann können Knoten und Neuronen miteinander verglichen werden.

Gewichtsvektor

Der Gewichtsvektor w_{I_x} beschreibt den euklidischen Abstand im n-dimensionalen Raum zwischen dem Knoten x_i auf der Input-Ebene und dem Neuron I_x auf der Output-Ebene.

Ist der Knoten $x_i = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$ und das Neuron $I_x = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$, so gilt für den euklidischen Abstand:

$$w_{I_x} = \sqrt{(v_1 - w_1)^2 + \cdots + (v_n - w_n)^2} = \sqrt{\sum_{i=1}^{n} (v_i - w_i)^2} \quad \text{[Gl.1]}$$

Initialisierung

Zunächst sind nur die Inputdaten, also die Datenvektoren $\vec{v}$ der und deren Dimension bekannt. Den Neuronen auf der Output-Ebene sind noch keine Datensets, auch Gewichte genannt, zugeordnet. Dies geschieht, nachdem die Knoten das erste Mal der SOM präsentiert werden.

Die Neuronen erhalten zufällige Gewichte, deren Wert sich im Intervall der Inputdaten befindet.

Normalisieren von Variablen

Anhand der untenstehenden Beispielknoten x_1, x_2 und x_n ist zu erkennen, dass sich die Variablen v_1, v_2 und v_3 in unterschiedlichen Intervallen befinden. Für die Initialisierung der Neuronen werden jedoch nicht die Intervalle der einzelnen Variablen betrachtet, sondern das Intervall aller Knoten als Ganzes. In diesem Beispiel bekommt die Output-Ebene die Information: „Initialisiere zufällige Gewichte in dem Intervall $[11; 1327]$“.

Knoten	Intervall der Variablen	Intervall aller Knoten	Neuron
$x_1 = \begin{bmatrix} v_1 = 205 \\ v_2 = 1327 \\ v_3 = 29 \end{bmatrix}$	$v_1 = [205; 294]$ $v_2 = [1033; 1327]$ $v_3 = [11; 29]$	$[11; 1327]$	$I_1 = \begin{bmatrix} 1287 \\ 13 \\ 681 \end{bmatrix}$
$x_2 = \begin{bmatrix} v_1 = 234 \\ v_2 = 1033 \\ v_3 = 11 \end{bmatrix}$			$I_2 = \begin{bmatrix} 11 \\ 16 \\ 966 \end{bmatrix}$
$x_n = \begin{bmatrix} v_1 = 294 \\ v_2 = 1201 \\ v_3 = 17 \end{bmatrix}$			$I_n = \begin{bmatrix} 1305 \\ 1289 \\ 584 \end{bmatrix}$

Falls die Intervalle der einzelnen Variablen zu stark voneinander abweichen, kann das bei der Organisation der SOM zu Fehlern führen. Aus diesem Grund sollten die Inputdaten zunächst normalisiert werden, um eine faire Verteilung der Neuronen zu gewährleisten. Dabei ist es unwichtig, auf welchen Wertebereich die Variablen bei der Normalisierung skaliert werden. Für diese Arbeit wurde festgelegt, dass alle Variablen immer auf den Wertebereich $[0; 1]$ skaliert werden. Die Normalisierung erfolgt nach Formel [Gl.2]:

$$v_{n,norm} = \frac{v_n - v_{min}}{v_{max} - v_{min}} \qquad \text{[Gl.2]}$$

mit: $x_{v,norm}$ normalisierte Variable

x_v Ausgangsvariable

v_{min} / v_{max} obere / untere Grenze des Variablenintervalls

Die oben eingeführten Beispielknoten werden wie folgt normalisiert:

Knoten	$x_1 = \begin{bmatrix} v_1 = 205 \\ v_2 = 1327 \\ v_3 = 29 \end{bmatrix}$	$x_2 = \begin{bmatrix} v_1 = 234 \\ v_2 = 1033 \\ v_3 = 11 \end{bmatrix}$	$x_n = \begin{bmatrix} v_1 = 294 \\ v_2 = 1201 \\ v_3 = 17 \end{bmatrix}$
Normalisieren der Variablen	$v_1 = [205; 234; 294]$ $v_2 = [1033; 1201; 1327]$ $v_3 = [11; 17; 29]$	$\longrightarrow$	$v_{1,norm} = [0; 0{,}32; 1]$ $v_{2,norm} = [0; 0{,}57; 1]$ $v_{3,norm} = [0; 0{,}33, 1]$
normalisierte Knoten	$x_{1,norm} = \begin{bmatrix} v_1 = 0 \\ v_2 = 1 \\ v_3 = 1 \end{bmatrix}$	$x_{2,norm} = \begin{bmatrix} v_1 = 0{,}32 \\ v_2 = 0 \\ v_3 = 0 \end{bmatrix}$	$x_{n,norm} = \begin{bmatrix} v_1 = 1 \\ v_2 = 0{,}57 \\ v_3 = 0{,}33 \end{bmatrix}$
Neuron	$I_1 = \begin{bmatrix} 0{,}98 \\ 0{,}02 \\ 0{,}56 \end{bmatrix}$	$I_2 = \begin{bmatrix} 0{,}06 \\ 0{,}31 \\ 0{,}63 \end{bmatrix}$	$I_n = \begin{bmatrix} 0{,}93 \\ 0{,}76 \\ 0{,}42 \end{bmatrix}$

Unter Kenntnis von v_{min} und v_{max} können die normalisierten Variablen wieder auf den ursprünglichen Wertebereich zurück skaliert werden. Dies funktioniert selbstverständlich auch bei den Gewichten der Neuronen. So kann an der konvergierten SOM nachvollzogen werden, welche Variablenwerte ein Neuron repräsentiert.

Überblick

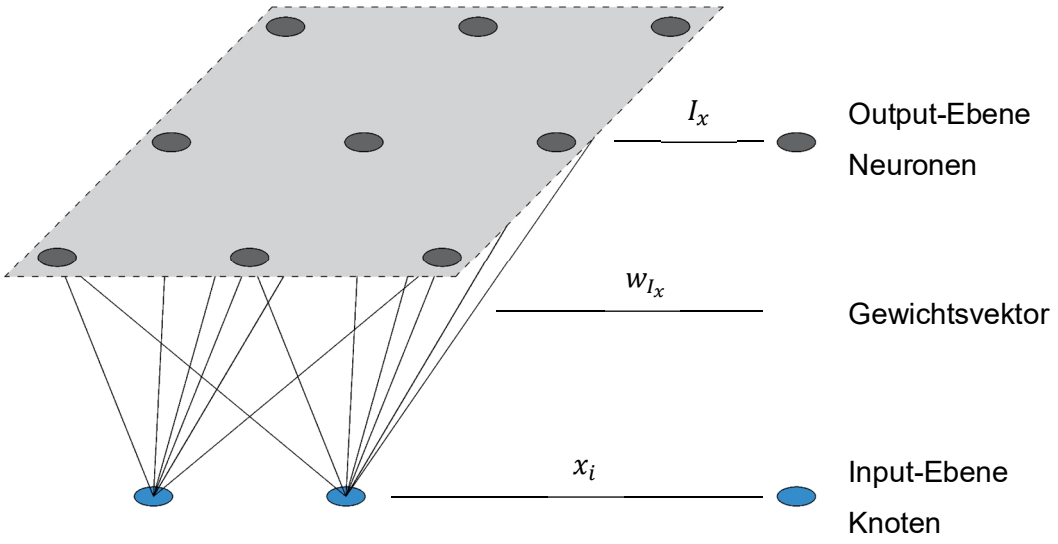


Abbildung 19: Komponenten einer SOM

Ein generelles Verständnis über die Komponenten und Arbeitsweise eines SOM Algorithmus ist für dessen Anwendung unerlässlich. Obwohl man, wie bereits erwähnt, auf das Ergebnis einer SOM keinen direkten Einfluss nehmen kann, spielt die Wahl der Randbedingungen und

Parameter sehr wohl eine große Rolle. Auf Änderungen kann die SOM unter Umständen empfindlich reagieren. Wird beispielsweise die Anzahl der Neuronen zu groß oder zu klein gewählt, findet der Algorithmus kein akzeptables Ergebnis. Für die Wahl der Parameter gibt es nicht immer konkrete Richtwerte, oft muss man entweder auf eigene Erfahrungswerte zurückgreifen oder die Trial-and-Error Methode anwenden.

4.2.2 Ablauf einer SOM Organisation

Abbildung 20 zeigt ein Ablaufdiagramm mit den notwendigen Rechenschritten zur Organisation einer SOM.

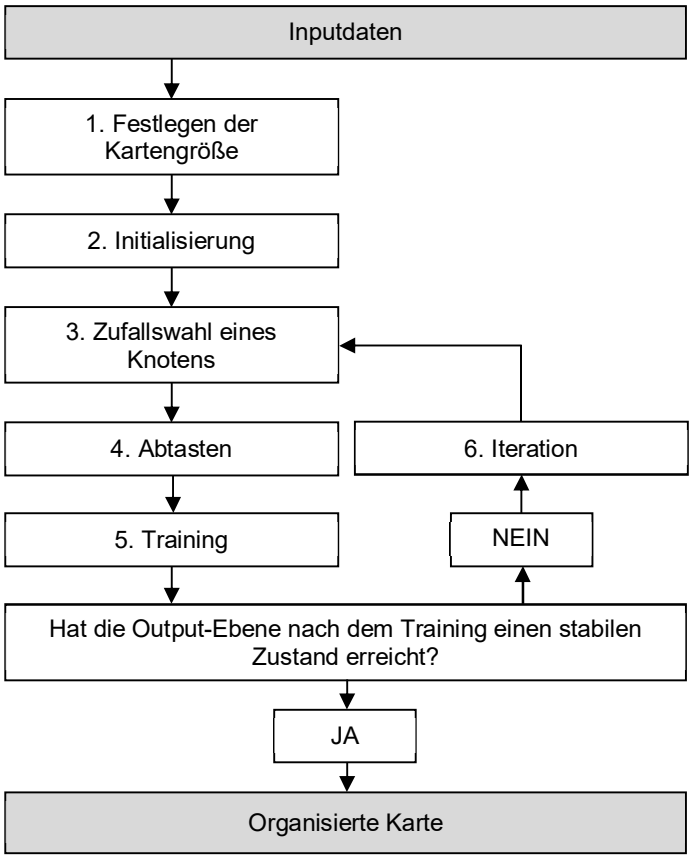


Abbildung 20: Ablauf einer SOM Organisation

Die Schritte 1 bis 6 werden im Folgenden detailliert beschrieben.

1. Festlegen der Kartengröße

Zunächst wird anhand der Anzahl der Knoten auf der Input-Ebene die Anzahl der Neuronen auf der Output-Ebene festgelegt. Im Zuge dieser Arbeit wurde festgestellt, dass die besten Ergebnisse zu erwarten sind, wenn die Output-Ebene mindestens fünfmal so groß ist wie die Input-Ebene. Stehen zu wenig Neuronen zur Verfügung, konvergiert der SOM Algorithmus nicht. Werden zu viele Neuronen angeordnet, gibt es Bereiche auf der Karte, die von den Inputdaten nicht stimuliert werden. Das bedeutet, dass die Karte zwar konvergiert, aber nicht alle Neuronen die Inputdaten repräsentieren.

2. Initialisierung

In folgender Abbildung ist eine Output-Ebene mit zwei Knoten und eine Input-Ebene mit neun Neuronen dargestellt. Den Knoten ist jeweils ein dreidimensionales, normalisiertes Variablenset zugeordnet. Nach der Initialisierung besitzen auch die Neuronen zufällige, dreidimensionale Gewichte.

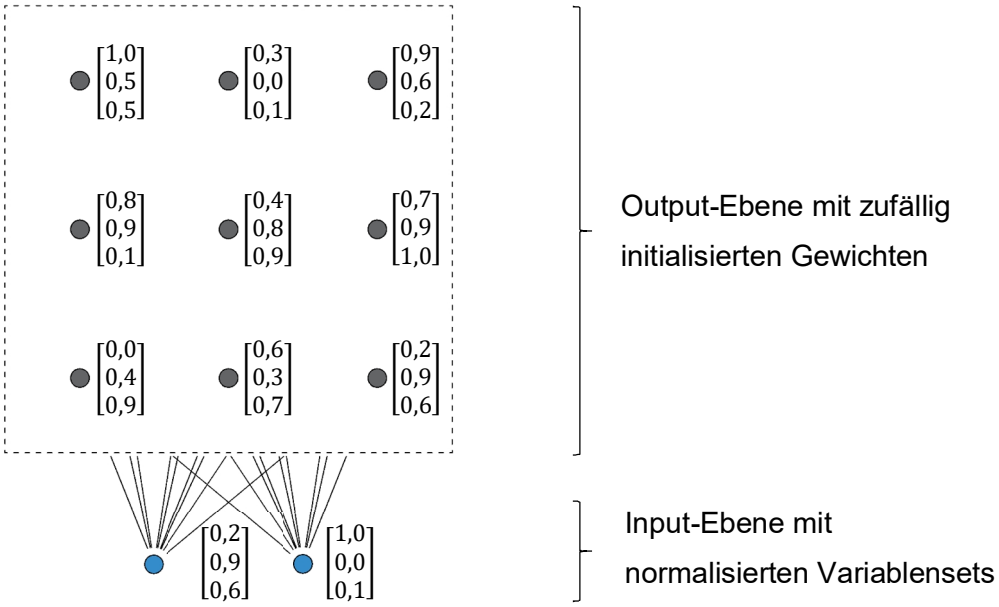


Abbildung 21: Initialisierte Output-Ebene

3. Zufallswahl eines Knotens

Nach der Initialisierung wird zunächst per Zufall ein Knoten auf der Input-Ebene ausgewählt. Wurde ein Knoten gewählt, kann er in dem gleichen Iterationsschritt nicht noch einmal gewählt werden. Erst im nächsten Iterationsschritt stehen wieder alle Knoten zur Verfügung.

4. Abtasten

Ausgehend von dem zuvor gewählten Knoten wird zu jedem einzelnen Neuron auf der Output-Ebene der Gewichtsvektor aufgestellt. Der Gewichtsvektor beziehungsweise die Länge des Gewichtsvektors repräsentiert den Abstand zwischen Knoten und Neuron. Es wird das Neuron mit dem kleinsten Gewichtsvektor, also mit dem geringsten Abstand zum Knoten, gesucht. Ein geringer Abstand bedeutet, dass das Variablenset des Knotens und die Gewichte des Neurons sich ähnlich sind, ein Abstand von 0 bedeutet, sie sind identisch. Das Neuron mit dem geringsten Abstand wird als Gewinnerneuron bezeichnet. Je nach SOM Algorithmus kann die Länge des Gewichtsvektors mit Hilfe von verschiedenen Methoden bestimmt werden. Für niedrige Dimensionen bietet sich die Berechnung des Skalarprodukts an, oft wird

jedoch der Hemming-Abstand oder der euklidische Abstand berechnet, da diese Methoden auch für n-Dimensionen und nicht normalisierte Daten zuverlässig arbeiten. Der in dieser Arbeit verwendete SOM Algorithmus berechnet die Länge der Gewichtsvektoren durch den euklidischen Abstand. Siehe Formel [Gl.1] auf Seite 52.

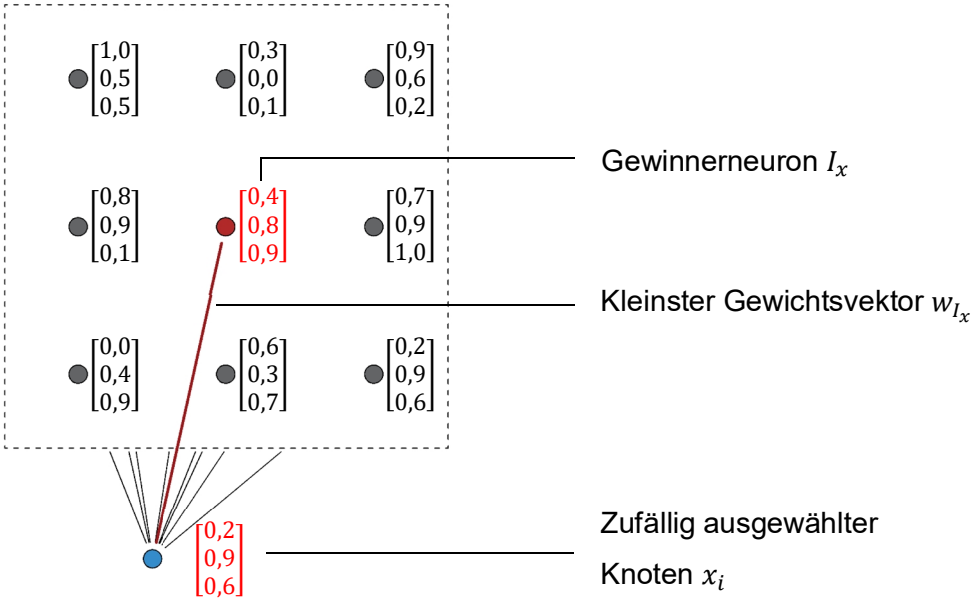


Abbildung 22: Abtasten der Output-Ebene und Bestimmen des Gewinnerneurons

5. Training

In der Trainingsphase, oder auch Lernphase, wird die sichtbare Output-Ebene gemäß den Inputdaten organisiert. Die Neuronen auf der Output-Ebene werden durch die Knoten auf der Input-Ebene so trainiert, dass sich am Ende beide Ebenen gleichen. Um dies zu erreichen sind je nach Dimensionen und Anzahl der Datensets mehrere hundert bis tausend Iterationsschritte nötig. Ein Iterationsschritt wird als Epoche bezeichnet, in der die Karte mit jedem Knoten genau einmal trainiert wird. In diesem Beispiel sind das zwei Trainingsschritte pro Epoche, wobei der Index einer Epoche s ist.

Der Gewichtvektor w_{I_x} des Gewinnerneurons I_x wird so angepasst, dass sich der Abstand zwischen Knoten und Neuron verkleinert. Räumlich ist dieser Vorgang als eine Verschiebung des Neurons in Richtung des Knotens zu verstehen. Wie stark beziehungsweise wie weit sich das Neuron in Richtung des Knotens bewegt, hängt von der Lernrate α_s ab, wobei die Lernrate mit zunehmenden Iterationsschritten abnimmt.

Während das Gewinnerneuron gemäß der Lernrate angepasst wird, werden die Neuronen in der Nachbarschaft auch angepasst. Allerdings sinkt die Lernrate für die

Nachbarschaftsneuronen mit zunehmendem Abstand zum Gewinnerneuron. Diese Abnahme wird durch die Nachbarschaftsfunktion $\theta(\delta_s)$ beeinflusst. Ob sich ein Neuron in der Nachbarschaft des Gewinnerneurons befindet, wird durch den Nachbarschaftsradius δ_s bestimmt. Auch dieser ist abhängig von den Iterationsschritten und nimmt mit zunehmenden Schritten ab.

Der Gewichtsvektor wird wie folgt angepasst:

$$w_{I_x\,(s+1)} = w_{I_x\,(s)} + \theta(\delta_s) \cdot \alpha_s \cdot \left[x_i - w_{I_x\,(s)}\right] \qquad \text{[Gl.3]}$$

mit:

w_{I_x}	Gewichtsvektor
x_i	Knoten
δ_s	Nachbarschaftsradius
$\theta(\delta_s)$	Nachbarschaftsfunktion
α_s	Lernrate
s	Index der Epoche / Iterationsschritt

In folgender Abbildung ist der erste Trainingsschritt für den zufällig ausgewählten Knoten dargestellt. Die Gewichtsvektoren und damit auch die Gewichte der Neuronen innerhalb des Nachbarschaftsradius bewegen sich gemäß der Lernrate in Richtung des Knotens.

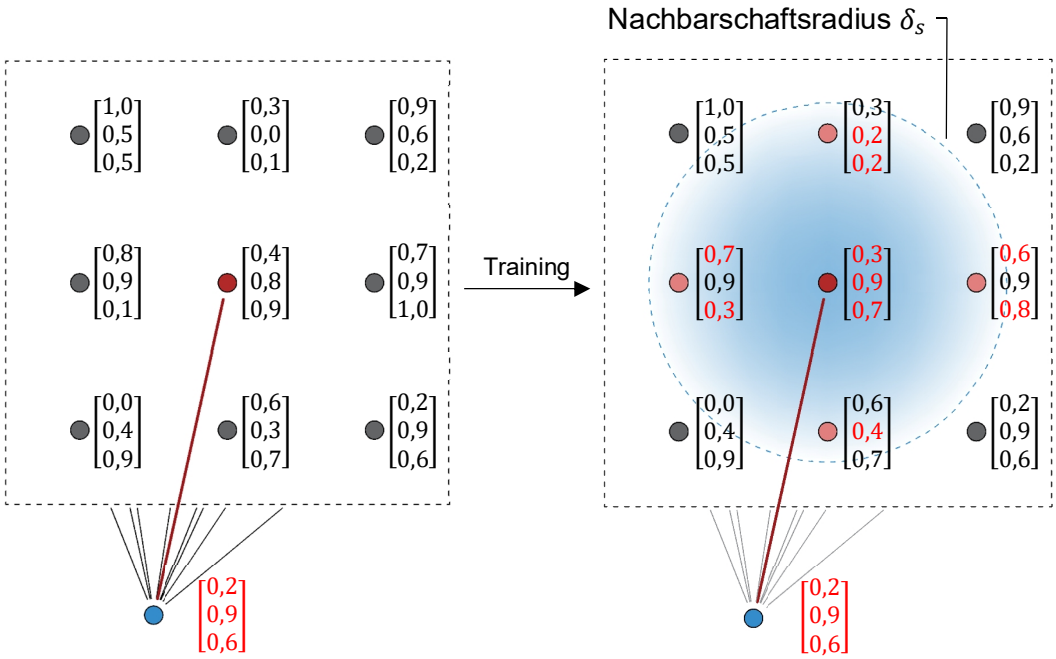


Abbildung 23: Trainieren der Neuronen innerhalb des Nachbarschaftsradius

6. Iteration

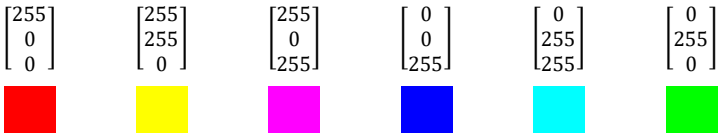
Nach jedem Trainingsschritt wird überprüft, ob sich die Gewichte der Neuronen während des Trainings verändert haben oder ob ein zuvor definierter Grenzwert erreicht wurde. Ist dies nicht der Fall, werden die Schritte 2, 3 und 4 so lange wiederholt, bis die SOM einen stabilen Endzustand erreicht hat. Nun gilt die Karte als konvergiert.

4.2.3 Beispiel

Das folgende Beispiel wird oft in der Literatur verwendet, da es die Funktionsweise einer SOM anschaulich darstellt. Das Beispiel wurde mit der Komponente „Kohonen Map“ für das Programm Grasshopper erstellt.

Es sollen farbige Punkte organisiert werden, wobei jeder Punkt über den Wert des Rot-, Grün-, und Blaukanals im RGB-Farbraum definiert wird. Das heißt, jeder Punkt besitzt drei Variablen, also drei Dimensionen. Als Inputdaten wurden der SOM insgesamt sechs Datensets zur Verfügung gestellt. Auf eine Normalisierung der Variablen kann in diesem Beispiel verzichtet werden, da der Variablenwert der Inputdaten nur einen von zwei Zuständen einnimmt. Entweder 0 oder 255, die Extreme im RGB-Farbraum.

Input Daten



Nachdem die obenstehenden Inputdaten der SOM das erste Mal präsentiert wurden, findet die Initialisierung statt. Die Größe der Output-Ebene umfasst 20x20, also 400 Neuronen, welchen nun zufällige Gewichte im Intervall von 0 bis 255 zugeordnet werden. Diese zufälligen Gewichte sind für jedes Neuron übersetzbar in eine Farbe, welche in Abbildung 24: *Initialisierte Karte* zu sehen sind. Nach der Initialisierung erfolgen die Schritte 3, 4 und 5 der Organisation, siehe Absatz 4.2.2. Für dieses Beispiel benötigt der SOM Algorithmus insgesamt 10.400 Iterationsschritte, um zu konvergieren, was bei 400 Neuronen 26 Epochen entspricht.

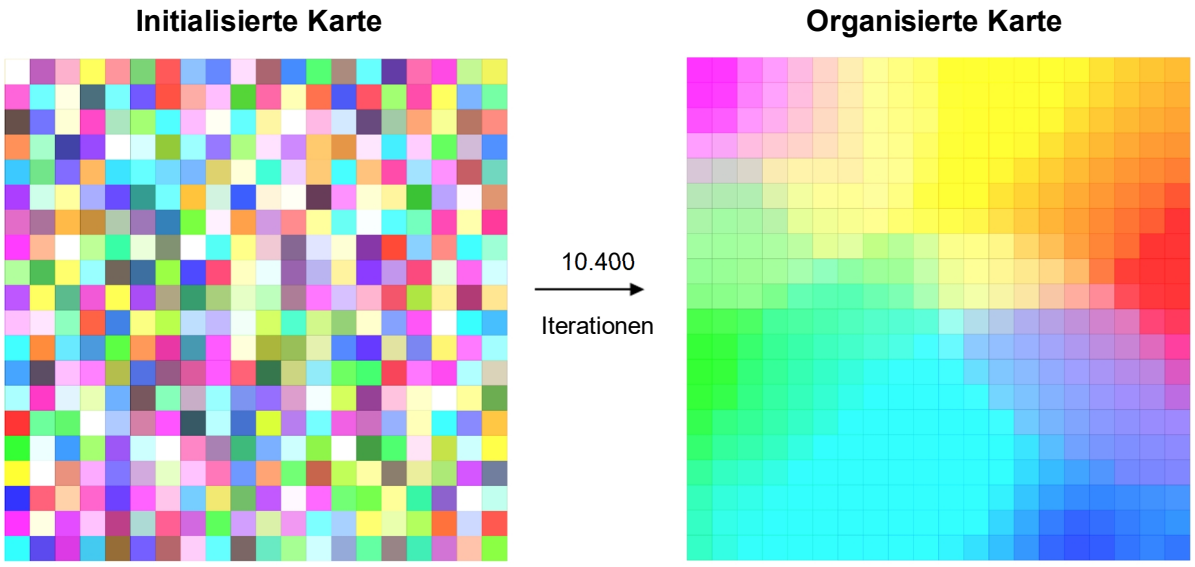
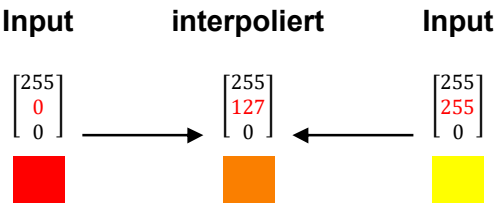


Abbildung 24: Beispiel einer SOM

Auf der fertigen, organisierten Karte sind zwei wichtige Merkmale einer SOM zu erkennen:

1. Alle Inputdaten, hier die Inputfarben, sind auf der organisierten Karte eindeutig wieder zu finden. Das bedeutet, dass sich die SOM tatsächlich nach den ihr präsentierten Daten organisiert und diese nach ihren Ähnlichkeiten wiedergibt. Farben wie Braun, Schwarz und Weiß sind in der Karte nicht zu finden.

2. Der Übergang zwischen den Farben ist nicht abrupt, sondern fließend. So erscheint zum Beispiel zwischen Rot und Gelb die Farbe Orange, obwohl sich diese nicht unter den Farben der Inputdaten befindet. Das heißt, die Gewichte der Neuronen, die zwischen Rot und Gelb liegen, wurden wie folgt interpoliert:



Vor allem das Interpolieren zwischen den Inputdaten ist eine wichtige Eigenschaft der SOM, denn so werden eventuelle Variablenkombinationen sichtbar, die vor dem Erstellen der SOM nicht beachtet wurden. Bei dem obigen „Farbenbeispiel“ ist es offensichtlich, dass zwischen Rot und Gelb Orange liegt. Bei einem n-dimensionalen Datenset, welches keine Farbe, sondern beispielsweise eine geometrische Form beschreibt, ist der Interpolationsschritt nicht mehr so offensichtlich. Durch eine SOM könnten so Formen sichtbar werden, an die aufgrund der großen Anzahl an Möglichkeiten zuvor nicht gedacht wurde.

5 Entwicklung der n-1-dimensionalen Gruppierung als Methode zur Visualisierung von mehrdimensionalen Daten

Die Visualisierung von mehrdimensionalen Fitnesslandschaften mit der Methode der n-1-dimensionalen Gruppierung beinhaltet kein neues mathematisches Verfahren und auch keinen neuen Ansatz zur multidimensionalen Skalierung. Es ist vielmehr ein neuer Prozessablauf, gepaart mit der bereits bestehenden Methode der selbstorganisierenden Karte (SOM). Die prinzipielle Funktionsweise einer SOM wird in Kapitel 4 ausführlich erläutert. Das Verfahren der SOM wird für die Visualisierung von Fitnesslandschaften zweckentfremdet, da es bisher vorwiegend in den Forschungsbereichen der künstlichen Intelligenz, Bioinformatik und Robotik zum Clustern von Daten angewendet wurde. Entscheidend ist jedoch nicht nur die Verwendung von künstlichen neuronalen Netzen, sondern der gesamte Prozessablauf und die Art der Datengewinnung.

Bei allen herkömmlichen Methoden zur Darstellung einer Fitnesslandschaft, werden die Daten vor der Visualisierung und unabhängig von der verwendeten Visualisierungsmethode gewonnen. Als Daten wird die Summe aller Sets, bestehend aus Variablen und dazugehörigem Fitnesswert bezeichnet. Eine Möglichkeit die Daten zu gewinnen, ist die Brute-Force-Methode. Dabei wird jede mögliche Variablenkombination inklusive dem dazugehörigen Fitnesswert ermittelt, berechnet und für die Visualisierung gespeichert. In den meisten Fällen ist das jedoch keine zielführende Methode, da es schlichtweg zu viele Kombinationsmöglichkeiten gibt. Ein dreidimensionales Problem mit einem Variablenraum von 0 bis 100 besitzt bereits 100^3 = 1 Million Variablenkombinationen. Eine weitere Möglichkeit, und zwar die am häufigsten verwendete, ist es, die Variablen und Fitnesswerte während des Optimierungsprozesses aufzuzeichnen. So stehen nach der Optimierung die Daten zur Verfügung, die der Optimierungsalgorithmus zum Finden der optimalen Lösung verwendet hat. Wie bereits beschrieben, kann aufgrund der großen Datenmengen nie eine komplette Landschaft dargestellt werden. Die visualisierten Daten sind deshalb als ein Ausschnitt aus der gesamten Fitnesslandschaft zu verstehen oder auch als den Teil der Landschaft, den der Optimierungsalgorithmus während des Optimierungsprozesses abgetastet hat. Anschließend wird unabhängig von der Methode der Datengewinnung die Fitnesslandschaft mit dem in Absatz 3.2 beschriebenen Prozess erstellt.

Die Methode der n-1-dimensionalen Gruppierung verfolgt sowohl bei der Datengewinnung, als auch bei dem Sortieren der Daten einen neuen Ansatz. In folgendem Abschnitt wird der Visualisierungsprozess mit der Methode der n-1-dimensionalen Gruppierung vorgestellt.

5.1 Visualisierungsprozess

Die Methode der n-1-dimensionalen Gruppierung zur Visualisierung von Fitnesslandschaften nutzt die positiven Eigenschaften der selbstorganisierenden Karte zum Sortieren und Ordnen der mehrdimensionalen Variablensets. Für die folgende Erläuterung der Methode ist deshalb ein Verständnis über die prinzipielle Arbeitsweise von künstlichen neuronalen Netzen, im speziellen von selbstorganisierenden Karten, hilfreich. In Absatz 4.2 werden die Grundlagen, Arbeitsweisen und Komponenten einer SOM ausführlich erläutert.

Die vier Schritte des Visualisierungsprozesses werden im Folgenden vorgestellt.

Schritt 1: Aufzeichnen der Variablensets

Wie auch andere Methoden, zeichnet die Methode der n-1-dimensionalen Gruppierung während des Optimierungsprozesses die Daten auf, die von dem Optimierungsalgorithmus berücksichtigt wurden. Allerdings sind hier zunächst nur die Variablensets von Bedeutung, die dazugehörigen Fitnesswerte werden zu Beginn noch nicht benötigt. Von einem Optimierungsproblem mit n Dimensionen werden also nur n-1 Dimensionen aufgezeichnet.

Schritt 2: Dimensionsreduktion: Sortieren der Variablensets durch das künstliche neuronale Netz einer selbstorganisierenden Karte

Im zweiten Schritt werden die aufgezeichneten Variablensets nicht direkt auf eine Ebene projiziert und nach ihren Ähnlichkeiten sortiert. Den Sortiervorgang übernimmt hier das künstliche neuronale Netz der selbstorganisierenden Karte. Dazu werden die aufgezeichneten Variablensets der SOM auf einer Input-Ebene in Form von Knoten präsentiert. Nach dem Iterationsprozess, dem Training der SOM, sind die Variablensets in Form von Neuronen auf der Output-Ebene nach Ähnlichkeiten angeordnet. Hier zahlen sich die beiden Vorteile einer SOM aus. Zum einen benötigt das Training einer SOM deutlich weniger Rechenleistung und Rechenzeit als der iterative Sortiervorgang herkömmlicher Methoden. Zum anderen verarbeitet eine SOM nicht nur die ihr präsentierten Inputdaten, sondern interpoliert zusätzlich zwischen den Variablensets. So können eventuelle Lücken in der Landschaft, die von dem Optimierungsalgorithmus nicht abgetastet wurden, geschlossen werden. Da die Output-Ebene einer SOM mindestens fünfmal so groß sein sollte wie die Input-

Ebene, sind für die Erstellung der Fitnesslandschaft deutlich mehr Variablensets vorhanden, als nach der Optimierung bekannt waren. So sind in der Landschaft auch Variablenkombinationen vertreten, die bei der Optimierung nicht berücksichtigt wurden.

Bei den oben genannten Vorteilen hat eine SOM auch einen Nachteil beziehungsweise eine Eigenschaft, welche bei der Analyse von Fitnesslandschaften berücksichtigt werden muss. Während die Organisation mit Hilfe des euklidischen Abstands ein lineares Verfahren ist, siehe Absatz 3.3, ist die SOM ein nichtlineares Verfahren. Das bedeutet, die Variablensets sind zwar nach ihrer Ähnlichkeit sortiert, jedoch entspricht der Abstand auf der Ebene nicht mehr dem tatsächlichen Abstand im n-dimensionalen Raum. In Abbildung 25 sind links vier Variablensets mit der Dimension 2 im Raum dargestellt. Die Abstände zueinander entsprechen den tatsächlichen im n-dimensionalen Raum. Auf der rechten Seite werden die vier Sets nach der Organisation durch eine SOM dargestellt. Sie sind offensichtlich nach Ähnlichkeit sortiert, allerdings sind die Informationen über die tatsächlichen Abstände verloren gegangen. Die tatsächlichen Abstände wurden nichtlinear skaliert.

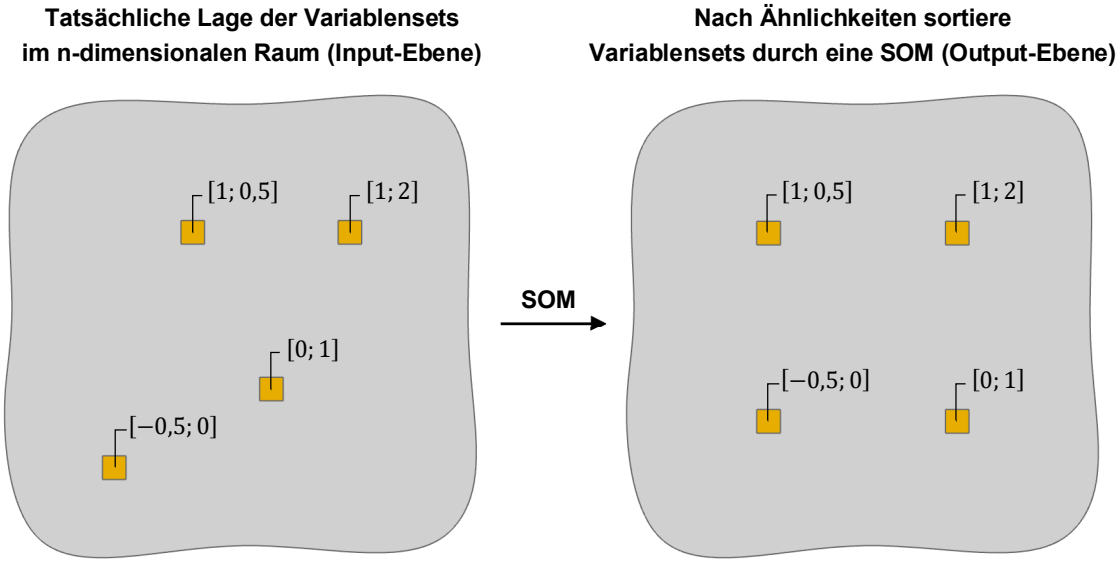


Abbildung 25: Sortieren von Variablensets durch eine SOM (Dimensionsreduktion)

Da sich dieses Beispiel im zweidimensionalen Raum bewegt, können sich die Variablensets als Punkte auf einer X-Y-Ebene vorgestellt werden. Durch die niedrige Dimensionalität ist bereits die Input-Ebene (Abbildung 25, links) visualisierbar und eine Organisation beziehungsweise Dimensionsreduktion durch eine selbstorganisierende Karte scheint überflüssig. Allerdings ist dieses Prinzip für beliebig hohe Dimensionen gültig. Unabhängig von der Dimensionalität der Input-Ebene, wird die Output-Ebene (Abbildung 25, rechts) immer im gleichmäßigen Raster auf einer zweidimensionalen Ebene dargestellt. Durch die

nichtlineare Skalierung der Abstände und die Rasteranordnung auf der Output-Ebene spricht man auch von einer Gruppierung ähnlicher Daten.

Schritt 3: Nachträgliche Berechnung der Fitnesswerte

Abbildung 25 stellt die Funktionsweise einer SOM nicht korrekt dar. Die Abbildung dient lediglich zur Demonstration der nichtlinearen Skalierung der Abstände. Würden der SOM, wie in Abbildung 25 dargestellt, vier Variablensets (Knoten) auf der Input-Ebene präsentiert werden, würde die Output-Ebene aus mindestens 20 Variablensets (Neuronen) bestehen. Außerdem besteht keine direkte Verbindung zwischen Knoten und Neuronen. Das bedeutet, dass die Variablensets auf der Output-Ebene die Variablensets auf der Input-Ebene zwar repräsentieren, sie jedoch nicht zu 100 % identisch sind. Je nach definierter Lernrate existiert unter Umständen eine kleine Diskrepanz zwischen Input und Output. Deshalb werden, wie bereits in Schritt 1 beschrieben, nur die Variablensets aufgezeichnet, da die Fitnesswerte nicht mehr exakt zu den Variablensets auf der Output-Ebene passen.

Aus diesem Grund wird in Schritt 3 zu jedem sortierten Variablenset auf der Output-Ebene der dazugehörige Fitnesswert neu berechnet. Die Berechnung erfolgt nach dem gleichen Schema wie bei der eigentlichen Optimierung. Da alle zu berechnenden Variablensets durch die SOM bekannt sind, wird der eigentliche Optimierungsalgorithmus nicht mehr benötigt, sondern nur noch der spezielle Solver beziehungsweise die Fitnessfunktion.

Schritt 4: Abtragen der Fitnesswerte auf der Hochachse und Aufspannen der Fitnesslandschaft

Schritt 4 unterscheidet sich nicht von dem allgemeinen Visualisierungsprozess. Die Fitnesswerte sind bekannt und alle Variablensets sind sortiert und auf der achsenlosen Ebene im Raster angeordnet. Anschließend werden die Fitnesswerte an der Stelle des entsprechenden Variablensets auf die Hochachse abgetragen. Die Hochachse hat ihren Ursprung auf Höhe der achsenlosen Ebene, wobei die positive Richtung nach oben definiert ist. Anschließend wird zur Visualisierung der Landschaft eine Fläche über alle Fitnesswerte gespannt. Höhenlinien und eine der Höhe entsprechende Farbcodierung sorgen dafür, dass die 3-D Landschaft bei einer ebenen Darstellung auf Papier oder einem Bildschirm leicht zu verstehen ist.

Zusammenfassung

Die Visualisierung von Fitnesslandschaften mit der Methode der n-1-dimensionalen Gruppierung unterscheidet sich im Wesentlichen in zwei Punkten von den herkömmlichen Visualisierungsmethoden. Zum einen geschieht die Dimensionsreduktion der Variablensets mit Hilfe einer selbstorganisierenden Karte, wobei eventuelle Lücken in der Landschaft durch die Interpolation von Variablen geschlossen werden. Zum anderen werden die Fitnesswerte nicht beim Optimierungsprozess mitgeschrieben, sondern im Nachgang noch einmal durch die Fitnessfunktion beziehungsweise durch den speziellen Solver berechnet. Folgendes Ablaufdiagramm stellt den Optimierungsprozess und den Visualisierungsprozess mit der Methode der n-1-dimensionalen Gruppierung dar.

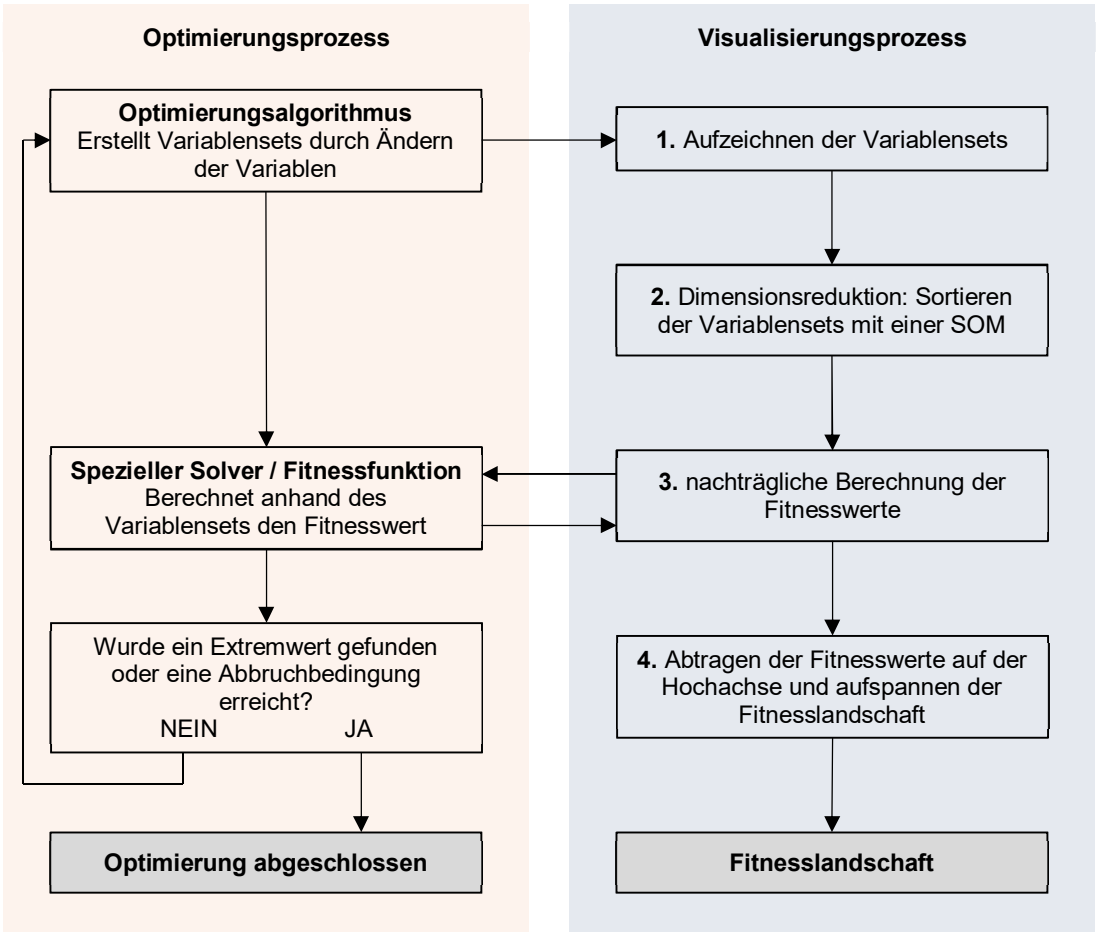


Abbildung 26: Interaktion des Optimierungs- und Visualisierungsprozesses

5.2 Namensgebung

N-1-dimensionale Gruppierung beschreibt den wichtigsten Schritt dieser neuen Visualisierungsmethode: Das Sortieren der Variablensets mit einer selbstorganisierenden Karte. Der Name setzt sich aus den folgenden Eigenschaften zusammen:

N-1-dimensional: Für das Sortieren mit der SOM werden nicht alle n-Dimensionen benötigt, sondern nur n-1-Dimensionen. Die Dimensionalität einer Fitnesslandschaft setzt sich aus der Anzahl der Variablen und dem Fitnesswert zusammen. Besitzt ein Optimierungsproblem fünf Variablen, ist die Dimension der Fitnesslandschaft $n = 6$. Für die SOM werden allerdings nur die Variablen benötigt, also $n - 1 = 5$ Dimensionen.

Gruppierung: Obwohl man bei einer SOM im Allgemeinen von Sortieren oder Organisieren spricht, entspricht das Ergebnis eher einer Gruppierung der Inputdaten. Der Grund ist die nichtlineare Skalierung der Abstände, wodurch sich ähnliche Variablensets als Gruppe auf der Karte wiederfinden.

Da beide oben genannten Eigenschaften den Visualisierungsprozess recht genau beschreiben und zudem keine herkömmliche Visualisierungsmethode diese Eigenschaften aufweist, eignet sich der Namen *n-1-dimensionale Gruppierung* als Beschreibung der neuen Methode.

5.3 Vor- und Nachteile

Jede Visualisierungsmethode hat Vor- und Nachteile, wodurch sich unterschiedliche Methoden je nach Einsatzgebiet und Zweck besser oder schlechter eignen. Die Vor- und Nachteile der n-1-dimensionalen Gruppierung als Methode zur Visualisierung werden in diesem Absatz ausführlich erläutert.

Vorteile

Ein Vorteil und gleichzeitig ein Grund warum diese neue Methode im Rahmen dieser Arbeit entwickelt wurde, ist die geringe Rechenleistung im Vergleich zu anderen Visualisierungsmethoden. Unabhängig der Methode wird der größte Anteil an der Rechenleistung für das Reduzieren der Dimensionen benötigt. Dabei ist die Dimensionsreduktion durch eine selbstorganisierende Karte im Vergleich zu Methoden wie der multidimensionalen Skalierung sehr effizient und reduziert die benötigte Rechenzeit für den gesamten Visualisierungsprozess um ein Vielfaches. Ein willkommener Nebeneffekt ist

dabei die einfache Handhabung der selbstorganisierenden Karten. Übliche Visualisierungsmethoden erfordern oft die Bedienung komplexer Rechenprogramme und eine aufwendige Aufbereitung der Daten. Für die Methode der n-1-dimensionalen Gruppierung gilt das nicht.

Ein weiterer und fast noch wichtigerer Vorteil der neuen Methode ist die Fähigkeit, zwischen Inputdaten zu interpolieren. Bildlich gesprochen bedeutet das, dass auch Daten links, rechts oder zwischen den bekannten Input-Daten berücksichtigt werden. Welchen Vorteil das für die Visualisierung von Optimierungsproblemen hat, wird im Folgenden anhand der Fitesslandschaft eines fiktiven Optimierungsproblems demonstriert.

Für die Fitnesslandschaft des fiktiven Problems wird angenommen, dass die Typologie der Landschaft über den gesamten Variablenraum bekannt ist. Das Optimierungsproblem weist folgende Eigenschaften auf:

- Ein globales Maximum
- Ein globales Minimum
- Stetiger Variablenraum

Des Weiteren ist bekannt, welchen „Weg“ der Optimierungsalgorithmus während des Optimierungsprozesses auf der Landschaft zurückgelegt hat. In Abbildung 27 ist dieser Weg als rote Linie auf der Fitnesslandschaft dargestellt. Punkt A ist ein vom Optimierungsalgorithmus zufällig ausgewählter Startpunkt. Ausgehend vom Startpunkt A wurde ein Weg gesucht, der stets bergauf führt, also immer eine positive Steigung aufweist. Das Ende des Wegs und damit der Hochpunkt der Fitnesslandschaft ist durch Punkt B gekennzeichnet. Auf der Landschaft gibt es keinen Punkt, der höher liegt. Somit existiert auch kein Weg, ausgehend von Punkt B, der eine positive Steigung besitzt. Für den Optimierungsalgorithmus ist damit die Suche beendet. Das Optimum des Problems ist gefunden.

Punkt C liegt im Tal der Fitnesslandschaft und markiert ein Minimum der Fitnessfunktion. Er wurde bei der Suche des Maximums von dem Optimierungsalgorithmus allerdings nicht beachtet, da die Suche bei Punkt A begann und von dort stets bergauf führte.

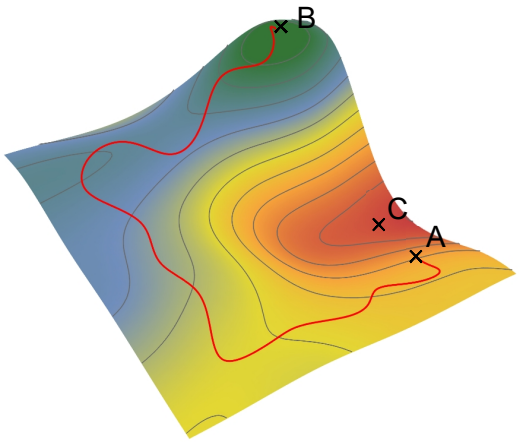


Abbildung 27: Gesamte Fitnesslandschaft

Die Suche nach dem Weg mit der größten Steigung verläuft natürlich nicht so exakt und zielstrebig wie es die rote Linie in Abbildung 27 suggeriert. Der Optimierungsalgorithmus tastet auch Regionen links und rechts des Weges ab, um auszuschließen, dass an einer anderen Stelle ein Weg mit einer noch größeren Steigung existiert.

In Abbildung 28 wird das oben beschriebene Suchmuster des Algorithmus deutlich. Es wurden alle Regionen zwischen Startpunkt A und Zielpunkt B abgetastet, solange sie eine positive Steigung aufweisen. Das Tal der Fitnesslandschaft, also die Region um Punkt C, wurde nicht beachtet. Dieses Verhalten ist allerdings kein Fehler des Optimierungsalgorithmus, sondern genau so gewollt. Das Ziel des Algorithmus ist es, schnell und effizient die optimale Lösung des Problems zu finden, ohne dabei alle möglichen Lösungen zu berechnen. Genau dieses Ziel wurde von dem Algorithmus streng verfolgt, indem er das Tal der Fitnesslandschaft bei der Suche ausgelassen hat.

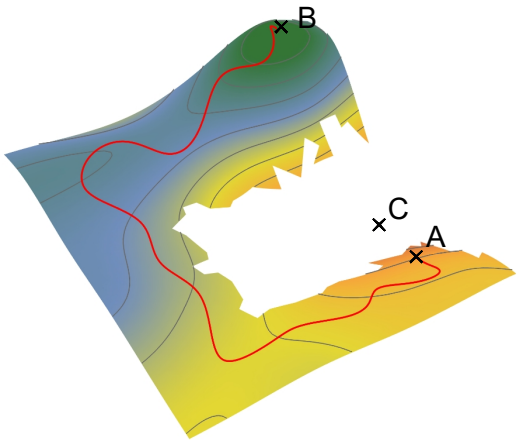


Abbildung 28: Vom Optimierungsalgorithmus abgetastete Regionen

Zur Visualisierung der Fitnesslandschaft werden, unabhängig von der Methode, die Variablensets während des Optimierungsprozesses aufgezeichnet. Wie aus Abbildung 28 hervorgeht, fehlen nach der Optimierung die Variablensets um den Punkt C.

Wird die Fitnesslandschaft mit einer herkömmlichen Visualisierungsmethode erstellt, können nur die Variablensets dargestellt werden, die zuvor aufgezeichnet wurden. Abbildung 29 zeigt eine so erstellte Landschaft. Es ist sowohl Punkt A, Punkt B als auch der Weg des Algorithmus zu erkennen, allerdings ist die Region um Punkt C in der Landschaft nicht vertreten. Wie bereits erwähnt, macht das für das Ergebnis der Optimierung keinen Unterschied, für die Analyse des Optimierungsproblems allerdings schon. Denn das globale Minimum, das Tal in der Fitnesslandschaft, ist in Abbildung 29 nicht zu erkennen. Man würde zu dem Schluss kommen, dass das Problem ein globales Maximum besitzt, jedoch kein globales Minimum. Das Minimum ähnelt eher einer großen Ebene, auf der mehrere ähnlich gleichschlechte Lösungen angeordnet sind. Geht man fälschlicherweise davon aus, dass ein Problem kein eindeutiges Minimum besitzt, kann das zu einer Fehleinschätzung oder Fehlinterpretation der Optimierungsergebnisse führen.

Abbildung 30 zeigt die Fitnesslandschaft zum gleichen Optimierungsproblem, allerdings wurde diese mit der Methode der n-1-dimensionalen Gruppierung erstellt. Während des Iterationsprozesses der SOM wurden die fehlenden Variablensets durch die Interpolation der vorhandenen Variablensets nachträglich erstellt. Beispielsweise entsteht durch die Interpolation der Variablensets von Punkt A und Punkt B ein Variablenset in der Region von Punkt C. Durch die anschließende Berechnung der Fitnesswerte kann die Typologie der bisher nicht sichtbaren Bereiche sichtbar gemacht werden. Siehe Abbildung 30.

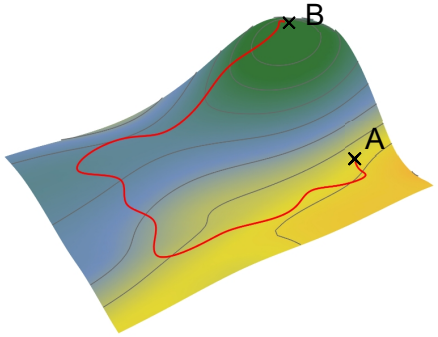


Abbildung 29: Visualisierung durch eine herkömmliche Methode

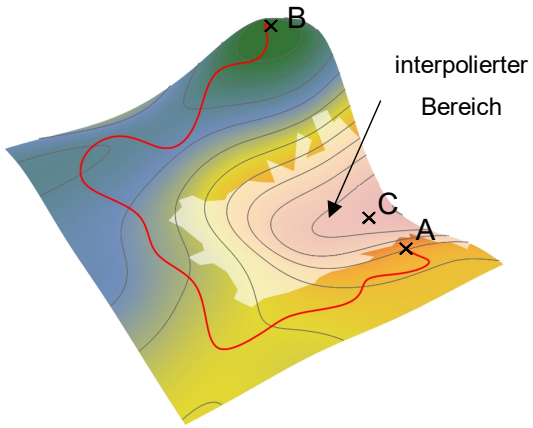


Abbildung 30: Visualisierung mit der Methode der n-1-dimensionalen Gruppierung

Werden die beiden obenstehenden Fitnesslandschaften miteinander verglichen, kann durchaus festgehalten werden, dass die Methode der n-1-dimensionalen Gruppierung eine qualitativ bessere Fitnesslandschaft liefert als herkömmliche Visualisierungsmethoden. Eine bessere und vor allem genauere Analyse eines Optimierungsproblems hilft bei der Beurteilung und Verifizierung der Ergebnisse, was ein wichtiger Bestandteil vieler ingenieurtechnischer

Aufgaben ist. Ein Ergebnis, das nicht verifizierbar oder überprüfbar ist, hat keine echte Relevanz. Der Vergleich zeigt des Weiteren, dass bei einer herkömmlichen Visualisierungsmethode die Typologie der Landschaft von der Wahl des Optimierungsalgorithmus abhängig ist. Wird ein Problem mit mehreren Algorithmen untersucht, muss für jede Untersuchung eine neue Fitnesslandschaft erstellt werden. Die neue Methode ist in diesem Punkt durch die Interpolation der Daten unabhängiger.

Nachteile

Obwohl das Interpolieren von Variablensets ein großer Vorteil ist, ist die Methode der n-1-dimensionalen Gruppierung keine Garantie für eine „vollständigere“ Darstellung der Fitnesslandschaft. Es kann nur sehr schwer kontrolliert werden, ob und welche Bereiche interpoliert wurden. Eine weitere Schwäche ist, dass naturgemäß nur zwischen Werten interpoliert werden kann. Variablensets, die komplett außerhalb des aufgezeichneten Bereichs liegen, können auch mit dieser neuen Methode nicht zuverlässig dargestellt werden.

5.4 Beispiel

Die Visualisierung einer Fitnesslandschaft mit der n-1-dimensionalen Methode wird anhand des folgenden Optimierungsbeispiels gezeigt. Die Ausgangssituation ist ein beidseitig gelenkig gelagerter Einfeldträger, welcher mit einer konstanten Linienlast belastet wird. Als Trägerquerschnitt ist ein geschweißtes Doppel-T-Stahlbauprofil der Güte S235 vorgesehen. In Abbildung 31 sind das statische System, die Belastungssituation und der Trägerquerschnitt dargestellt (nicht maßstäblich).

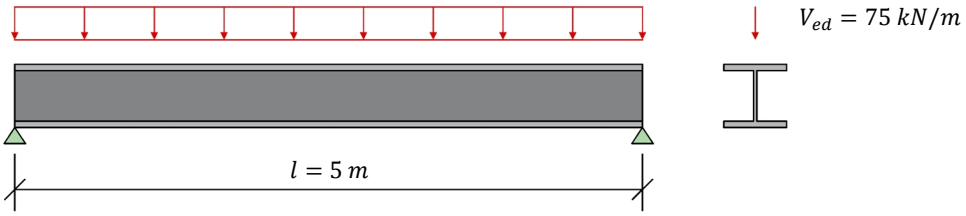


Abbildung 31: Statisches System und Belastungssituation

Eigenschaften

Optimierungsziel

Der Einfeldträger soll unter Berücksichtigung der Belastung und der konstanten Länge von $l = 5\,m$ hinsichtlich des Trägereigengewichts optimiert werden. Ein möglichst geringes Eigengewicht bedeutet einen geringeren Materialbedarf. Diese Art von Träger wird häufig als Dachbinder oder Pfetten in Hallenkonstruktionen eingesetzt. Aufgrund der hohen Anzahl an

Trägern in einer solchen Konstruktion kann bereits eine geringe Gewichtsersparnis an nur einem Trägertyp eine enorme Materialeinsparung für das gesamte Tragwerk bedeuten.

Der Fitnesswert entspricht dem Trägereigengewicht.

Variablen

Um das Optimierungsziel zu erreichen, sollen die Profilabmessungen des Trägers variiert werden. Zur Auswahl stehen nicht die Doppel-T-Profile aus der Eurocode Normfamilie, sondern ein geschweißtes Profil mit variablen Blechabmessungen. Das Doppel-T-Profil wird durch folgende Abmessungen definiert:

h	Profilhöhe
b	Flanschbreite
t	Flanschdicke
s	Stegdicke

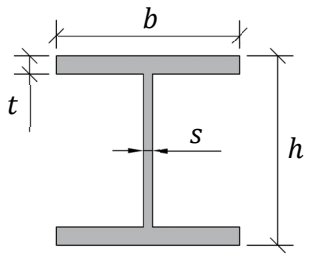


Abbildung 32: Profilabmessungen

Es ist allerdings nur die Profilhöhe, die Flanschbreite und Flanschdicke variabel. Die Stegdicke s verändert sich in Abhängigkeit der Profilhöhe. So wird gewährleistet, dass der Steg immer den Grenzwert der Querschnittsklasse 2 einhält. Damit auch der Flansch den Grenzwert der Querschnittsklasse 2 einhält, sind beliebige Kombinationen der Flanschbreite b und der Flanschdicke t nicht zulässig. Bei zunehmender Flanschbreite erhöht sich prozentual auch die Flanschdicke.

Da die Stegdicke von der Profilhöhe linear abhängig ist, wird die Stegdicke nicht als eigenständige Variable gewertet. Somit ist das ein Optimierungsproblem der Dimension 3.

Variablenraum

Folgende Intervalle sind zulässig:

Profilhöhe	h:	*100 – 300 mm*	
Flanschbreite	b:	*80 – 300 mm*	
Flanschdicke	t:	*10 – 35 mm*	
Stegdicke	s:	*7 – 20 mm*	*In Abhängigkeit der Profilhöhe*

Grenzen

Ein Profilquerschnitt hinsichtlich des Eigengewichts zu optimieren scheint zunächst trivial zu sein. Die kleinsten Querschnittsabmessungen resultieren im geringsten Trägereigengewicht. Allerdings wäre dieser Träger höchstwahrscheinlich nicht realisierbar, da Spannungs- und Verformungsgrenzen nicht eingehalten werden. Aus diesem Grund wird für dieses Optimierungsproblem die Verformungsgrenze d_{max} eingeführt. Die maximale Verformung in der Feldmitte darf höchstens $l/250$ betragen, also $5.000\,mm/250 = 20\,mm$. Ist die Verformung von $20\,mm$ erreicht, wird der Fitnesswert beziehungsweise das Trägereigengewicht erhöht. In welchem Maße der Fitnesswert erhöht wird, bestimmt die Penalty-Funktion.

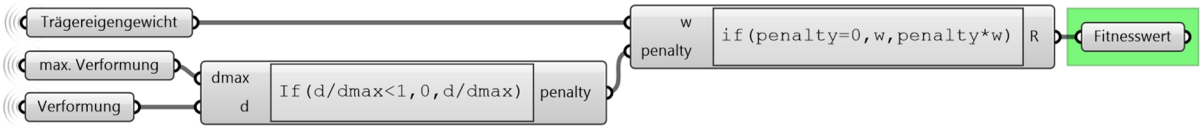


Abbildung 33: Definition der Penalty-Funktion

Zunächst wird der Penaltyfaktor, der Quotient aus Verformung d und maximal zulässiger Verformung d_{max} gebildet. Ist dieser Faktor kleiner gleich 1, ist der Fitnesswert gleich dem Trägereigengewicht. Ist der Faktor größer als 1, wird das Trägereigengewicht mit dem Penaltyfaktor multipliziert.

Optimierungsergebnis

Der Optimierungsprozess lieferte folgendes Ergebnis.

Solver	Fitnesswert	Verformung	Iterationen	Zeit
Galapagos (GA)	3,47 kN 0,694 kN/m	20,1 mm +0,5%	8000	≈ 6‘ 30“

Folgende Abbildung zeigt die Abmessungen des Trägerquerschnitts, der unter den gegebenen Randbedingungen den geringsten Materialbedarf aufweist und gleichzeitig die Verformungsgrenze von 20 mm um nur 0,5 % überschreitet.

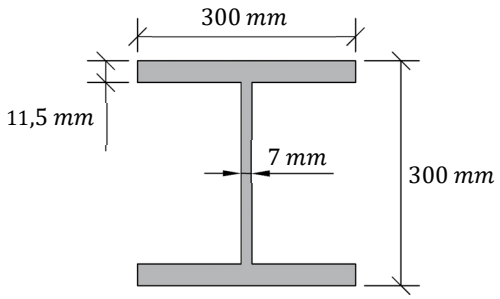


Abbildung 34: Optimale Querschnittsabmessungen

Ohne diesen Optimierungsschritt durchgeführt zu haben, würde im Zuge eines statischen Nachweises die Profilwahl vermutlich auf das warmgewalzte Normprofil HE-A 300 fallen (DIN 1025-3: [15]), denn das Profil kommt den obenstehenden Abmessungen am nächsten. Allerdingst wiegt ein HE-A 300 Profil 0,883 kN/m statt 0,694 kN/m, was einer Gewichts- und Materialkostenzunahme von über 27 % entspricht. Gegen die Wahl des optimierten Profils spricht der Mehraufwand, der bei der Herstellung eines Schweißprofils entsteht. Bei einer hinreichenden Anzahl an Trägern kann dieser jedoch ausgeglichen werden.

Visualisierung der Fitnesslandschaft

Die Fitnesslandschaft wurde nach dem in Absatz 5.1 beschrieben Visualisierungsprozess mit der Methode der n-1-dimensionalen Gruppierung erstellt.

Für den Visualisierungsprozess wurden die Variablensets während der Optimierung aufgezeichnet. Bei den oben genannten 8.000 Iterationsschritten entstehen auch 8.000 Variablensets. Während jeder Optimierungsalgorithmus zu Beginn des Optimierungsprozesses die Variablen noch stark variiert, sozusagen mit großen Schritten über die Fitnesslandschaft läuft, ändern sich die Variablensets gegen Ende des Prozesses nur noch minimal. Aus diesem Grund sind viele der 8.000 Variablensets bis auf die zweite oder dritte Nachkommastelle identisch und müssen nicht zwingend für die Visualisierung herangezogen werden. Bei diesem Beispiel wurden deshalb alle Werte der Variablensets auf die erste Nachkommastelle gerundet und alle bis auf eines der identischen Sets gelöscht. Nach diesem Vorgang sind noch 300 eindeutige Variablensets übrig, die in Form von Knoten auf der Input-Ebene einer SOM präsentiert werden. Die Output-Ebene der SOM wurde auf die fünffache Größe der Input-Ebene festgelegt, was einer Kartengröße von 1.500 Neuronen entspricht. Im Anschluss an den Iterationsprozess der SOM wurden die Fitnesswerte zu allen 1.500 Variablensets berechnet und auf der Hochachse abgetragen. Damit das beste Ergebnis, also der kleinste Fitnesswert, in der Landschaft als Hochpunkt dargestellt wird,

wurden alle Fitnesswerte mit -1 multipliziert. Diese Art der Darstellung dient der besseren Lesbarkeit der Landschaft, da Hochpunkte intuitiv als ein Optimum wahrgenommen werden.

Ein weiterer, bisher nicht diskutierter Schritt ist das Skalieren der schlechten Fitnesswerte. Gerade bei der Verwendung von Penalty-Funktionen ist das wichtig, da ansonsten das Verhältnis zwischen guten und schlechten Lösungen möglicherweise verfälscht wird. Durch das Multiplizieren von Fitnesswerten mit dem Penaltyfaktor können einzelne Werte so stark erhöht werden, dass die Spanne zwischen den Extremwerten für eine realistische Darstellung zu groß wird. In diesem Beispiel wurden alle Fitnesswerte größer als 10 kN gekappt beziehungsweise auf die 10 kN Ebene skaliert. Abbildung 35 zeigt die unskalierte Fitnesslandschaft mit der 10 kN Ebene. In der Landschaft mit unskalierten Extremwerten sind keine Hochpunkte, geschweige denn Details zu erkennen. Es sind lediglich eine Ebene mit einer Vielzahl an guten Lösungen (grün) und tiefe Täler mit schlecht Lösungen (rot) zu erkennen.

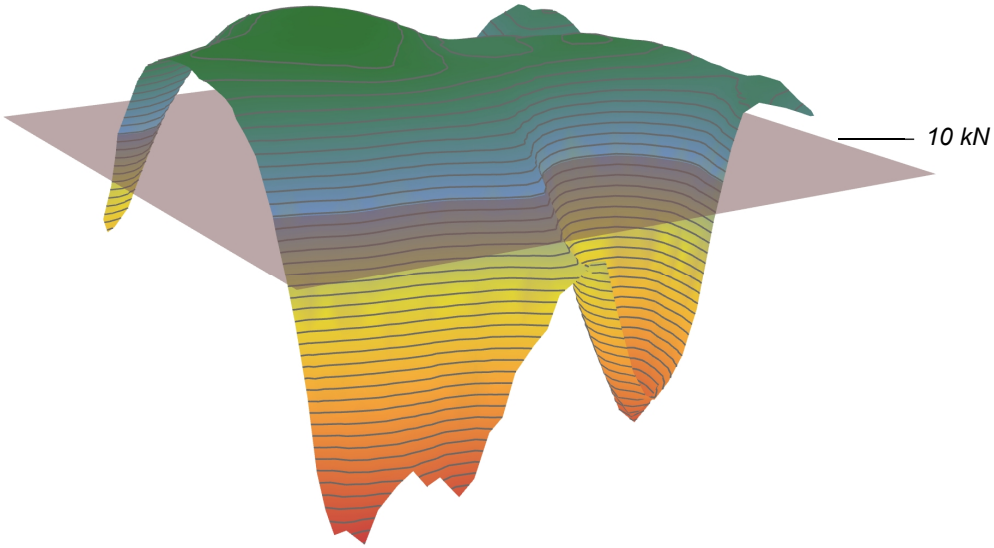


Abbildung 35: Isometrische Darstellung der nicht skalierten Fitnesslandschaft

Abbildung 36 zeigt die gleiche Fitnesslandschaft, allerdings mit skalierten Extremwerten. Nachdem alle Fitnesswerte schlechter als 10 kN auf eine Ebene verschoben wurden, sind in der Landschaft Details wie Gipfel, Ebenen, Schluchten und Täler zu erkennen. Durch die Skalierung gehen zwar Informationen über die schlechtesten Lösungen verloren, allerdings werden nur so Informationen über die guten Lösungen sichtbar.

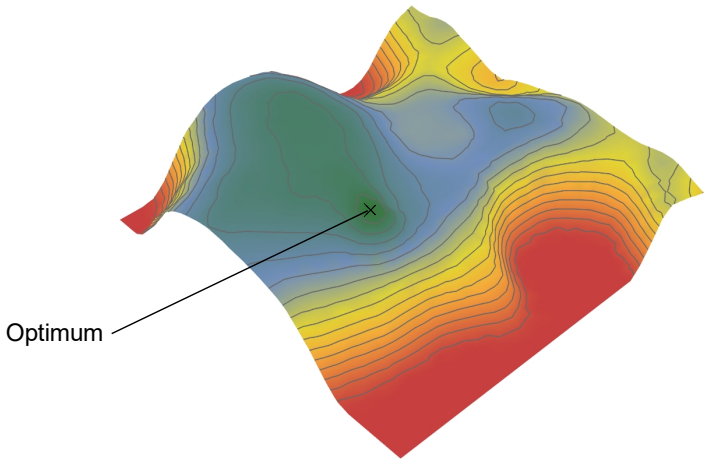


Abbildung 36: Isometrische Darstellung der skalierten Fitnesslandschaft

Die Fitnesslandschaft weist einen absoluten Gipfel (Hochpunkt) auf, welcher das beste Ergebnis der Optimierung repräsentiert. Des Weiteren ist eine relativ große Ebene direkt unterhalb des Gipfels zu sehen, was darauf schließen lässt, dass neben dem Optimum noch weitere ähnlich gute Lösungen existieren. Die rot eingefärbten Ebenen in den Tälern der Landschaft repräsentieren die schlechten Optimierungsergebnisse. Hier befinden sich die Lösungen, die entweder ein deutlich höheres Gewicht oder eine zu große Verformung aufweisen. Eine detaillierte Analyse der Fitnesslandschaft findet in folgendem Absatz statt.

Analyse

Durch die Analyse der Ergebnisse und der Kategorisierung des Problems können Lösungen gefunden werden, die zwar nicht dem absoluten Optimum entsprechen, aber es dennoch wert sind, genauer untersucht zu werden. Bei der Analyse geht es im Prinzip darum zu erkennen, welche Querschnittsgeometrien, abgesehen von der optimalen Lösung, eventuell noch in Frage kommen. Für diese Untersuchung werden einige interessante und markante Punkte auf der Landschaft genauer betrachtet. Hierfür werden zusätzlich zur Fitnesslandschaft auch die Variablensets visualisiert.

In diesem Beispiel besteht ein Variablenset aus den Querschnittsparametern Profilhöhe, Flanschbreite und Flanschdicke. Um die Variablensets darstellen zu können, wird auf der Landschaft für jedes Set ein Rechteck mit drei Freiheitsgraden projiziert. Es werden in gleicher Anzahl Rechtecke benötigt wie die Landschaft Variablensets besitzt. In diesem Fall sind das 1.500 Rechtecke. Die drei Freiheitsgrade bestehen aus den beiden Kantenlängen und der Rotation um die eigene Achse. Abbildung 37 zeigt den Zusammenhang zwischen Variablenset und Rechteckgeometrie.

h .. Profilhöhe
b .. Flanschbreite
t .. Flanschdicke

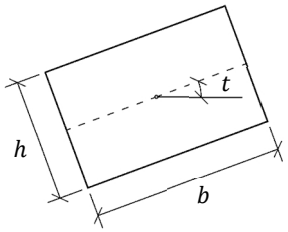


Abbildung 37: Darstellung der Variablensets

In Abbildung 38 ist die Fitnesslandschaft mit allen 1.500 Variablensets dargestellt. Zur Analyse des Optimierungsproblems wurden insgesamt acht Variablensets zur näheren Betrachtung ausgewählt. Die Sets 1, 2 und 3 decken die Bereiche der guten und sehr guten Lösungen ab. Hier ist es interessant herauszufinden, ob neben der optimalen Lösung (Set 1) noch weitere akzeptable Lösungen existieren. Die Sets 4, 5, 6, 7 und 8 liegen in den mittleren bis schlechten Bereichen. Hier gilt es zu bestimmen, welche Querschnittsabmessungen auf keinen Fall in Frage kommen.

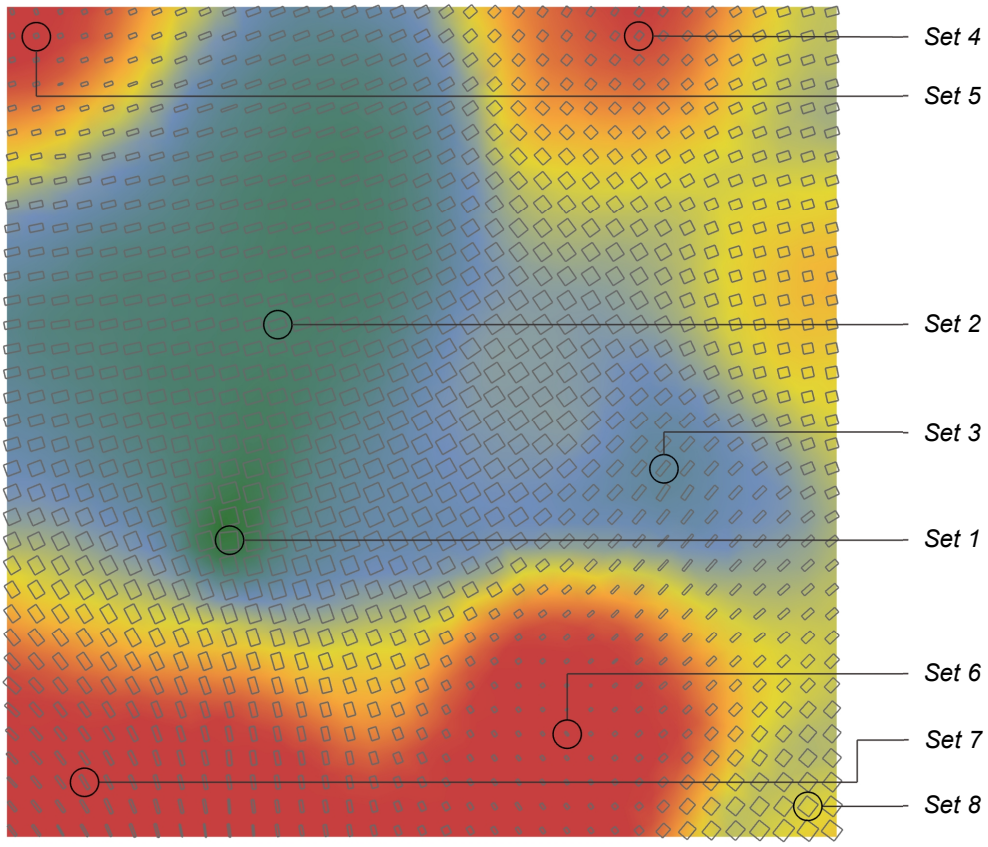


Abbildung 38: Grundriss der Fitnesslandschaft mit den entsprechenden Variablensets

In folgender Tabelle werden die zuvor ausgewählten Sets ausgewertet. Es werden jeweils die Profilabmessungen sowie der daraus resultierende Fitnesswert und die Verformung inklusive der Abweichung zum Optimum angegeben.

Tabelle 1: Analyse ausgewählter Variablensets

Set	Variablen			Fitnesswert	Verformung
	Profilhöhe h	Flanschbreite b	Flanschdicke t		
1	*300 mm*	*300 mm*	*11,5 mm*	3,47 kN ± 0 %	20,1 mm ≈ 0 %
2	*286 mm*	*190 mm*	*17 mm*	3,95 kN[p)] + 13,8 %	24,3 mm + 21,5 %
3	*277 mm*	*160 mm*	*31 mm*	4,54 kN + 30,8 %	20,1 mm ≈ 0 %
4	*190 mm*	*165 mm*	*30 mm*	9,98 kN [p)] + 188 %	45,8 mm + 129 %
5	*163 mm*	*121 mm*	*17,7 mm*	12,46 kN [p)] + 259 %	121 mm + 509 %
6	*131 mm*	*100 mm*	*27 mm*	22,48 kN [p)] + 548 %	192 mm + 860 %
7	*144 mm*	*257 mm*	*22,8 mm*	16,72 kN [p)] + 382 %	68 mm + 241 %
8	*255 mm*	*229 mm*	*32,5 mm*	6,37 kN + 83,8 %	17,2 mm -13,9 %

[p)] Fitnesswert künstlich durch Penalty-Funktion erhöht

Set 1, 2, 3 und 8 weisen alle relativ gute Fitnesswerte und maximale Verformungen auf. Interessant ist, dass dabei die Profilhöhe nicht stark variiert, sie bewegt sich zwischen 255 und 300 mm. Dahingegen nutzt die Flanschbreite und Flanschdicke fast den gesamten Variablenraum aus. Die Flanschbreite variiert zwischen 160 und 300 mm, die Flanschdicke zwischen 11,5 und 32,5 mm. Bei den Sets 4, 5, 6 und 7 sind auch kleinere Profilhöhen vertreten, allerdings liegen auch die Fitnesswerte bis zu 548 % über der optimalen Lösung.

Aus dieser Analyse scheint sich ein interessantes Muster abzuleiten. Je geringer die Profilhöhe, desto größer müssen Flanschbreite und Flanschdicke sein. Ist die Profilhöhe zu gering, in diesem Fall unter 255 mm, kann keine akzeptable Lösung gefunden werden. Für den Tragwerksplaner bedeutet diese Information, dass er bei der Profilwahl zunächst Set 1 favorisiert. Falls beispielsweise keine 300 mm Einbauhöhe zur Verfügung stehen, kann er auf Kosten der Flanschbreite und Flanschdicke die Profilhöhe reduzieren. Diese Erkenntnis war natürlich vorherzusehen, da das Flächenträgheitsmoment I_y eines Doppel-T-Profils durch den Steineranteil im Quadrat mit der Profilhöhe steigt. Allerdings war dies auch nur ein Beispiel zur Erläuterung des Visualisierungsprinzips. Bei Optimierungsproblemen mit nicht eindeutig vorhersehbaren Ergebnissen, bietet die Visualisierung einen echten Mehrwert an Informationen. Es können Muster und Strukturen innerhalb des Problems erkannt werden, die es dem Tragwerksplaner erlauben, effizientere und nachhaltigere Lösungen zu finden.

6 Reduktion des Optimierungsproblems auf Grundlage der n-1-dimensionalen Gruppierung

Im Allgemeinen gilt: Je größer und komplexer ein Problem, desto schwieriger gestaltet sich die Optimierung. Der Variablenraum ist unter anderem eine maßgebende Kenngröße für die Komplexität und Größe eines Optimierungsproblems. Dabei gilt, je größer die Wertebereiche, desto mehr mögliche Variablenkombinationen, desto größer der Variablenraum. Aus diesem Grund gilt es, den Wertebereich jeder einzelnen Variablen so klein wie möglich zu halten. Jedoch bedeutet jede Eingrenzung des Wertebereichs auch eine Reduzierung der möglichen Lösungen. Optimal wäre es also, den Wertebereich so einzugrenzen, dass nur schlechte Lösungen wegfallen und alle guten bis sehr guten Lösungen weiterhin durch den Variablenraum vertreten werden. Ohne umfassende Informationen über das Optimierungsproblem ist das allerdings nur sehr schwer durchführbar.

Aus diesem Grund wurde im Rahmen dieser Arbeit ein Verfahren entwickelt, welches es dem Anwender erlaubt, den Variablenraum bereits vor dem Optimierungsprozess zu analysieren, und gegebenenfalls den Wertebereich einzelner Variablen einzuschränken. Die Grundidee zu diesem Verfahren entstand während der Ausarbeitung und Ausformulierung der Methode der n-1-dimensionalen Gruppierung. Neu ist, dass nicht nur ein Ausschnitt des gesamten Problems dargestellt wird, wie es bei der Visualisierung der Fitnesslandschaft der Fall ist, sondern der gesamte Variablenraum in abstrahierter Form.

Im folgenden Absatz findet zunächst eine kurze Einordnung der Thematik *große Optimierungsprobleme* statt. An einem kurzen Rechenbeispiel wird gezeigt, wie sich der Variablenraum auf die Problemgröße auswirkt. Daraufhin wird in Absatz 6.2 die Grundidee des Verfahrens dargestellt und in Absatz 6.3 durch ein Beispiel erläutert.

6.1 Problematik des „großen Optimierungsproblems“

Ein mathematisches Verfahren wird als Heuristik oder heuristische Methode bezeichnet, wenn in kurzer Zeit und mit geringem Rechenaufwand eine zulässige Lösung gefunden werden kann. In diesem Sinne sind alle Black-Box Optimierungsalgorithmen Heuristiken. Sie basieren auf Schätzungen, Beobachtungen, Annahmen und Vermutungen und kommen dann zum Einsatz, wenn eine exakte Lösung nicht berechnet werden kann. Das kann der Fall sein, wenn ein Problem so groß ist, dass die Berechnung aller Lösungen zu lange dauern würde oder

wenn es schlichtweg zu aufwändig wäre, alle möglichen Lösungen durchzuprobieren. Der Ansporn einer Heuristik ist es also, in kurzer Zeit eine möglichst gute Lösung eines ansonsten nicht lösbaren Problems zu finden.

Um zu verdeutlichen, wann ein Optimierungsproblem bereits zu groß ist, um exakt berechnet zu werden, wird folgendes Rechenbeispiel vorgestellt:

Es sollen die Querschnittsabmessungen eines Stahlbeton Hohlkastenquerschnitts so optimiert werden, dass das Brückenbauwerk unter allen maßgebenden Lastfällen die Grenzwerte der Tragfähigkeit und Gebrauchstauglichkeit einhält. Acht Querschnittsabmessungen können für die Optimierung variiert werden, welche sich aufgrund der Querschnittssymmetrie auf die Mittelachse beziehen, siehe Abbildung 39. Alle nicht markierten Abmessungen, wie beispielsweise der Längsneigungswinkel, sind fest vorgegeben. Jeder Variablen wird zunächst ein Basiswert zugewiesen, von dem der Optimierungsalgorithmus um maximal ± 10 cm abweichen darf. Als Schrittweite wird 1 cm festgelegt. Wird beispielsweise der Variable x_1 der Basiswert 400 cm zugewiesen, muss eine Lösung für x_1 zwischen 390 und 410 cm gesucht werden.

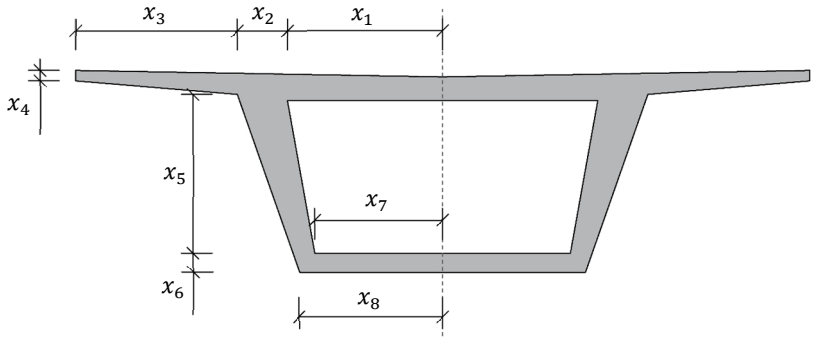


Abbildung 39: Hohlkastenquerschnitt: freie Optimierungsparameter

Um zu ermitteln, ob eine gewählte Querschnittsgeometrie alle Grenzwerte unter den gegebenen Bedingungen einhält, müssen zunächst die Querschnittswerte berechnet werden, mit denen im Anschluss eine FE-Berechnung des gesamten Systems durchgeführt wird. Wie viel Rechenzeit die Querschnitts- und FE-Berechnung in Anspruch nimmt, ist ohne ein konkretes Beispiel schwer zu sagen, jedoch sind ein bis fünf Sekunden pro Rechenschritt eine recht gute Annahme. Demzufolge lässt sich die benötigte Zeit für die Berechnung aller mögliche Querschnittsgeometrien wie folgt berechnen.

$$20^8 = 2{,}56 \cdot 10^{10} \rightarrow \textbf{25,6 Milliarden Möglichkeiten}$$

mit: Dimension / Variablen: 8
Werte je Variable: 20

$$1\,s \cdot 2{,}56 \cdot 10^{10} = 2{,}56 \cdot 10^{10} \text{ Sekunden}$$
$$2{,}56 \cdot 10^{10} / 60 \cdot 60 \cdot 24 \cdot 365 = \textbf{811,8 Jahre}$$

mit: Rechenzeit je Durchgang: 1 s (Annahme)

Das obige Rechenbeispiel zeigt recht anschaulich, dass die Optimierung des Hohlkastenquerschnitts nur mit Hilfe eines heuristischen Algorithmus durchführbar ist. Die Berechnung aller möglichen Hohlkastenquerschnitte ist aufgrund der hohen Rechenzeit von 811,8 Jahren unmöglich.

Doch auch heuristische Optimierungsmethoden stoßen bei unbegrenzt großen Lösungsräumen an ihre Grenzen. Die Verwendung von Black-Box Algorithmen ist kein Garant dafür, dass jedes Problem schnell und zufriedenstellend gelöst wird. Bei zu großen Lösungsräumen kann selbst die Rechenzeit eines Black-Box Algorithmus schnell mehrere Stunden oder Tage überschreiten. Hinzu kommt, dass zu große Probleme schnell unübersichtlich werden. Da Lösungen nichtlinearer Optimierungen nicht auf mathematischen Beweisen beruhen, müssen gefundene Lösungen immer auf Plausibilität überprüft werden. Eine Optimierungslösung ungeprüft zu übernehmen und weiterzuverwenden, wäre vor allem im Ingenieurwesen grob fahrlässig.

Der Umstand, dass viele Optimierungsergebnisse nur schwer zu verifizieren sind, ist vermutlich einer der Gründe, warum Black-Box Optimierungsmethoden bis heute im Ingenieursalltag kaum eingesetzt werden. Um das zu ändern, müssen die Probleme so weit verkleinert werden, dass zum einen die Rechenzeit verkürzt wird, und zum anderen ein Zusammenhang zwischen Variable und Ergebnis herzustellen ist. Dafür muss der Anwender, in diesem Fall der Tragwerksplaner verstehen, welche Variable welche Auswirkung auf das Tragverhalten der Struktur hat. Erreicht werden kann das nur, indem entweder die Dimension des Problems oder die Größe des Lösungsraums reduziert wird. Um die Dimension eines Problems zu reduzieren, müssten unbekannten Parametern konstante Werte zugewiesen werden. Da die Bestimmung dieser Werte der Grund für den Optimierungsprozess ist, wäre dieses Vorgehen nicht zielführend.

Im Gegensatz zur Dimension, kann der Variablenraum jedoch ohne Probleme reduziert werden, vorausgesetzt der Wertebereich der Variablen kann eingegrenzt werden. Der

Wertebereich richtet sich in der Regel nach den Randbedingungen der Struktur oder auch nach den Erfahrungswerten des Tragwerksplaners. Am Beispiel Holkastenquerschnitt (Abbildung 39 auf Seite 80) kann das anhand der Variable x_4 verdeutlicht werden. x_4 steuert die Dicke der Fahrbahnplatte am äußeren Rand und kann auf einen Wertebereich zwischen 20 und 40 cm begrenzt werden. Eine dünnere Fahrbahnplatte würde den Korrosionsschutzanforderungen nicht standhalten und eine Dicke von mehr als 40 cm kann aus wirtschaftlichen Gründen ausgeschlossen werden. Nach diesem Prinzip wird für jede Variable ein individueller Wertebereich festgelegt. Zusammen bilden sie den Variablenraum.

Um den Variablenraum des Optimierungsproblems zu reduzieren, müssen die Wertebereiche weiter eingegrenzt werden. Allerdings kann am Beispiel von Variable x_4 ohne weitere Informationen nicht bestimmt werden, ob eine optimale Lösung zwischen 20 - 30 cm, 30 - 40 cm oder 25 - 35 cm zu finden ist. In folgendem Absatz wird eine Methode vorgestellt, mit Hilfe derer eine Eingrenzung des Variablenraums bereits vor dem Optimierungsprozess vorgenommen werden kann.

6.2 Verfahren zur Eingrenzung des Variablenraums

Das Verfahren zur Eingrenzung des Variablenraums basiert auf der in Kapitel 5 vorgestellten Methode der n-1-dimensionalen Gruppierung. Die Grundidee ist, den Variablenraum eines Problems in Abhängigkeit der Fitnesswerte bereits vor dem Optimierungsprozess zu visualisieren, um den Variablenraum in gute und schlechte Wertebereiche einzuteilen. So können die schlechten Wertebereiche ausgeschlossen werden und nur mit den guten Wertebereichen die anschließende Optimierung durchgeführt werden.

Wie bereits mehrfach in dieser Arbeit diskutiert wurde, kann der Variablenraum beziehungsweise die Fitnesslandschaft eines mehrdimensionalen Optimierungsproblems nie vollständig visualisiert werden, da die Berechnung aller Lösungen nicht möglich ist. Um jedoch einen Variablenraum in gute und schlechte Wertebereiche einteilen zu können, muss er möglichst vollständig dargestellt werden. Um das zu erreichen, wird ein gleichmäßiges Raster über die Fitnesslandschaft gelegt und nur an den Schnittpunkten der Rasterlinien die Fitnesswerte ermittelt, siehe Abbildung 40. Durch das gleichmäßige Raster wird gewährleistet, dass alle Regionen der Fitnesslandschaft vertreten sind.

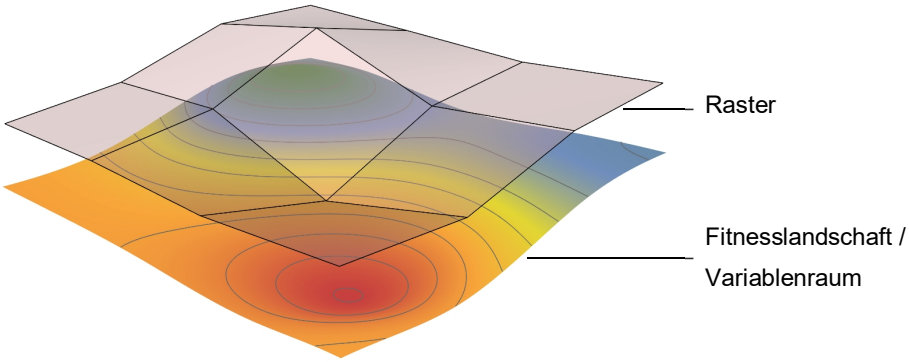


Abbildung 40: Abstraktion der Fitnesslandschaft

Um an den Schnittpunkten der Rasterlinien den Fitnesswert zu berechnen, müssen zunächst die Variablensets an diesen Punkten generiert werden. Bei n-dimensionalen Problemen ist die gleichmäßig über den Variablenraum verteilte Generierung von Variablensets nicht trivial. Der n-dimensionale Variablenraum entspricht einem räumlichen Polygon mit 2^n Ecken, wobei n der Dimension des Optimierungsproblems entspricht. 2^n, da eine Variable jeweils zwei Extremwerte einnehmen kann, die untere Grenze und die obere Grenze des Wertebereichs. Der Variablenraum eines zweidimensionalen Problems ist also mit 2^2 Extremwerten eine Ebene mit vier Ecken, siehe Abbildung 41. Variablenräume von Problemen höherer Dimension müssen sich räumlich vorgestellt werden. In der Geometrie werden räumliche Polygone als n-Simplex bezeichnet, wobei eine n-Simplex n+1 Ecken besitzt. Daraus folgt, dass der Variablenraum eines n-dimensionalen Problems durch eine n-1-Simplex dargestellt wird. Dementsprechend ist der Variablenraum eines vierdimensionalen Problems mit 2^4 Extremwerten ein 15-Simplex mit 16 Ecken.

Zweidimensionales Optimierungsproblem

Die Variablen wurden auf das Intervall [0,00;1,00] mit der Schrittweite 0,01 normalisiert

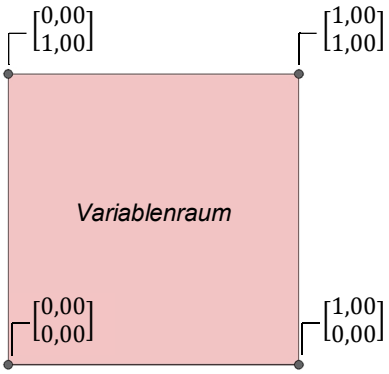


Abbildung 41: Normierter Variablenraum eines 2-D Optimierungsproblems

Wie in obiger Abbildung zu sehen ist, wird der Variablenraum durch die Extremwert-Variablensets begrenzt. Alle anderen Variablensets, in diesem Fall $100^2 = 10.000$, liegen innerhalb des Variablenraums, hier der rot markierte Bereich. Bei diesem zweidimensionalen Beispiel können alle Variablensets im Variablenraum gemäß eines zweidimensionalen

kartesischen Koordinatensystems angeordnet werden. In diesem Fall ist das Generieren eines Rasters und die Ermittlung der entsprechenden Variablensets eine einfach zu lösende Aufgabe. Siehe Abbildung 42.

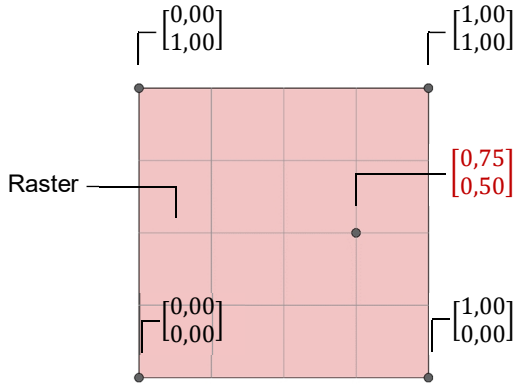


Abbildung 42: Anordnungsschema der Variablensets im Variablenraum

Die Anordnung der Sets gemäß eines 2-D Koordinatensystems ist jedoch ab der Dimension 3 nicht mehr möglich. Um mehrdimensionale Variablensets auf einer Ebene anzuordnen, müssen sie nach ihrer Ähnlichkeit sortiert werden. An dieser Stelle wird auf das Kapitel 4 verwiesen. Hier werden die Grundlagen zum Sortieren von mehrdimensionalen Daten ausführlich erklärt und an einem Beispiel erläutert.

Wie viele Sets generiert werden, hängt von der Dimension des Problems ab. Die Anzahl der generierten Sets bezeichnet man als Rasterauflösung. Sie wird als Quotient des gesamten Variablenraums angegeben. Eine Rasterauflösung von 1/10 bedeutet, dass jedes zehnte Set generiert wird. Bei Optimierungsproblemen höherer Dimension muss die Rasterauflösung auf bis zu 1/1.000 oder 1/10.000 herabgesetzt werden.

Für ein zweidimensionales Optimierungsproblem mit den normalisierten Variablen x_1 und x_2 werden die Variablensets gemäß folgender Matrix beispielhaft generiert.

Dimension: 2
Intervall: $[0{,}00; 1{,}00]$
Schrittweite: 0,01
Rasterauflösung: $1/10$
generierte Sets: $11^2 = 121$

x_1 / x_2	**0,0**	**0,1**	**0,2**	⋯	**1,0**
0,0	$\begin{bmatrix}0{,}0\\0{,}0\end{bmatrix}$	$\begin{bmatrix}0{,}1\\0{,}0\end{bmatrix}$	$\begin{bmatrix}0{,}2\\0{,}0\end{bmatrix}$	⋯	$\begin{bmatrix}1{,}0\\0{,}0\end{bmatrix}$
0,1	$\begin{bmatrix}0{,}0\\0{,}1\end{bmatrix}$	$\begin{bmatrix}0{,}1\\0{,}1\end{bmatrix}$	$\begin{bmatrix}0{,}2\\0{,}1\end{bmatrix}$	⋯	$\begin{bmatrix}1{,}0\\0{,}1\end{bmatrix}$
0,2	$\begin{bmatrix}0{,}0\\0{,}2\end{bmatrix}$	$\begin{bmatrix}0{,}1\\0{,}2\end{bmatrix}$	$\begin{bmatrix}0{,}2\\0{,}2\end{bmatrix}$	⋯	$\begin{bmatrix}1{,}0\\0{,}2\end{bmatrix}$
⋮	⋮	⋮	⋮	⋱	⋮
1,0	$\begin{bmatrix}0{,}0\\1{,}0\end{bmatrix}$	$\begin{bmatrix}0{,}1\\1{,}0\end{bmatrix}$	$\begin{bmatrix}0{,}2\\1{,}0\end{bmatrix}$	⋯	$\begin{bmatrix}1{,}0\\1{,}0\end{bmatrix}$

Die generierten Sets werden anschließend einer selbstorganisierenden Karte (SOM) als Input präsentiert. Der Ablauf einer SOM Organisation wird in Absatz 4.2.2 erläutert und an dieser Stelle nicht weiter beschrieben.

Durch die Organisation mit Hilfe der selbstorganisierenden Karte wird Folgendes erreicht:

- Die generierten Variablensets werden entsprechend ihren Ähnlichkeiten auf einer 2-D Ebene sortiert angeordnet.
- Zwischen den generierten Variablensets werden weitere Sets erzeugt. Die Werte der neuen Sets werden durch lineare Interpolation der generierten Sets bestimmt.

Abbildung 43 zeigt links einen beliebigen, n-dimensionalen Variablenraum, dargestellt als n-1-Simplex. In dem n-dimensionalen Variablenraum befinden sich die generierten, gleichmäßig verteilten Variablensets. Rechts ist der Variablenraum nach der Organisation durch eine SOM dargestellt. Zwischen den generierten Variablensets (blau) befinden sich die interpolierten Variablensets (rot).

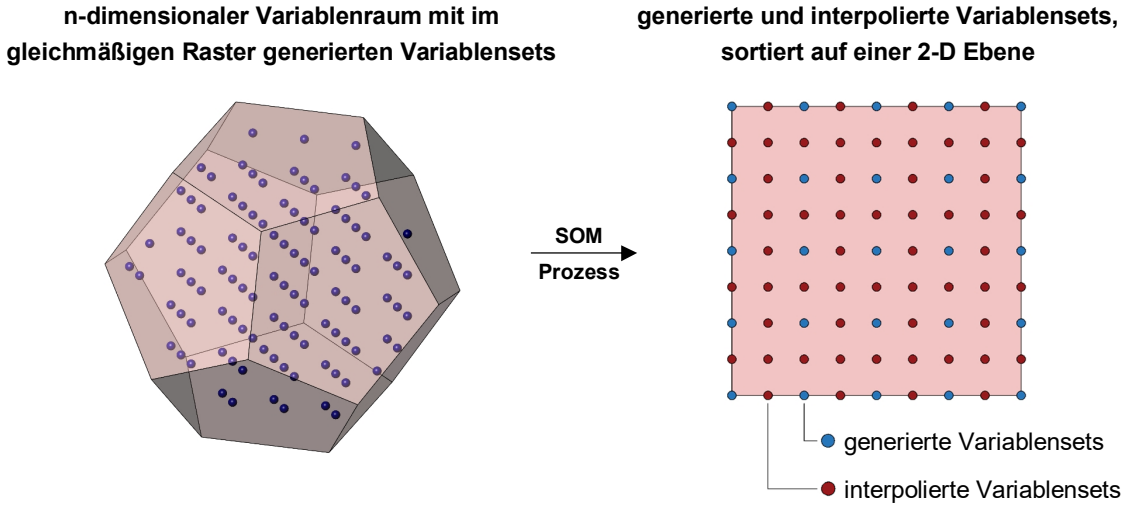


Abbildung 43: Organisation der n-dimensionalen Variablensets mit Hilfe einer SOM

Durch die im Raster generierten Variablensets und die anschließende Organisation durch den SOM Algorithmus, wird der gesamte Variablenraum in abstrahierter Form repräsentiert.

Um den Variablenraum zu analysieren und gegebenenfalls die Wertebereiche eingrenzen zu können, müssen die Sets in Abhängigkeit der Fitnesswerte dargestellt werden. Dafür erfolgt zunächst die Berechnung aller generierten und interpolierten Variablensets. Im Anschluss werden die berechneten Fitnesswerte dem jeweiligen Variablenset zugeordnet. Dargestellt

wird der Variablenraum als Raster aus farbigen Quadraten, wobei jedes Quadrat ein Variablenset repräsentiert und die Farbe den Fitnesswert wiederspiegelt.

In der farbigen Karte können schnell und intuitiv gute und schlechte Regionen definiert werden, siehe Abbildung 44. Durch die Analyse dieser Regionen beziehungsweise durch die Analyse der Variablensets in diesen Regionen, kann der Wertebereich jeder einzelnen Variablen in gute und schlechte Bereiche eingeteilt werden.

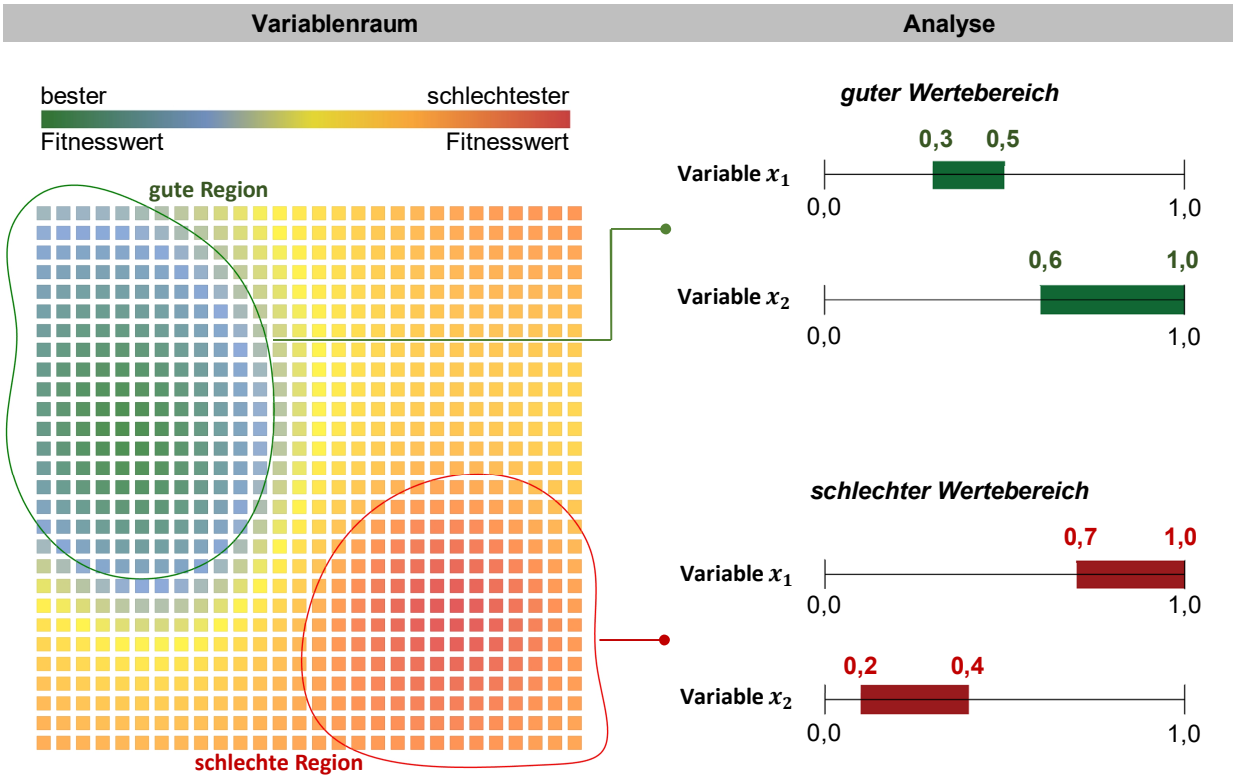


Abbildung 44: Analyse des Variablenraums eines 2-D Optimierungsproblems

Aufgrund der Visualisierung und der anschließenden Analyse des Variablenraums können die Wertebereiche eingegrenzt und damit der Variablenraum und das Optimierungsproblem verkleinert werden. Bei der Eingrenzung des Wertebereichs können grundsätzlich zwei Strategien verfolgt werden:

1. Es wird ausschließlich der gute Wertebereich für den Optimierungsprozess verwendet.
2. Es wird der gesamte Variablenraum abzüglich des schlechten Wertebereichs für den Optimierungsprozess verwendet.

Mit Strategie 1 läuft man Gefahr, dass mittelgute Bereiche vom Optimierungsprozess ausgeschlossen werden, in denen sich eventuell lokale Optima befinden, die aufgrund ihrer geringen Ausbreitung in der visuellen Darstellung des Variablenraums nicht erfasst wurden.

Wählt man Strategie 2, werden auch die mittelguten Bereiche für den Optimierungsprozess verwendet, wodurch sich die Problemgröße nicht um das Maß reduziert, welches mit Strategie 1 möglich wäre.

Welche Strategie bei der Eingrenzung des Wertebereichs angewendet wird, hängt von der Beschaffenheit des Variablenraums ab. Weist der Variablenraum eindeutig einen guten und einen schlechten Bereich auf, wie es in Abbildung 44 der Fall ist, kann Strategie 1 gewählt werden. Sind jedoch mehrere gute und schlechte Bereiche zu erkennen oder die Grenzen dazwischen nicht eindeutig, liegt man mit Strategie 2 auf der sicheren Seite.

Im Fall von Abbildung 44 wird also ausschließlich der gute Wertebereich für den anschließenden Optimierungsprozess verwendet. So kann die Größe des Optimierungsproblems von 10.000 möglichen Lösungen um 92 % auf 800 mögliche Lösungen reduziert werden.

Tabelle 2: Vergleich: Problemgröße vor und nach der Eingrenzung des Variablenraums

Optimierungsproblem	Ursprüngliche Porblemgröße	Porblemgröße nach der Eingrezung des Variablenraums	Reduktion
Dimension: 2 Intervall: $[0{,}00; 1{,}00]$ Schrittweite: $0{,}01$	Variable $x_1: 0{,}00 - 1{,}00$ Variable $x_2: 0{,}00 - 1{,}00$ **Größe: $\mathbf{100 \cdot 100 = 10.000}$**	Variable $x_1: 0{,}30 - 0{,}50$ Variable $x_2: 0{,}60 - 1{,}00$ **Größe: $\mathbf{20 \cdot 40 = 800}$**	- 92 %

In folgendem Absatz wird das Verfahren zur Eingrenzung des Variablenraums anhand eines Optimierungsbeispiels erläutert.

6.3 Beispiel

Als Erläuterungsbeispiel dient an dieser Stelle das gleiche Optimierungsproblem wie in Absatz 5.4. Aus diesem Grund werden hier nur die Randbedingungen des Optimierungsproblems kurz wiederholt, eine ausführliche Beschreibung der Ausganssituation, des Variablenraums, der Grenze und der Penalty-Funktion ist in Absatz 5.4 zu finden.

Es soll ein beidseitig gelenkig gelagerter Einfeldträger mit einer konstanten Belastung hinsichtlich des Trägereigengewichts optimiert werden. Dabei kann die Profilhöhe, die Flanschbreite und Flanschdicke des Doppel-T-Profils variiert werden, alle anderen Profilabmessungen sowie die Trägerlänge und die Lagerbedingungen sind konstant. Begrenzt ist das Optimierungsproblem durch die maximale Verformung d_{max} von $l/250$. Eine Verletzung der Grenze wird durch die Erhöhung des Fitnesswertes durch eine Penalty-Funktion bestraft.

In Abbildung 45 ist das statische System, die Belastungssituation, sowie der Doppel-T-Querschnitt mit den Variablen dargestellt.

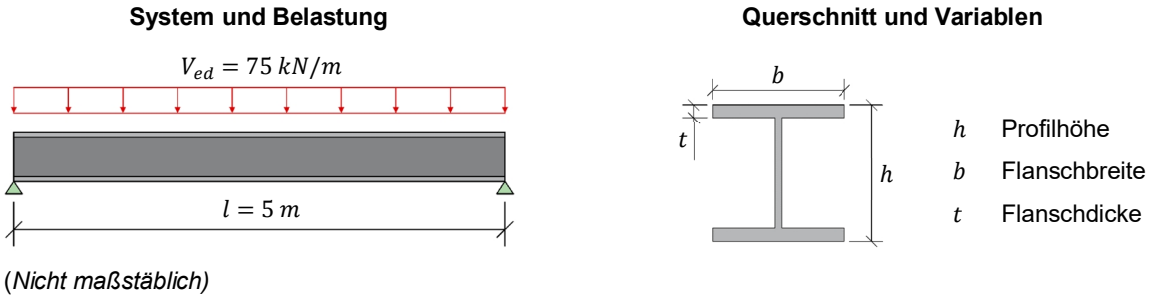


Abbildung 45: Beidseitig gelenkig gelagerter Doppel-T-Profil mit konstanter Belastung

Durch eine lineare Abhängigkeit zwischen Profilhöhe und Stegdicke wird gewährleistet, dass die Stegabmessungen den Grenzwert der Querschnittsklasse 2 nicht überschreiten. Dasselbe gilt für das Verhältnis von b/t, hier werden nur Variablenkombinationen zugelassen, die ebenfalls den Grenzwert der Querschnittsklasse 2 erfüllen.

Eingrenzung des Variablenraums

Um die Wertebereiche der Variablen und dementsprechend den Variablenraum des Problems einzugrenzen, wird zunächst der gesamte Variablenraum nach dem in Kapitel 6 entwickelten Verfahren visualisiert. Die Variablen sind normalisiert auf das Intervall $[0{,}00; 1{,}00]$ mit einer Schrittweite von $0{,}01$. Für die Organisation mit einer SOM werden aus dem gesamten normalisierten Variablenraum $11^3 = 1.331$ Variablensets generiert, was einer Rasterauflösung von $1/10$ entspricht. Die SOM Output-Ebene besitzt mit $80 \cdot 80 = 6.400$ Neuronen ungefähr die fünffache Größe der Input-Ebene. Somit werden zu jedem generierten Variablenset vier weitere Sets durch lineare Interpolation erzeugt. Im Anschluss an die SOM-Organisation erfolgt die Berechnung aller Fitnesswerte, sowohl von den generierten als auch von den interpolierten Variablensets. Zur Darstellung des Variablenraums werden die Variablensets in Form von Quadraten entsprechend ihrem Fitnesswert farblich gekennzeichnet. Dabei werden die besten Fitnesswerte grün und die schlechtesten rot gekennzeichnet.

In Abbildung 46 auf Seite 89 sind alle 6.400 Variablensets entsprechend des Fitnesswerts dargestellt. Die drei besten Regionen wurden gekennzeichnet und werden hinsichtlich der Wertebereiche analysiert.

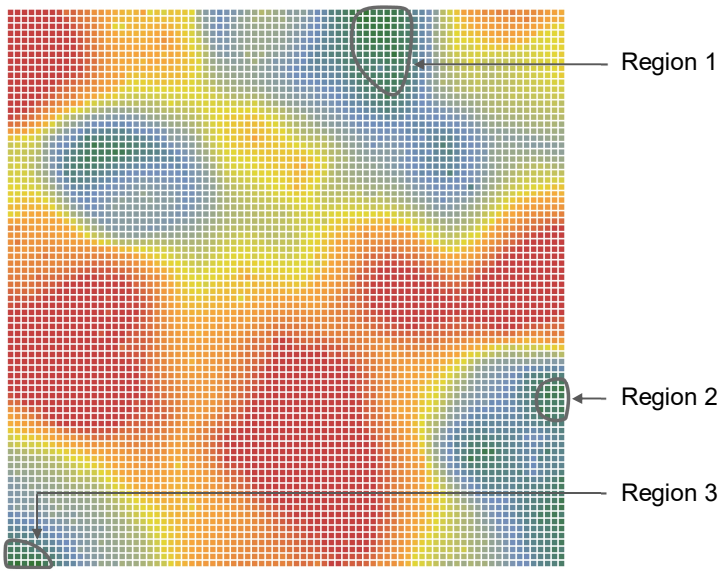


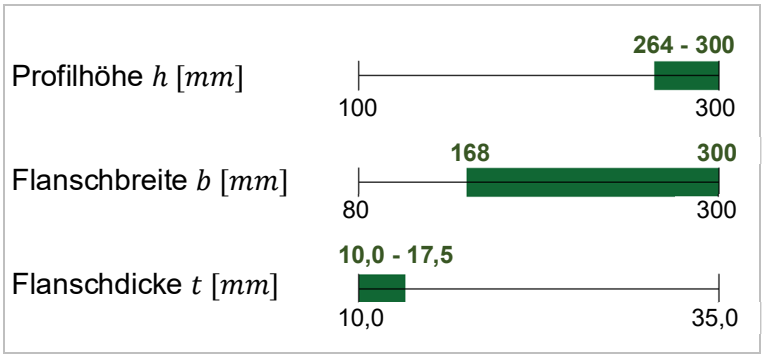
Abbildung 46: Visualisierter Variablenraum: Querschnittsoptimierung – Doppel-T-Profil

In folgender Tabelle sind der gesamte Variablenraum vor der Eingrenzung sowie die Wertebereiche der Variablen in Region 1, 2 und 3 aufgelistet. Zur besseren Übersicht wurden die normalisierten Variablen auf ihre tatsächlichen Werte in Millimeter zurückgerechnet. Bei der Analyse wurden ausschließlich die guten Bereiche des Variablenraums betrachtet, da davon auszugehen ist, dass dieses Optimierungsproblem nur ein globales Optimum besitzt. Der Variablenraum kann auf die guten Bereiche beschränkt werden und eine Betrachtung der mittelguten bis schlechten Bereiche ist nicht notwendig.

Tabelle 3: Analyse des Variablenraums: Querschnittsoptimierung – Doppel-T-Profil

	Profilhöhe h $[mm]$	**Flanschbreite b $[mm]$**	**Flanschdicke t $[mm]$**
Variablenraum	**100 - 300**	**80 - 300**	**10 - 35**
Region 1	276 - 296	190 - 256	12,5 - 17,5
Region 2	282 - 295	168 - 181	10 - 11,5
Region 3	264 - 300	234 - 300	10 - 10,5

Aufgrund der Analyse der Region 1, 2 und 3 wird der Variablenraum wie folgt eingeschränkt:



Die Größe des Optimierungsproblems wurde durch die Einschränkung des Variablenraums von $\mathbf{200 \cdot 220 \cdot 50 = 1.540.000}$ mögliche Lösungen auf $\mathbf{36 \cdot 132 \cdot 15 = 71.280}$ mögliche Lösungen reduziert. Das entspricht einer Reduktion der Problemgröße von über 95 %.

Optimierungsergebnis mit reduziertem Variablenraum

Das Optimierungsergebnis ohne reduzierten Variablenraum ist in Absatz 5.4 zu finden. An dieser Stelle wurde der Einfeldträger mit dem reduzierten Variablenraum ein zweites Mal optimiert. Als Abbruchbedingung für den Optimierungsalgorithmus wurde der beste Fitnesswert aus dem ersten Optimierungsprozess gewählt. So kann festgestellt werden, wie sich der reduzierte Variablenraum auf die Anzahl der Iterationsschritte und die Rechenzeit auswirkt.

Variablenraum	Iterationen	Zeit	Reduziert	Fitnesswert	Verformung
Nicht reduziert	8000	≈ 6‘ 30“	**≈ 55 %**	3,47 kN 0,694 kN/m	20,1 mm
Reduziert	3600	≈ 3‘ 00“			

Bei diesem Optimierungsproblem kann die Rechenzeit um etwas mehr als die Hälfte verkürzt werden, wenn zuvor der Variablenraum verkleinert wird. Bei Rechenzeiten von nur einigen Minuten hat das zwar keine große Relevanz. Bei größeren Problemen mit Rechenzeiten von mehreren Stunden oder Tagen kann dies allerdings ein entscheidender Vorteil sein.

Hier wurde wie bereits beschrieben der Optimierungsprozess abgebrochen, sobald der bekannte Fitnesswert erreicht wurde. Für Optimierungsprobleme, bei denen die Rechenzeit keine Rolle spielt, muss diese Abbruchbedingung nicht implementiert werden. So können mit einem reduzierten Variablenraum bei, im Vergleich zum gesamten Variablenraum, gleicher Anzahl an Iterationsschritten und gleicher Rechenzeit bessere Ergebnisse erzielt werden.

7 Benchmark Tests

7.1 Allgemeines

In folgendem Absatz werden drei Benchmark Tests, bestehend aus typischen Problemstellungen aus dem konstruktiven Bauingenieurwesen, mit jeweils sechs unterschiedlichen Black-Box Algorithmen durchgeführt. Zudem erfolgt für jedes Optimierungsproblem eine Visualisierung und Eingrenzung des Variablenraums auf Grundlage der in dieser Arbeit entwickelten Methode der n-1-dimensionalen Gruppierung.

Sowohl bei der Auswahl der Optimierungsprobleme als auch bei der Auswahl der verwendeten Optimierungsalgorithmen wurde darauf geachtet, dass möglichst alle Kriterien, nach denen unterschieden werden kann, vertreten sind. Zur Differenzierung von Optimierungsproblemen siehe Absatz 2.3. In folgender Tabelle 4 sind die in dieser Arbeit verwendeten Optimierungsalgorithmen gemäß der Optimierungsmethode, dem entsprechenden Plug-In und der Art der Suche zusammengefasst. Eine ausführliche Beschreibung der Algorithmen und Plug-Ins ist in Absatz 2.6 zu finden.

Tabelle 4: Übersicht: Black-Box Optimierungsalgorithmen

<table>
<tr><th>Algorithmus</th><th>Methode</th><th>Plug-In</th><th>Extremwertsuche</th></tr>
<tr><td>GA: Genetischer Algorithmus</td><td rowspan="3">Metaheuristik</td><td rowspan="2">Galapagos</td><td rowspan="3">global und lokal</td></tr>
<tr><td>SA: Simulated Annealing</td></tr>
<tr><td>PSO: Partikelschwarmoptimierung</td><td>Silvereye</td></tr>
<tr><td>RBFOpt: MSRSM Algorithmus</td><td>Ersatzmodell-Methode</td><td>Opossum</td><td rowspan="2">global</td></tr>
<tr><td>DIRECT</td><td rowspan="2">Downhill-Simplex-Methode</td><td rowspan="2">Goat</td></tr>
<tr><td>Sbplx Algorithmus</td><td>lokal</td></tr>
</table>

7.1.1 Ablauf der Benchmark Tests

Jeder Benchmark Test erfolgt nach dem gleichen Schema und kann in die folgenden fünf Punkte eingeteilt werden.

1. **Vorstellung und Kategorisierung des Optimierungsproblems**

   Zunächst wird das Optimierungsproblem anhand einer System- oder Prinzipskizze kurz dargestellt. Anschließend erfolgt die Kategorisierung des Problems in beispielsweise stetig, diskret, global oder lokal. Zur Kategorisierung gehören auch die Angaben zur Dimension, Begrenzung, Penalty-Funktion und Problemgröße sowie Angaben zu den Wertebereichen der einzelnen Variablen.

2. **Zusammenfassung und Bewertung der Optimierungsergebnisse**

   In diesem Schritt werden jeweils die besten Fitnesswerte, die benötigten Iterationsschritte sowie die benötigte Rechenzeit in Abhängigkeit der Optimierungsalgorithmen zusammengefasst und in zwei entsprechenden Grafen dargestellt. Auf Grundlage der Optimierungsergebnisse erfolgt anschließend eine Bewertung der Optimierungsalgorithmen. Die Bewertungskriterien sind in folgendem Absatz zu finden.

3. **Eingrenzung des Variablenraums**

   Im Anschluss an die „herkömmliche“ Optimierung erfolgt die Eingrenzung des Variablenraums mit Hilfe der Methode der n-1-dimnesionalen Gruppierung. Dazu wird jeweils das in Absatz 6.2 vorgestellte Verfahren angewendet.

4. **Analyse des Optimierungsergebnisses aus dem zweiten Optimierungsprozess mit reduzierter Problemgröße**

   Mit dem Ergebnis des zweiten Optimierungsprozesses wird analysiert, welchen Einfluss die Eingrenzung des Variablenraums auf die Rechenzeit und die Qualität des Ergebnisses hat. Der zweite Optimierungsprozesses wird nicht mehr mit allen sechs Optimierungsalgorithmen durchgeführt, sondern nur noch mit dem Algorithmus, der im ersten Optimierungsprozess das beste Ergebnis geliefert hat.

5. **Zusammenfassung**

   Im Anschluss an jeden Benchmark Test werden die Ergebnisse kurz zusammengefasst und ein Fazit gezogen. Es wird dargestellt, welcher Algorithmus bei dem jeweiligen Problem am besten performt hat.

7.1.2 Bewertungskriterien

Alle Optimierungsergebnisse der einzelnen Benchmark Tests werden auf Grundlage der gleichen Bewertungskriterien ausgewertet. Dabei ist der Fitnesswert das wichtigste Kriterium für die Bewertung eines Optimierungsalgorithmus. Wird im Vergleich zu anderen Algorithmen ein deutlich schlechterer Fitnesswert erreicht, ist die Lösung nicht akzeptabel und der entsprechende Algorithmus ist für diese Art des Problems nicht geeignet.

Neben dem Fitnesswert ist die benötigte Rechenzeit ein weiterer wichtiger Faktor. Dabei können hinsichtlich der Rechenzeit zwei Strategien verfolgt werden. Entweder es wird eine maximale Rechenzeit vorgegeben und die Fitnesswerte werden nach dem Erreichen dieser Zeit miteinander verglichen, oder umgekehrt, die benötigten Rechenzeiten werden nach dem Erreichen eines definierten Fitnesswertes miteinander verglichen.

Ein Abbruch des Optimierungsprozesses nach einer zu kurzen Rechenzeit ist allerdings nicht zu empfehlen, da die Ergebnisse falsche Schlussfolgerungen zulassen. Eine Erklärung dafür liefert folgende Grafik.

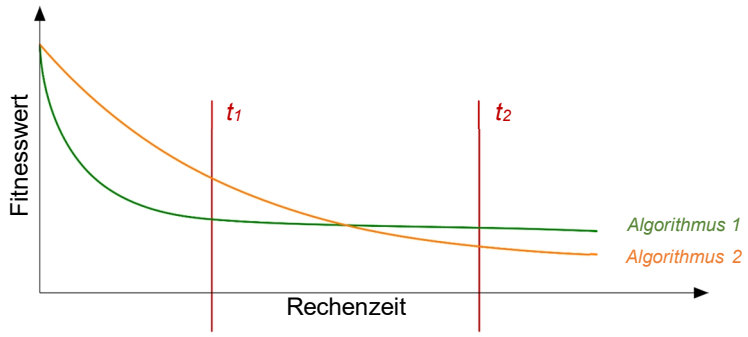


Abbildung 47: Exemplarischer Verlauf zweier Optimierungsprozesse über die Rechenzeit

Gemäß der obenstehenden Grafik nähert sich *Algorithmus 1* zu Beginn des Optimierungsprozesses relativ schnell dem Ziel, stagniert aber mit zunehmender Rechenzeit und verbessert den Fitnesswert nur gering. Im Gegensatz dazu nähert sich *Algorithmus 2* dem Ziel zwar langsamer, dafür aber stetig. Wird der Optimierungsprozess bereits zum Zeitpunkt t_1 abgebrochen, kommt man zu dem Schluss, dass *Algorithmus 1* für das Optimierungsproblem besser geeignet ist, da *Algorithmus 2* ein deutlich schlechteres Ergebnis liefert. Dabei ist zum Zeitpunkt t_1 nicht bekannt, ob *Algorithmus 2* bei einer längeren Rechenzeit das Ergebnis von *Algorithmus 1* einholt oder sogar noch übertrifft, wie es in diesem Beispiel der Fall wäre.

Aus diesem Grund ist es zu empfehlen, falls eine zeitliche Abbruchbedingung implementiert wird, diese so hoch zu setzten, dass der eben beschriebene Effekt ausgeglichen werden kann. Im obenstehenden Beispiel sollte die Grenze in der Region von t_2 liegen. Wo sich diese Grenze tatsächlich befindet, hängt von der Rechenzeit des speziellen Solvers sowie von der Art des Problems ab und kann nicht pauschal bestimmt werden.

Die Optimierungsergebnisse aller Benchmark Test werden im Rahmen dieser Arbeit mit Hilfe der folgenden zwei Grafiken ausgewertet. Zum einen wird der Optimierungsfortschritt, also die Verbesserung des Fitnesswerts über die logarithmisch skalierte Rechenzeit dargestellt (Abbildung 48, links). In dieser Grafik ist zu erkennen, welcher Algorithmus zu welchem Zeitpunkt im Optimierungsprozess wie performt hat. Zum anderen wird mit der zweiten Grafik (rechts) ein Zusammenhang zwischen dem besten Fitnesswert und der gesamten Rechenzeit hergestellt. Somit ist schnell zu erkennen, welcher Algorithmus in der kürzesten Rechenzeit das beste Ergebnis geliefert hat. Die Grafik kann dafür in die Bereiche gut, akzeptabel und schlecht eingeteilt werden. Die Informationen über Fitnesswert und gesamte Rechenzeit sind zwar auch in der linken Grafik des Optimierungsfortschritts enthalten, allerdings ist das Verhältnis zwischen Ergebnis und Zeit aufgrund der logarithmischen Skalierung der Zeit verzerrt und somit schwer abzulesen.

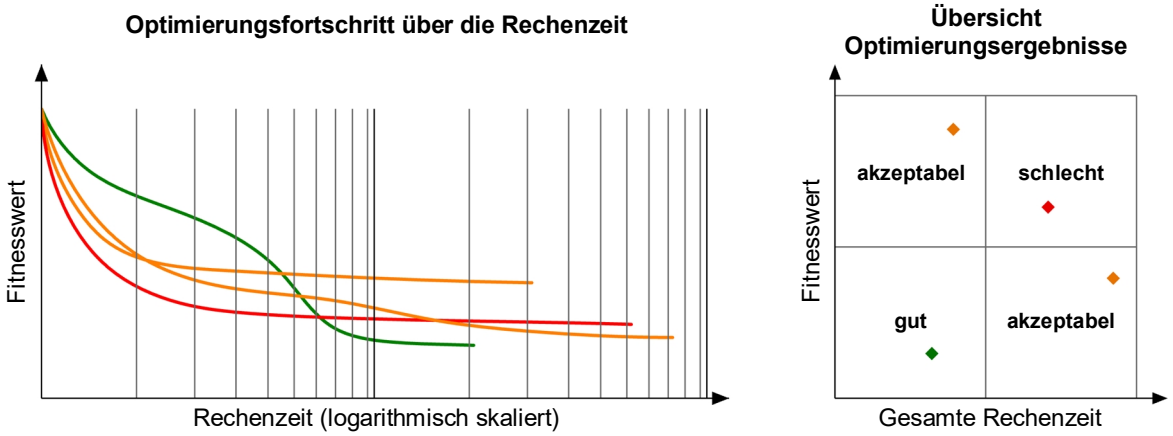


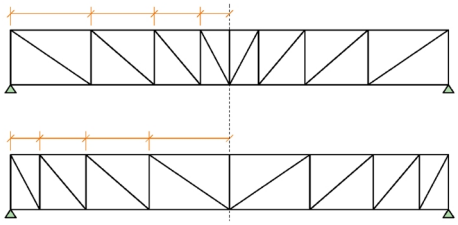
Abbildung 48: Prinzipdarstellung der Grafiken zur Ergebnisauswertung

7.2 Übersicht: Benchmark Testprobleme

Im Rahmen dieser Arbeit wurden für die Durchführung der Benchmark Tests insgesamt drei Test-Optimierungsprobleme entwickelt. Bei der Entwicklung der Probleme und dem jeweiligen Optimierungsziel spielten vor allem zwei Kriterien eine Rolle. Zum einen sollen die Probleme realen konstruktiven Ingenieursaufgaben entsprechen, die so auch in der Praxis Anwendung

finden, zum anderen sollen mit den drei Problemen möglichst alle Kriterien, zwischen denen unterschieden werden kann, abgedeckt werden. Im Folgenden werden die drei Test-Optimierungsprobleme kurz vorgestellt, eine detaillierte Beschreibung und Kategorisierung ist jeweils zu Beginn des entsprechenden Absatzes zu finden.

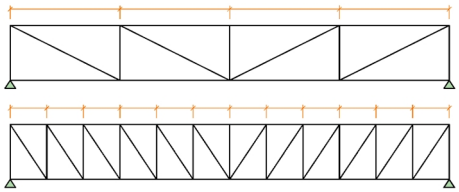
1. Fachwerkträger mit variablen Querschnittsabmessungen und variabler Zugstrebenneigung



Der Fachwerkträger soll durch die Variation der Querschnitte und der Zugstrebenneigung hinsichtlich Materialbedarf und maximaler Verformung optimiert werden.

Eigenschaften:
- **5-dimensional**
- **groß**
- **stetig**
- **global**

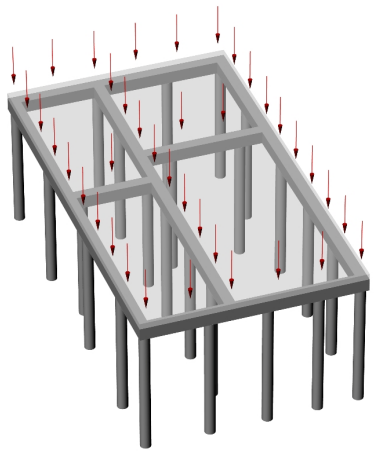
2. Fachwerkträger mit variablen Querschnittsabmessungen und variabler Zugstreben-Anzahl



Im Gegensatz zu Problem 1 wird hier zusammen mit den Querschnitten nicht die Zugstrebenneigung, sondern die Anzahl der Zugstreben variiert. Das Ziel ist weiterhin die Einhaltung der Verformungsgrenze bei gleichzeitiger Minimierung des Materialbedarfs.

Eigenschaften:
- **5-dimensional**
- **mittelgroß**
- **diskret**
- **global**

3. Pfahlgründung: Anordnung von Pfählen unter Pfahlkopfbalken



Die Ausgangssituation dieses Problems ist die Pfahlgründung eines typischen Mehrfamilienhauses. Das Ziel ist, die Pfähle unter den Pfahlkopfbalken so anzuordnen, dass sich alle Pfähle auf dem gleichen Lastniveau befinden, während die Pfahlkopfbalken eine minimale Verformung erfahren.

Eigenschaften:
- **20-dimensional**
- **sehr groß**
- **stetig**
- **lokal**

7.3 Fachwerkträger: Variable Querschnittsabmessungen und variable Zugstrebenneigung

Das folgende Optimierungsproblem ist ein typisches und häufig vorkommendes Bemessungsproblem im Bauingenieurwesen. Fachwerkträger, oder im Allgemeinen Fachwerkkonstruktionen, kommen häufig zur Überbrückung von großen Spannweiten oder zur Aufnahme von hohen Lasten zum Einsatz. Dazu zählen unter anderem Brückenbauwerke, Hallendachkonstruktionen oder Aussteifungssysteme von Hochhäusern.

Die Randbedingungen dieses Optimierungsproblems könnten beispielsweise aus dem Entwurf einer Fußgängerbrücke oder eines Hallendachs stammen. Der Fachwerkträger soll hinsichtlich des Materialverbrauchs optimiert werden, wobei die maximale Verformung in Feldmitte als Grenzwert definiert wird. Es können sowohl die Querschnittsprofile als auch der Winkel der Zugstreben variiert werden. In Abbildung 49 ist das statische System mit den Trägerabmessungen sowie die Belastungssituation dargestellt.

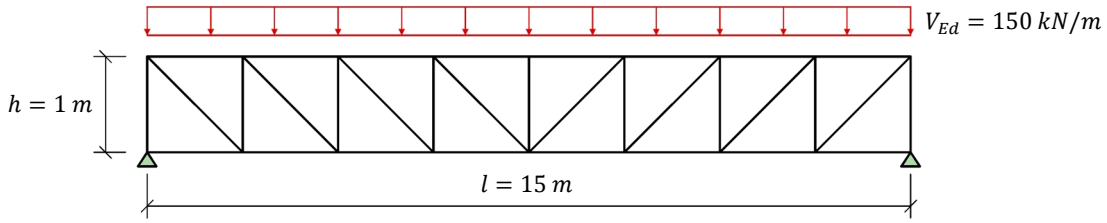


Abbildung 49: Statisches System und Belastungssituation

Variablen

Die Stäbe des Fachwerkträgers werden in Zug- und Druckstreben sowie in Ober- und Untergurt eingeteilt, siehe Abbildung 50. Für jede Stabgruppe kann unabhängig voneinander ein Profilquerschnitt gewählt werden. Zur Auswahl stehen Doppel-T-Profile aus HE-A Normfamilie des Eurocodes [15]. Variiert wird zwischen den Profilen HE-A 100 und HE-A 500.

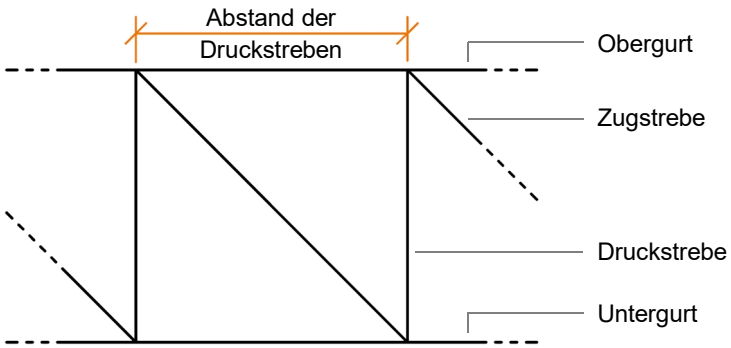


Abbildung 50: Übersicht der Querschnittsvariablen

Obwohl die Zugstrebenneigung variabel ist, kann sie nicht komplett frei gewählt werden. Der Winkel zwischen der Horizontalen und der Zugstrebe ist definiert durch die Höhe des Trägers und der Anzahl der Druckstreben, beziehungsweise durch den Abstand zwischen den Druckstreben. Da die Höhe des Fachwerkträgers und die Anzahl der Druckstreben in diesem Entwurf konstante Größen sind, wird zur Variation der Zugstrebenneigung der Abstand zwischen den Druckstreben verändert. Da eine willkürliche Veränderung der Druckstrebenabstände nicht zielführend wäre, werden die Abstände über einen Faktor v, ausgehend von der Symmetrieachse des Trägers, linear nach außen oder innen verschoben. Der Grad der Verschiebung einer Druckstrebe ist durch $v(x) = v \cdot x$ definiert, wobei die Variable v der Faktor der Verschiebung ist und x der Abstand zur Symmetrieachse. In folgender Abbildung sind zwei mögliche Lösungen des Optimierungsproblems dargestellt.

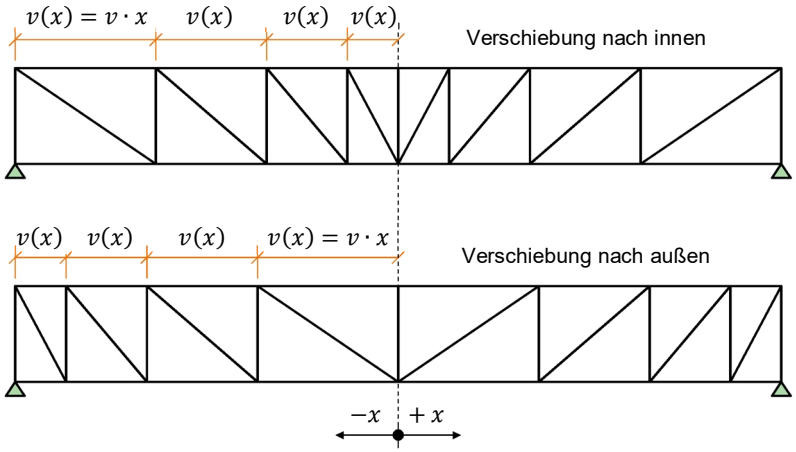


Abbildung 51: Variationsprinzip der Zugstrebenneigung

Variablenraum

In Tabelle 5 sind alle fünf Variablen mit den entsprechenden Wertebereichen und der Menge an möglichen Werten angegeben. Die Summe aller Wertebereiche bildet den Variablenraum.

Tabelle 5: Variablenraum: Fachwerkträger mit variabler Zugstrebenneigung

Variable		Intervall / Wertebereich	Auspräung
Profil Obergurt	p_O:	*HE-A 100 - 500*	17
Profil Untergurt	p_U:	*HE-A 100 - 500*	17
Profil Druckstrebe	p_D:	*HE-A 100 - 500*	17
Profil Zugstrebe	p_Z:	*HE-A 100 - 500*	*17*
Faktor der Verschiebung (Zugstrebenneigung)	v:	*-1,00 – 1,00*	*200*

Per Definition sind die hier verwendeten Variablen diskret, da sie jeweils eine abzählbare Menge an Werten besitzen, siehe Absatz 2.1 *Stetige und diskrete Variablen*. Allerdings ist die

Menge der möglichen Ausprägungen der Variable der Zugstrebenneigung so groß, dass sie als quasi-stetig definiert werden kann. Ähnlich verhält es sich mit den Querschnittsvariablen. Diese haben zwar keine große Ausprägung, jedoch ist der Sprung zwischen den Querschnitten so gering, dass die Berechnung des Fitnesswertes auch als quasi-stetig angesehen werden kann.

Unter den Querschnittsvariablen gibt es keine dominierenden oder untergeordneten Variablen, da alle das Ergebnis der Optimierung in gleichem Maße beeinflussen. Ob die Variable der Zugstrebenneigung dominierend, untergeordnet oder ebenbürtig mit den Querschnittsvariablen ist, könnte erst mit der Auswertung der Fitnesslandschaft festgestellt werden. Obwohl kleinere lokale Extremwerte zu vermuten sind, kann die Extremwertsuche global durchgeführt werden, da zu erwarten ist, dass der Fitnesswert des optimalen Fachwerkträgers deutlich besser ist, als der aller anderen möglichen Entwürfe.

Nebenbedingung

Die Minimierung des Materialbedarfs durch die Variation der Querschnitte und der Zugstrebenneigung wird durch die Verformungsgrenze $\boldsymbol{d_{max}}$ begrenzt. Die maximale Verformung in Feldmitte darf höchstens $\boldsymbol{l/250}$ betragen, also $\boldsymbol{15.000\ mm/250 = 60\ mm}$. Ist eine Verformung von $\boldsymbol{60\ mm}$ erreicht, wird der Fitnesswert beziehungsweise das Eigengewicht der Konstruktion erhöht, in welchem Maße bestimmt die Penalty-Funktion.

Kategorisierung des Optimierungsproblems

Das Optimierungsproblem kann aufgrund der oben beschriebenen Eigenschaften wie folgt kategorisiert werden:

Tabelle 6: Kategorisierung: Fachwerkträger mit variabler Zugstrebenneigung

Dimensionalität	5
Größe des Problems	$17 \cdot 17 \cdot 17 \cdot 17 \cdot 200 = \mathbf{16.704.200}$ mögliche Lösungen
Beschaffenheit	Quasi-stetig
Extremwertsuche	global
Fitnesswert	Konstruktionseigengewicht / Materialbedarf
Nebenbedingung	Verformung $d_{max} = 60\ mm$
Penalty-Funktion	wenn $d > d_{max}$; dann $Fitesswert \cdot (d/d_{max})^2$; sonst $Fitesswert$
Spezieller Solver	FE-M Berechnung des Fachwerkträgers

Die Grasshopper Definition des Benchmark Tests ist als Schemadarstellung in Anhang A, oder in digitaler Form in Anlage C zu finden.

7.3.1 Optimierungsergebnisse

In folgender Tabelle sind die Ergebnisse der einzelnen Optimierungsprozesse zusammengefasst dargestellt.

Tabelle 7: Übersicht Optimierungsergebnisse

	EA	SA	PSO	RBFOpt	DIRECT	Sbplx
Rechenzeit	06:08	05:24	04:05	27:36	23:44	22:14
Fitnesswert (Eigengewicht)	28,43 kN	28,99 kN	27,72 kN	27,78 kN	27,95 kN	28,59 kN
Nebenbedingung (Verformung)	60 mm	59 mm	59 mm	59 mm	59 mm	60 mm

Der Partikelschwarmalgorithmus (PSO) liefert in kürzester Zeit den besten Fitnesswert und unterbietet die Nebenbedingung um 1 mm. Folgende Grafik stellt den Verlauf der Optimierungsprozesse dar. Es ist festzuhalten, dass die drei metaheuristischen Algorithmen, EA, SA und PSO, im Vergleich zu den drei anderen Algorithmen, sehr schnell konvergieren. Dabei sind PSO und EA die einzigen Algorithmen, die sich über den Optimierungsprozess hinweg nahezu konstant verbessern.

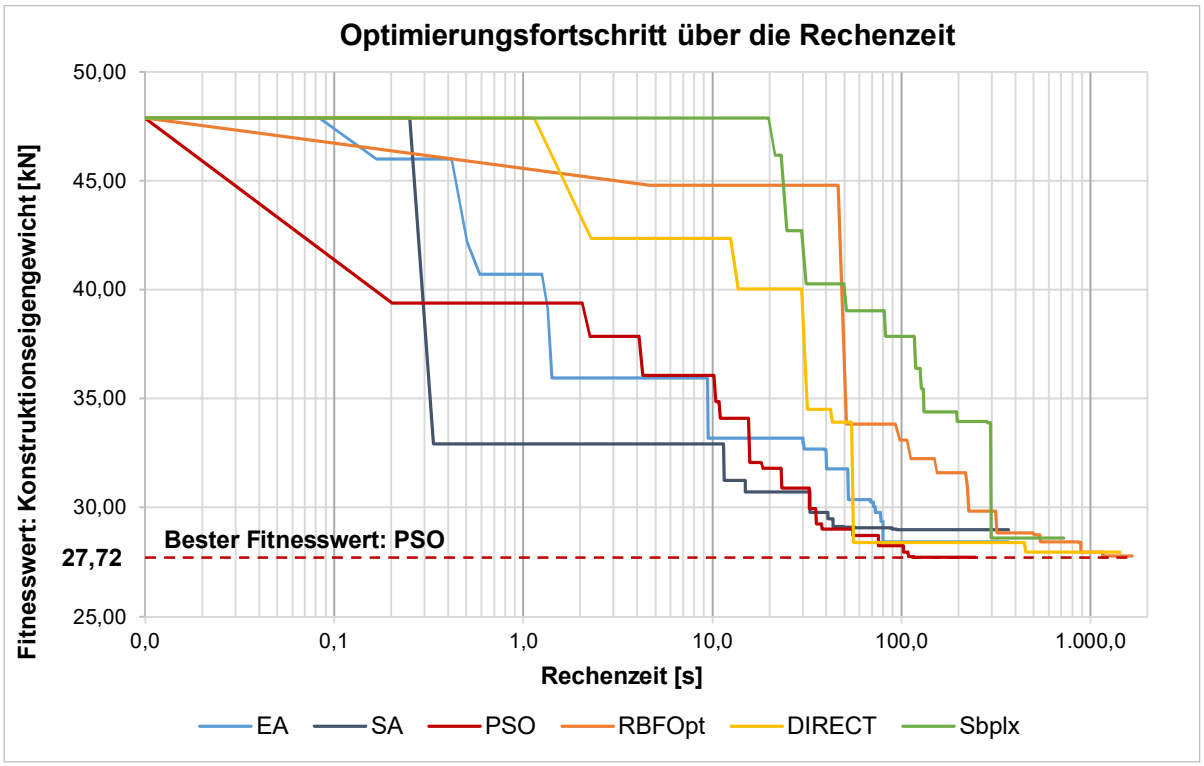


Abbildung 52: Optimierungsfortschritt: Fachwerkträger, variable Zugstrebenneigung

Die metaheuristischen Algorithmen, im speziellen aber der PSO Algorithmus ist dem RBFOpt, dem DIRECT und dem Sbplx Algorithmus in puncto Rechenzeit überlegen. SA liefert

zwar den schlechtesten Fitnesswert unter den sechs getesteten Algorithmen, allerdings ist die Spanne der Fitnesswerte nicht besonders groß. Zwischen dem besten und dem schlechtesten Wert liegen nur 0,71 kN, was zu vernachlässigen ist. Anders verhält es sich bei der Rechenzeit. Während die Metaheuristiken um die fünf Minuten für die Lösung des Optimierungsproblems benötigen, brauchen die anderen drei Algorithmen etwa fünfmal so lang.

Der Optimierungsprozess mit dem besten Fitnesswert ergab die folgenden Querschnittsabmessungen und den folgenden Faktor der Verschiebung:

Tabelle 8: Bestes Variablenset

Variable		Wert	Fitnesswert
Profil Obergurt	p_O:	*HE-A 280*	27,72 kN (PSO)
Profil Untergurt	p_U:	*HE-A 180*	
Profil Druckstrebe	p_D:	*HE-A 160*	
Profil Zugstrebe	p_Z:	*HE-A 180*	
Faktor der Verschiebung	v:	*-0,22*	

7.3.2 Eingrenzung des Variablenraums

Die Visualisierung des Variablenraums und die Analyse der Regionen erfolgt nach dem in Kapitel 6 beschriebenen Verfahren. In folgender Abbildung ist der visualisierte Variablenraum mit den zur Analyse ausgewählten Regionen dargestellt. Anschließend sind in Tabelle 9 die Ergebnisse der Analyse zu finden.

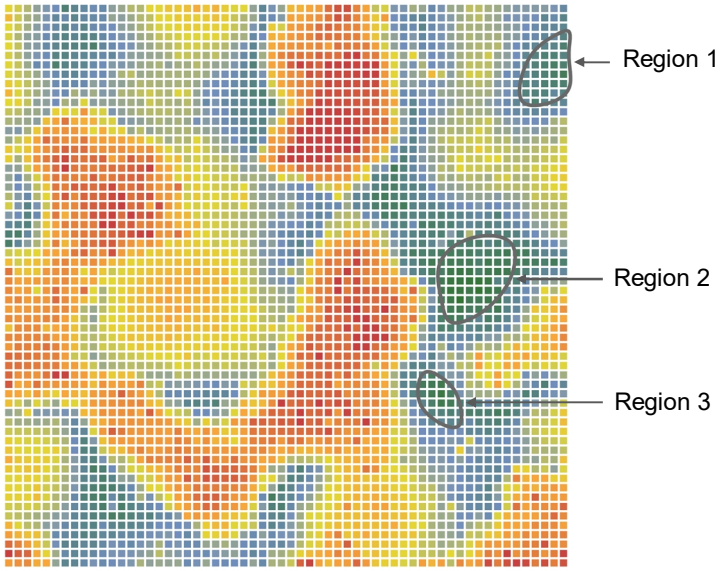


Abbildung 53: Visualisierter Variablenraum: Fachwerkträger, var. Zugstrebenneigung

Die Regionen 1, 2 und 3 markieren die Bereiche des Variablenraums mit den besten Fitnesswerten. Dabei repräsentiert jedes Rechteck in dem Variablenraum ein Variablenset. Die Wertebereiche, in denen sich die Variablen in den entsprechenden Regionen bewegen, sind in folgender Tabelle zu finden.

Tabelle 9: Analyse des Variablenraums

	Obergurt [HE-A]	**Untergurt [HE-A]**	**Druckstrebe [HE-A]**	**Zugstrebe [HE-A]**	**Faktor der Verschiebung**
Variablenraum	**100 – 500**	**100 – 500**	**100 – 500**	**100 – 500**	**100 – 500**
Region 1	*260 – 280*	*120 – 220*	*160 – 240*	*100 – 180*	*-0,25 – -0,01*
Region 2	*260 – 450*	*120 – 200*	*140 – 240*	*120 – 200*	*-0,86 – 0,50*
Region 3	*300 – 450*	*100 – 120*	*200 – 260*	*140 – 180*	*-0,30 – -0,89*

Aufgrund der Analyse der Region 1, 2 und 3, wird der Variablenraum wie folgt eingeschränkt:

Tabelle 10: Eingegrenzter Variablenraum

Variable		**Reduzierter Wertebereich**
Profil Obergurt	p_O:	*HE-A 260 – 450*
Profil Untergurt	p_U:	*HE-A 100 – 220*
Profil Druckstrebe	p_D:	*HE-A 140 – 260*
Profil Zugstrebe	p_Z:	*HE-A 100 – 200*
Faktor der Verschiebung	v:	*- 0,86 – 0,50*

Die Größe des Optimierungsproblems wurde durch die Einschränkung des Variablenraums von $17 \cdot 17 \cdot 17 \cdot 17 \cdot 200 = \mathbf{16.704.200}$ mögliche Lösungen auf $6 \cdot 8 \cdot 6 \cdot 7 \cdot 136 = \mathbf{274.176}$ mögliche Lösungen reduziert. Das entspricht einer Reduktion der Problemgröße von rund 98 %.

7.3.3 Optimierungsergebnis mit eingegrenztem Variablenraum

Der Optimierungsprozess mit eingegrenztem Variablenraum beziehungsweise reduzierter Problemgröße wird in diesem Fall nicht wie in Absatz 7.1.1 beschrieben mit nur einem Algorithmus durchgeführt, sondern mit zwei unterschiedlichen Algorithmen. Zunächst wird das reduzierte Problem mit dem PSO Algorithmus optimiert, da dieser das beste Optimierungsergebnis geliefert hat. Da der PSO Algorithmus bereits nach 4:05 Minuten konvergiert hat, ist jedoch keine große absolute Verbesserung der Rechenzeit zu erwarten. Aus diesem Grund wird das reduzierte Problem zusätzlich mit dem RBFOpt Algorithmus optimiert. Das Ergebnis des RBFOpt Algorithmus ist vergleichbar mit dem des PSO, jedoch

ist die ursprüngliche Rechenzeit mit 27:36 Minuten fasst siebenmal so lang, siehe Tabelle 7 auf Seite 99. Durch das Testen des RBFOpt Algorithmus wird sich erhofft, dass ein weiterer Algorithmus für die Optimierung von stetigen Problemen in Frage kommt, der nicht der Gruppe der Metaheuristiken angehört.

In Tabelle 11 sind die Ergebnisse des PSO und des RBFOpt Algorithmus aus beiden Optimierungsprozessen gegenübergestellt.

Tabelle 11: Vergleich: Ergebnisse mit vollständigem und reduziertem Variablenraum

	Vollständiger Variablenraum	Reduzierter Variablenraum	Verbesserung
Algorithmus	**PSO**		
Iterationen	1202	842	
Rechenzeit	4:05	2:29	**39,2 %**
Fitnesswert (Eigengewicht)	27,72 kN	27,62 kN	**≈ 0 %**
Nebenbedingung (Verformung)	59 mm	59 mm	
Algorithmus	**RBFOpt**		
Iterationen	358	64	
Rechenzeit	27:36	8:25	**69,5 %**
Fitnesswert (Eigengewicht)	27,78 kN	27,78 kN	**≈ 0 %**
Nebenbedingung (Verformung)	59 mm	59 mm	

Wie zu erwarten war, ist die Verbesserung der Rechenzeit des RBFOpt Algorithmus sowohl absolut als auch relativ höher als die des PSO Algorithmus.

7.3.4 Schlussfolgerung

Mit der Dimension fünf und einer Variablenraumgröße im zweistelligen Millionenbereich kann das Optimierungsproblem des Fachwerkträgers, im Hinblick auf andere typische Ingenieursprobleme, als groß bezeichnet werden. Aufgrund der Ergebnisse aus den ersten Optimierungsprozessen mit vollständigem Variablenraum, können für stetige Optimierungsprobleme die metaheuristischen Algorithmen empfohlen werden. Der PSO Algorithmus liefert im Benchmark Test mit vollständigem Variablenraum das beste Ergebnis, wobei auch der EA und SA Algorithmus im Hinblick auf die Rechenzeit gut performen. Algorithmen der Ersatzmodell-Methode und der Downhill-Simplex-Methode sind aufgrund der hohen Rechenzeiten nicht geeignet. Dies ändert sich allerdings, wenn vor dem Optimierungsprozess eine Eingrenzung des Variablenraums vorgenommen wird. Die Rechenzeit des RBFOpt Algorithmus verkürzt sich von 27:36 Minuten um 69,5 % auf 8:25

Minuten, siehe Tabelle 11 auf Seite 102 und Abbildung 54. Damit kann der Algorithmus der Ersatzmodel-Methode in puncto Rechenzeit und Fitnesswert durchaus mit den Metaheuristiken mithalten.

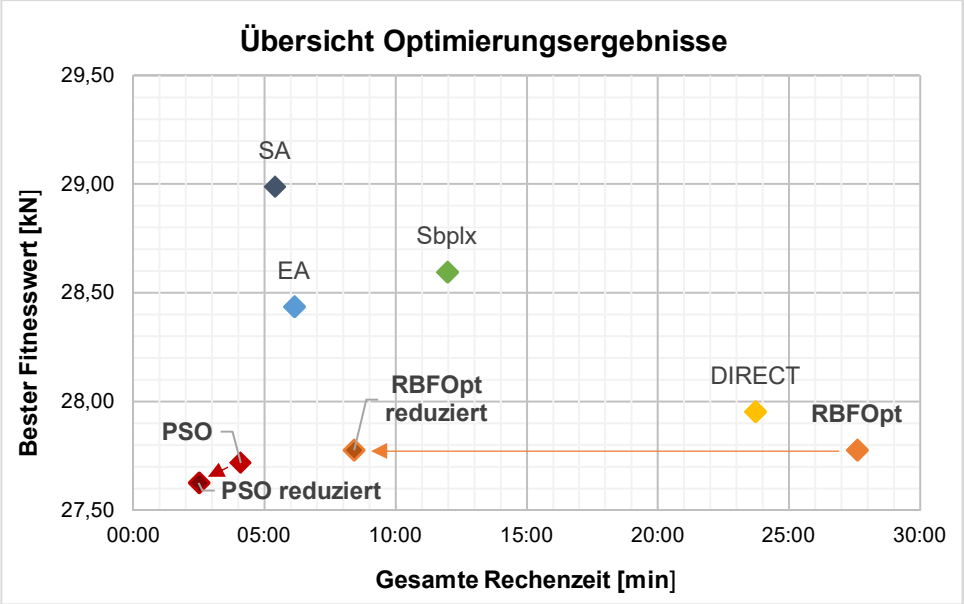


Abbildung 54: Übersicht Optimierungsergebnisse: Fachwerkträger, variable Zugstrebenneigung

Abschließend kann für ein stetiges Optimierungsproblem dieser Größe die Anwendung von metaheuristischen Algorithmen nur empfohlen werden, wenn der Variablenraum nicht eingegrenzt werden kann. Vor allem durch die kurze Rechenzeit sind sie den Algorithmen der Ersatzmodell-Methode und der Downhill-Simplex-Methode vorzuziehen.

Da die Metaheuristiken unter den Black-Box Algorithmen als die ungenausten bezeichnet werden [3], bietet es sich an, vor dem Optimierungsprozess den Variablenraum einzugrenzen und anschließend den RBFOpt Algorithmus der Ersatzmodell-Methode zu verwenden.

7.4 Fachwerkträger: Variable Querschnittsabmessungen und variable Zugstreben-Anzahl

Die Randbedingungen des folgenden Benchmark Tests sowie das statische System und die Belastungssituation unterscheiden sich nicht von dem Benchmark Test in Absatz 7.2. Anstatt der Zugstrebenneigung ist an diesem Fachwerkträger die Anzahl der Zugstreben variabel. So wird aus dem stetigen Optimierungsproblem aus Absatz 7.2 ein diskretes. Abbildung 55 zeigt einen Entwurf des Fachwerkträgers mit vier und einen mit zwölf Zugstreben.

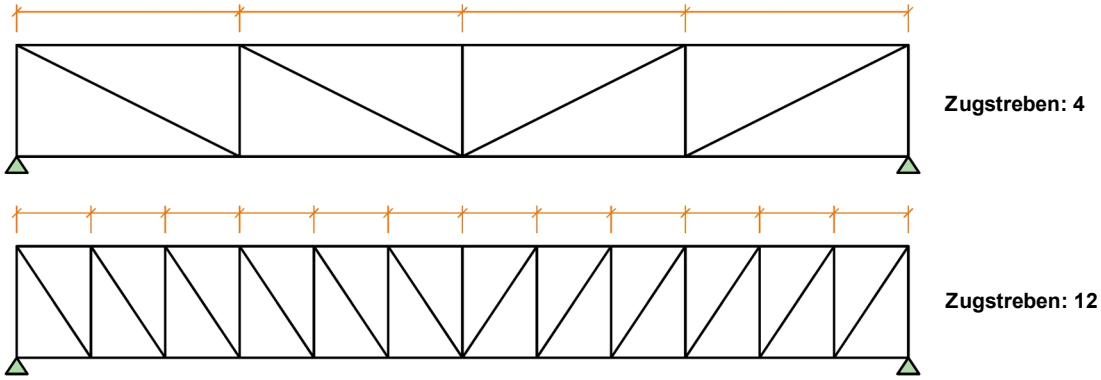


Abbildung 55: Mögliche Entwürfe des Fachwerkträgers

Variablen

Die Querschnitte des Ober- und Untergurts, sowie der Zug- und Druckstreben können jeweils unabhängig voneinander variiert werden. Hier stehen die Doppel-T-Profile der Eurocode Normfamilie HE-A 100 bis HE-A 500 zur Verfügung [15]. Die Anzahl der Zugstreben kann zwischen vier und zwölf variiert werden, wobei nur gerade Anzahlen zugelassen werden. So wird garantiert, dass sich eine Druckstrebe immer in Feldmitte befindet.

Variablenraum

Tabelle 12: Variablenraum: Fachwerkträger, variable Zugstreben-Anzahl

Variable		Intervall / Wertebereich	Auspräung
Profil Obergurt	p_O:	*HE-A 100 - 500*	17
Profil Untergurt	p_U:	*HE-A 100 - 500*	17
Profil Druckstrebe	p_D:	*HE-A 100 - 500*	17
Profil Zugstrebe	p_Z:	*HE-A 100 - 500*	*17*
Zugstreben-Anzahl	z:	*4 – 12 (nur gerade Anzahl)*	*5*

Während die Querschnittsvariablen p_O, p_U, p_D und p_z aufgrund ihrer Ausprägung und dem geringen Unterschied zwischen den Querschnitten als quasi-stetig zu definieren sind, ist die

Zugstreben-Anzahl z eine diskrete Variable. Durch die Variation der Zugstreben-Anzahl ist ein Sprung im Fitnesswert zu erwarten.

Nebenbedingung

Die Nebenbedingung wird analog zu dem Optimierungsproblem in Absatz 7.2 gewählt. Die maximale Verformung in Feldmitte wird auf $\boldsymbol{d_{max} = l/250}$ begrenzt. Das entspricht einer maximalen Verformung von $\boldsymbol{15.000\,mm/250 = 60\,mm}$. Eine Verletzung des Grenzwertes $\boldsymbol{d_{max}}$ wird durch die Penalty-Funktion bestraft.

Kategorisierung des Optimierungsproblems

Das Optimierungsproblem kann aufgrund der oben beschriebenen Eigenschaften wie folgt kategorisiert werden.

Tabelle 13: Kategorisierung: Fachwerkträger, variable Zugstreben-Anzahl

Dimensionalität	5
Größe des Problems	$17 \cdot 17 \cdot 17 \cdot 17 \cdot 5 = \mathbf{417.605}$ mögliche Lösungen
Beschaffenheit	diskret
Extremwertsuche	global
Fitnesswert	Konstruktionseigengewicht / Materialbedarf
Nebenbedingung	Verformung $d_{max} = 60\ mm$
Penalty-Funktion	wenn $d > d_{max}$; dann $Fitesswert \cdot (d/d_{max})^2$; sonst $Fitesswert$
Spezieller Solver	FE-M Berechnung des Fachwerkträgers

Die Grasshopper Definition des Benchmark Tests ist als Schemadarstellung in Anhang A, oder in digitaler Form in Anlage C zu finden.

7.4.1 Optimierungsergebnisse

In folgender Tabelle sind die Ergebnisse der einzelnen Optimierungsprozesse zusammengefasst dargestellt.

Tabelle 14: Übersicht Optimierungsergebnisse: Fachwerkträger, var. Zugstreben-Anzahl

	EA	SA	PSO	RBFOpt	DIRECT	Sbplx
Rechenzeit	09:04	14:39	05:42	1:03:43	36:05	09:41
Fitnesswert (Eigengewicht)	28,39 kN	28,61 kN	29,57 kN	28,76 kN	27,72 kN	28,60 kN
Nebenbedingung (Verformung)	59 mm	50 mm	60 mm	59 mm	59 mm	60 mm

Der DIRECT Algorithmus der Downhill-Simplex-Methode liefert mit einem Konstruktionseigengewicht von 27,27 kN oder 2,73 Tonnen das beste Ergebnis unter den getesteten Algorithmen. Alle drei metaheuristischen Algorithmen konvergieren trotz eines schlechteren Endergebnisses am schnellsten. Sie benötigen für die Optimierung des stetigen Problems nur zwischen 5:42 und 14:39 Minuten.

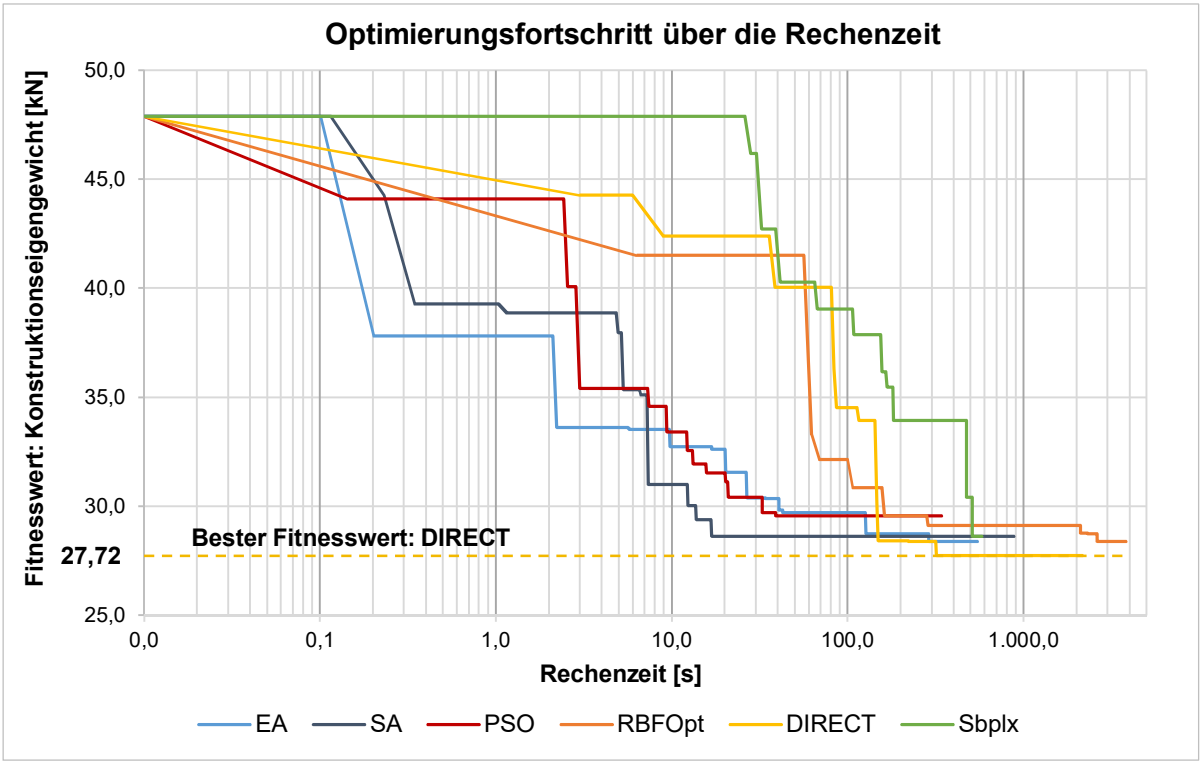


Abbildung 56: Optimierungsfortschritt: Fachwerkträger, variable Zugstreben-Anzahl

Der Optimierungsprozess mit dem DIRECT Algorithmus ergab die folgenden Querschnittsabmessungen und folgende Zugstreben-Anzahl:

Tabelle 15: Bestes Variablenset: Fachwerkträger, variable Zugstreben-Anzahl

Variable		Wert	Fitnesswert
Profil Obergurt	p_O:	*HE-A 280*	27,72 kN (DIRECT)
Profil Untergurt	p_U:	*HE-A 180*	
Profil Druckstrebe	p_D:	*HE-A 160*	
Profil Zugstrebe	p_Z:	*HE-A 180*	
Zugstreben-Anzahl	z:	*8*	

7.4.2 Eingrenzung des Variablenraums

Die Visualisierung des Variablenraums und die Analyse der markierten Regionen erfolgt nach dem in Kapitel 6 beschriebenen Verfahren. Aufgrund der Einfärbung der Variablensets

entsprechend der Fitnesswerte können die Regionen 1, 2 und 3 definiert werden. Sie markieren die Bereiche im Variablenraum, deren Variablensets die besten Ergebnisse liefern.

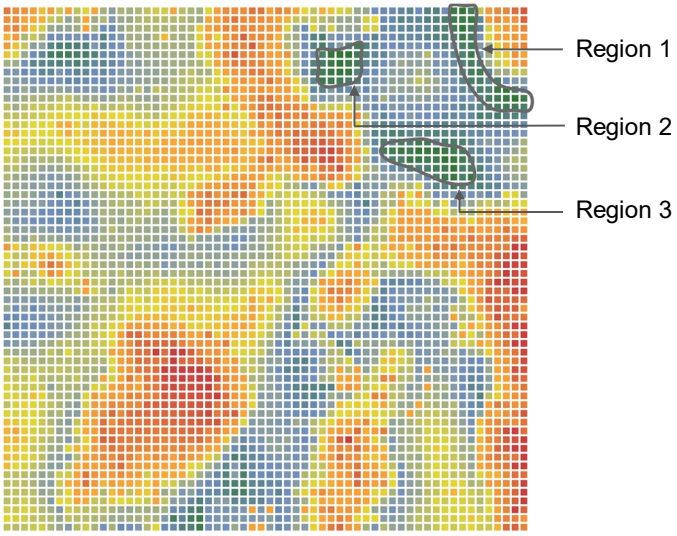


Abbildung 57: Visualisierter Variablenraum: Fachwerkträger, variable Zugstreben-Anzahl

Die Analyse der Regionen 1, 2 und 3 ist in folgender Tabelle zu finden.

Tabelle 16: Analyse des Variablenraums

	Obergurt [HE-A]	**Untergurt [HE-A]**	**Druckstrebe [HE-A]**	**Zugstrebe [HE-A]**	**Zugstreben-Anzahl**
Variablenraum	**100 – 500**	**100 – 500**	**100 – 500**	**100 – 500**	**100 – 500**
Region 1	*240 – 280*	*140– 220*	*160 – 200*	*160 – 200*	*8*
Region 2	*260 – 300*	*180 – 200*	*120 – 220*	*180 – 200*	*10*
Region 3	*280 – 320*	*140 – 200*	*160 – 160*	*160 – 180*	*8*

Der Variablenraum kann aufgrund der Visualisierung wie folgt eingeschränkt werden:

Tabelle 17: Eingegrenzter Variablenraum

Variable		**Reduzierter Wertebereich**
Profil Obergurt	p_O:	*HE-A 240 – 320*
Profil Untergurt	p_U:	*HE-A 140 – 220*
Profil Druckstrebe	p_D:	*HE-A 120 – 220*
Profil Zugstrebe	p_Z:	*HE-A 160 – 200*
Zugstreben-Anzahl	v:	*8 - 10*

Die Größe des Optimierungsproblems wurde durch die Einschränkung des Variablenraums von $17 \cdot 17 \cdot 17 \cdot 17 \cdot 5 = \mathbf{417.605}$ mögliche Lösungen auf $5 \cdot 5 \cdot 6 \cdot 3 \cdot 2 = \mathbf{900}$ mögliche Lösungen reduziert. Das entspricht einer Reduktion der Problemgröße von 99,8 %.

7.4.3 Optimierungsergebnis mit eingegrenztem Variablenraum

Der Optimierungsprozess mit eingegrenztem Variablenraum wird mit dem DIRECT Algorithmus durchgeführt. Durch den Vergleich der Ergebnisse aus dem ersten und zweiten Optimierungsprozess wird gezeigt, inwiefern die Rechenzeit und das Optimierungsergebnis durch eine Reduzierung des Variablenraums beeinflusst werden.

Tabelle 18: Vergleich: Ergebnisse mit vollständigem und reduziertem Variablenraum

	Vollständiger Variablenraum	Reduzierter Variablenraum	Verbesserung
Algorithmus	DIRECT		
Iterationen	721	151	
Rechenzeit	36:05	06:35	**81,8 %**
Fitnesswert (Eigengewicht)	27,72 kN	27,72 kN	**= 0 %**
Nebenbedingung (Verformung)	59 mm	59 mm	

Durch die Reduzierung der Problemgröße infolge eines eingegrenzten Variablenraums kann die Rechenzeit der Optimierung um rund 80 % verkürzt werden.

7.4.4 Schlussfolgerung

Durch die diskrete Variable z (Zugstreben-Anzahl) ist das Optimierungsproblem, im Vergleich zu dem stetigen Problem aus Absatz 7.3, merklich komplexer geworden, da alle Algorithmen im Schnitt mehr Zeit für die Optimierung des Problems benötigen. Die metaheuristischen Algorithmen EA, SA und PSO konvergieren mit Rechenzeiten von 5:42 bis 14:39 Minuten zwar immer noch relativ schnell, das beste Ergebnis liefert allerdings der DIRECT Algorithmus mit einem Fitnesswert von 27,72 kN und einer Rechenzeit von 36:05 Minuten. Wird eine Eingrenzung des Variablenraums vorgenommen, kann bei gleichbleibendem Fitnesswert die Rechenzeit des DIRECT Algorithmus um rund 82 % auf 6:35 Minuten verkürzt werden (siehe Abbildung 58 auf folgender Seite). Der DIRECT Algorithmus gehört wie der Sbplx Algorithmus zu der Downhill-Simplex-Methode, welche unter dem diskreten Problem am besten zu performen scheint. Während der DIRECT Algorithmus den besten Fitnesswert liefert, gehört der Sbplx Algorithmus mit einer Rechenzeit von 9:41 Minuten mit zu den schnellsten unter den hier getesteten.

Der RBFOpt Algorithmus wendet zur Suche des Optimus die Ersatzmodell-Methode an. Die lange Rechenzeit von 1:03:43 Stunde lässt darauf schließen, dass die Approximation eines Problems mit Hilfe von differenzierbaren Funktionen für die Optimierung von diskreten

Optimierungsproblemen nicht geeignet ist. Bis ein akzeptables Ersatzmodell erstellt ist, müssen die Ersatzfunktionen aufgrund der diskreten Variablen zu oft angepasst werden.

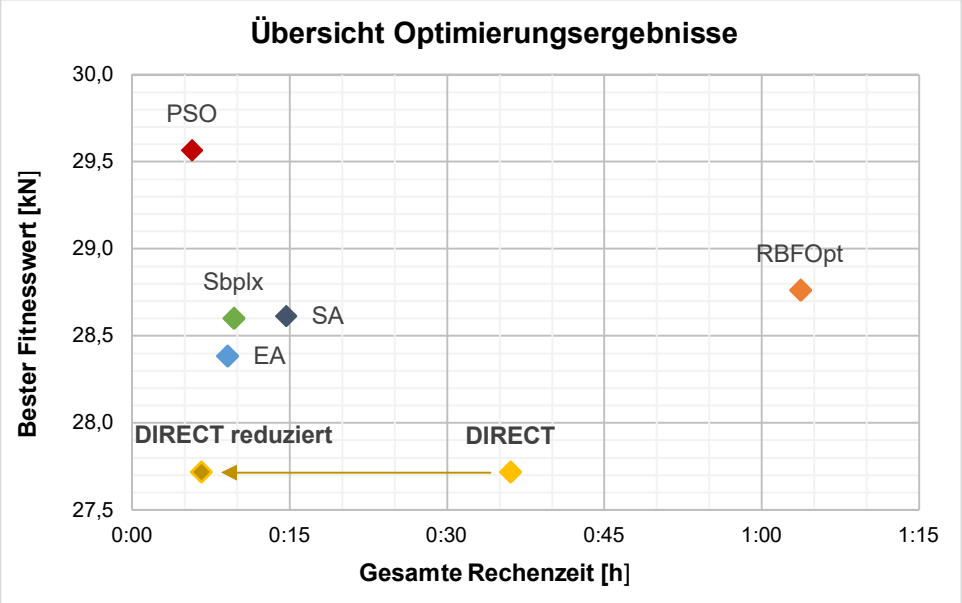


Abbildung 58: Übersicht Optimierungsergebnisse: Fachwerkträger, variable Zugstreben-Anzahl

Für diskrete Optimierungsprobleme mittlerer Größe können auf Grundlage dieses Benchmark Tests Algorithmen der Downhill-Simplex-Methode in Verbindung mit einer Eingrenzung des Variablenraums empfohlen werden. Wird vor dem Optimierungsprozess der Variablenraum durch das in dieser Arbeit vorgestellte Verfahren eingegrenzt, können die Rechenzeiten durchaus mit denen der metaheuristischen Algorithmen mithalten.

An dieser Stelle wird nochmal auf Andrew Conn, Katya Scheinberg und Luis Vicente [3] verwiesen, die die Metaheuristik als „method of last resort“ beschreiben. Wann immer Algorithmen anderer Methoden ähnliche oder sogar bessere Ergebnisse liefern, sollte auf die Verwendung von Metaheuristiken verzichtet werden. Da in diesem Benchmark Test weder der EA noch der SA oder PSO Algorithmus in puncto Rechenzeit oder Fitnesswert einen entscheidenden Vorteil mit sich bringt, wird von der Verwendung von Metaheuristiken bei diskreten Problemen mittlerer Größe abgeraten.

7.5 Pfahlgründung: Anordnung der Pfähle unter den Pfahlkopfbalken

Der dritte Benchmark Test, der im Rahmen dieser Arbeit durchgeführt wurde, basiert auf einem typischen ingenieurtechnischen Problem aus dem Hochbau. Je nach Baugrund und Belastungssituation können Bauwerke nicht flach gegründet werden. In diesen Fällen werden Pfahl- oder kombinierte Pfahl-Plattengründungen angeordnet. Neben der eigentlichen Bemessung der Gründungsbauteile ist die Anordnung der Pfähle unter dem Bauwerk eine weitere Aufgabe, die vom jeweiligen Tragwerksplaner gelöst werden muss. Vor allem bei reinen Pfahlgründungen stellt diese Aufgabe ein Problem dar, da die in den Baugrund abzuführenden Lasten nicht durch eine Fundamentplatte verteilt werden, sondern an der Stelle der tragenden Wände als Linienlasten auftreten. In folgendem Optimierungsproblem wird genau diese Situation durch ein typisches Mehrfamilienhaus mit Pfahlgründung simuliert.

Die Ausganssituation des Optimierungsproblems ist wie folgt: Aus dem Entwurf des Architekten sind bereits die Positionen der tragenden und aussteifenden Wände bekannt. Eine vorgezogene Lastermittlung ergab im untersten Geschoss eine konstante Wandlast von $200\ kN/m$, welche durch die Pfahlkopfbalken gesammelt und an die Pfähle weitergeleitet wird. Die Pfähle sind für den Lastabtrag in den Baugrund zuständig.

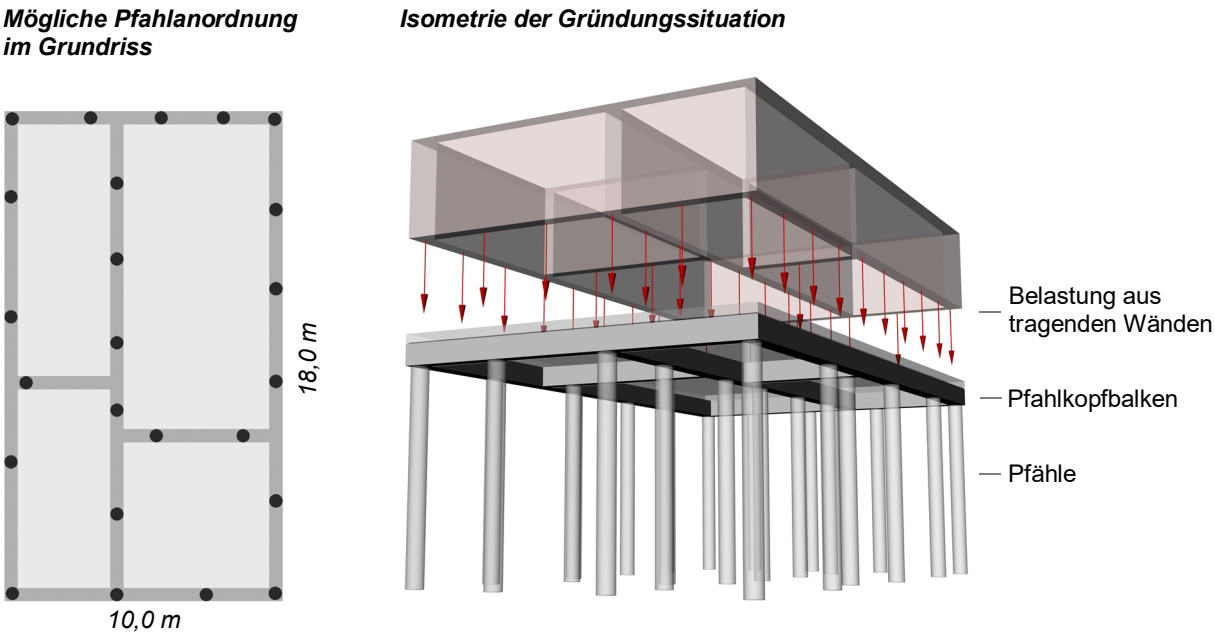


Abbildung 59: Ausgangssituation des Optimierungsproblems: Pfahlgründung

Optimiert werden soll die Anordnung der Pfähle unter den Pfahlkopfbalken hinsichtlich zweier Ziele. Zum einen soll jeder Pfahl bei konstanter Länge und konstantem Durchmesser das gleiche Lastniveau beziehungsweise den gleichen Ausnutzungsgrad erfahren, wobei zum

anderen die Verformung der Pfahlkopfbalken so gering wie möglich gehalten werden soll. Die Implementierung beider Ziele ist wichtig, da ein gleiches Lastniveau keine gleichmäßige Verteilung der Pfähle und damit keine geringen Verformungen garantiert. Da ein akzeptables Optimierungsziel nur unter Beachtung beider Ziele erreicht werden kann, sind sie als gleichwertig anzusehen und keines der beiden kann als Nebenbedingung abgewertet werden. Um das zweikriterielle Problem in ein einkriterielles umzuwandeln, kann aufgrund der gleichwertigen Ziele keine Penalty-Funktion angewendet werden. Stattdessen werden die Fitnesswerte beider Ziele getrennt voneinander berechnet und anschließend gewichtet miteinander addiert. Über die Gewichtung wird sichergestellt, dass beide Fitnesswerte den resultierenden Fitnesswert in gleichem Maße beeinflussen. Der Gewichtungsfaktor muss in den meisten Fällen empirisch ermittelt werden. Für diesen Benchmark Test hat sich nach einigen Testoptimierungen herausgestellt, dass keiner der beiden Fitnesswerte erhöht werden muss und somit der Gewichtungsfaktor *1,0* verwendet werden kann.

Die Grasshopper Definition des Benchmark Tests ist als Schemadarstellung in Anhang A, oder in digitaler Form in Anlage C zu finden.

Spezieller Solver

Für die Berechnung der beiden Fitnesswerte wurde zunächst in Grasshopper 3D ein parametrisches Modell der Pfahlgründung erstellt, welches aus einfachen Linien und Punkten besteht. Anschließend wurden die Linien mit dem Plug-In Karamba 3D diskretisiert und zu Balken, entsprechend der Pfahlkopfbalken zusammengesetzt. Die im Grasshopper Modell definierten Punkte entsprechen den Positionen der Pfähle und markieren die Auflagerpunkte der Balken, siehe Abbildung 60.

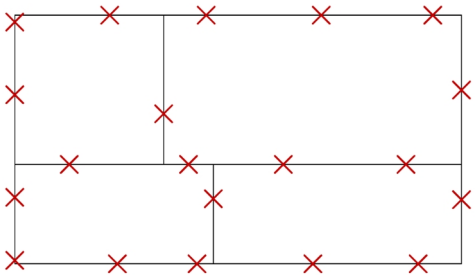


Abbildung 60: Parametrisches Modell der Pfahlgründung

Fitnesswerte

Wie bereits beschrieben, werden für dieses Optimierungsproblem zwei Fitnesswerte unabhängig voneinander berechnet. Der erste Fitnesswert F_η beschreibt die absolut mittlere Abweichung vom optimalen Lastniveau. F_η erreicht dann ein Minimum, wenn alle Pfähle das

gleiche Lastniveau erreicht haben und somit gleich ausgenutzt sind. Für die Berechnung von F_η werden zunächst folgende Basiswerte ermittelt:

Konstante Linienlast aus den tragenden Wänden	$200\ kN/m$
Laufender Meter tragende Wand	$(10 + 18\ m) = 84\ m$
$\sum V_z$: Gesamte, in den Baugrund weiterzuleitende Last	$200\ kN/m \cdot 84\ m = 16.800\ kN$
n : Pfahlanzahl	20
N_{SOLL} : Optimales Lastniveau	$16.800\ kN\ /\ 20 = 840\ kN$

Auf Grundlage der Basiswerte lässt sich F_η, die absolut mittlere Abweichung vom optimalen Lastniveau, wie folgt berechnen:

$$F_\eta = \frac{\left|\sum_{i=1}^{n} N_{i,IST} - N_{SOLL}\right|}{n} = \frac{\left|\sum_{i=1}^{20} N_{i,IST} - 840kN\right|}{20}$$

Der zweite Fitnesswert F_w beschreibt die Verformung der Pfahlkopfbalken. Da aufgrund des komplexen statischen Systems der Pfahlkopfbalken keine absolute Verformungsgrenze definiert werden kann, wird die Verformungsenergie des Systems berechnet. Die Verformungsenergie entspricht der potentiellen Energie, die durch den Widerstand des Systems gegen die Verformung entsteht. Somit entspricht ein Minimum an potentieller Energie einer minimalen Verformung des Systems. Der Fitnesswert F_w wird am FE-Modell mit dem Modul Karamba 3D berechnet.

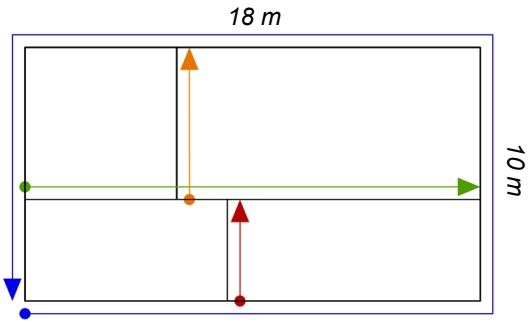
Der resultierende Fitnesswert entspricht der Summe aus F_η und F_w:

$$F_{res} = F_\eta + F_w$$

Variablen

Jeder Pfahl erhält eine individuelle Variable, über den die Position unter den Pfahlkopfbalken gesteuert wird. Theoretisch kann damit jeder Pfahl jede Position unter den 84 Metern Pfahlkopfbalken einnehmen. Für die Berechnung des FE-Modells wurden jedoch die Pfahlkopfbalken in insgesamt 1.000 Schritte diskretisiert, sodass die Position eines Pfahls um jeweils 8,4 cm variiert werden kann.

Jeder der 20 Variablen wird ein Wertebereich von 0,000 bis 1,000 mit einer Schrittweite von 0,001 zugewiesen. Somit besitzt jede Variable die Ausprägung 1.000 und kann jede mögliche Position unter den Pfahlkopfbalken in 8,4 cm Schritten ansteuern. Nach welchem Prinzip die Position eines Pfahls mit nur einer Variable gesteuert wird, ist in folgender Abbildung zu dargestellt.

Gesamte Kopfbalkenlänge = 84 m	Variable n:	0,000 bis 1,000	0 bis 84 m
	wird in folgende Wertebereiche aufgeteilt		
	Blau	0,000 bis 0,666	0 – 56 m
	Grün	0,667 bis 0,881	56 – 74 m
	Rot	0,881 bis 0,929	74 – 78 m
	Organge	0,930 bis 1,000	78 – 84 m

Nach dem obigen Prinzip kann durch die Variation der Variablen ein Pfahl jede mögliche Position unter den Pfahlkopfbalken einnehmen. Aus ingenieurtechnischer Sicht ist das jedoch nicht zielführend. Es wäre sinnvoller, wenn jeder Pfahl aus seiner jeweiligen Ausgangsposition nur um wenige Meter verschoben werden kann. Dennoch wurde sich im Rahmen des Benchmark Tests bewusst für das obenstehende Prinzip entschieden, da so eine große Anzahl an lokalen Extremwerten generiert wird. Geht man davon aus, dass nur eine optimale Pfahlanordnung für dieses Problem existiert, kommt der Extremwert bei 20 Pfählen in der Fitnesslandschaft insgesamt $20! = 2{,}43 \cdot 10^{18}$ Mal vor.

Dieses Vorgehen wurde zum einen gewählt, um die Black-Box Algorithmen gegen ein hochgradig lokales Optimierungsproblem testen zu können, und zum anderen, da dieses Variationsprinzip so in der Praxis Anwendung finden könnte. Dadurch, dass jeder Variable der gesamte Variationsbereich zur Verfügung steht, muss man sich vor dem Optimierungsprozess keine Gedanken über eine sinnvolle Eingrenzung machen. Des Weiteren ist der so erstellte spezielle Solver universell auf alle Gründungs- und Wandgeometrien anwendbar.

Der 20-dimensionale Variablenraum ist wie folgt definiert:

Tabelle 19: Variablenraum: Pfahlgründung

Variable		Intervall / Wertebereich	Auspräung
Position des Pfahls	*n = 1, 2, … , 20*	*0,000 – 1,000*	1000

Aufgrund der geringen Variablenschrittweite von *0,001*, was dem Äquivalent zu einer Verschiebung um 8,4 cm entspricht, kann das Optimierungsproblem als quasi-stetig definiert werden.

Kategorisierung des Optimierungsproblems

Das Optimierungsproblem kann aufgrund der oben beschriebenen Eigenschaften wie folgt kategorisiert werden.

Tabelle 20: Kategorisierung: Pfahlgründung

Dimensionalität	20
Größe des Problems	$1000^{20} = \mathbf{1 \cdot 10^{60}}$ mögliche Lösungen
Beschaffenheit	stetig
Extremwertsuche	hochgradig lokal (mindestens 20! lokale Extremwerte)
Fitnesswert	Summe aus der absoluten mittleren Abweichung und der Verformungsenergie
Penalty-Funktion	keine
Spezieller Solver	FE-M Berechnung der Pfahlkopfbalken und der resultierenden Lagerkräfte

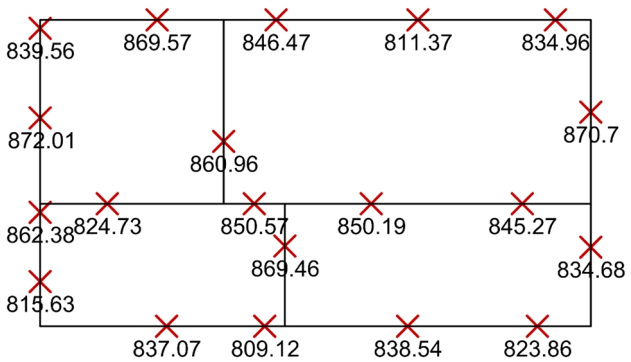
7.5.1 Optimierungsergebnisse

In folgender Tabelle sind die Ergebnisse der einzelnen Optimierungsprozesse zusammengefasst dargestellt.

Tabelle 21: Übersicht Optimierungsergebnisse

	EA	SA	PSO	RBFOpt	DIRECT	Sbplx
Rechenzeit	1:47:43	1:53:32	14:48	-	-	22:58
Fitnesswert [-]	38,56	55,86	86,85	-	-	65,30

Aufgrund des sehr komplexen Problems und der großen Anzahl an lokalen Extremwerten, ist eine größere Streuung unter den Rechenzeiten und Fitnesswerten zu bemerken. Der EA Algorithmus liefert mit einem Fitnesswert von 38,56 das mit Abstand beste Ergebnis, wobei die Rechenzeit von 1:47:43 Stunde zu den längsten gehört. Das Ergebnis des EA Algorithmus ist in Form der Pfahlnormalkräfte (N_{IST}) in folgender Abbildung zu sehen.



$$N_{SOLL} = 840 \frac{kN}{Pfahl}$$

Abbildung 61: Ergebnis der Optimierung mit dem EA Algorithmus (N_{IST} [kN])

Sowohl der DIRECT Algorithmus als auch der RBFOpt Algorithmus konnten jeweils keine Lösung für das Problem finden, was zu erwarten war, da beide Algorithmen eine streng globale Suche betreiben, siehe Tabelle 4 auf Seite 91. Weiterhin ist zu bemerken, dass bis auf den Sbplx alle anderen Algorithmen erst jenseits der 100-Sekunden-Marke den Fitnesswert verbessern können, siehe folgende Abbildung.

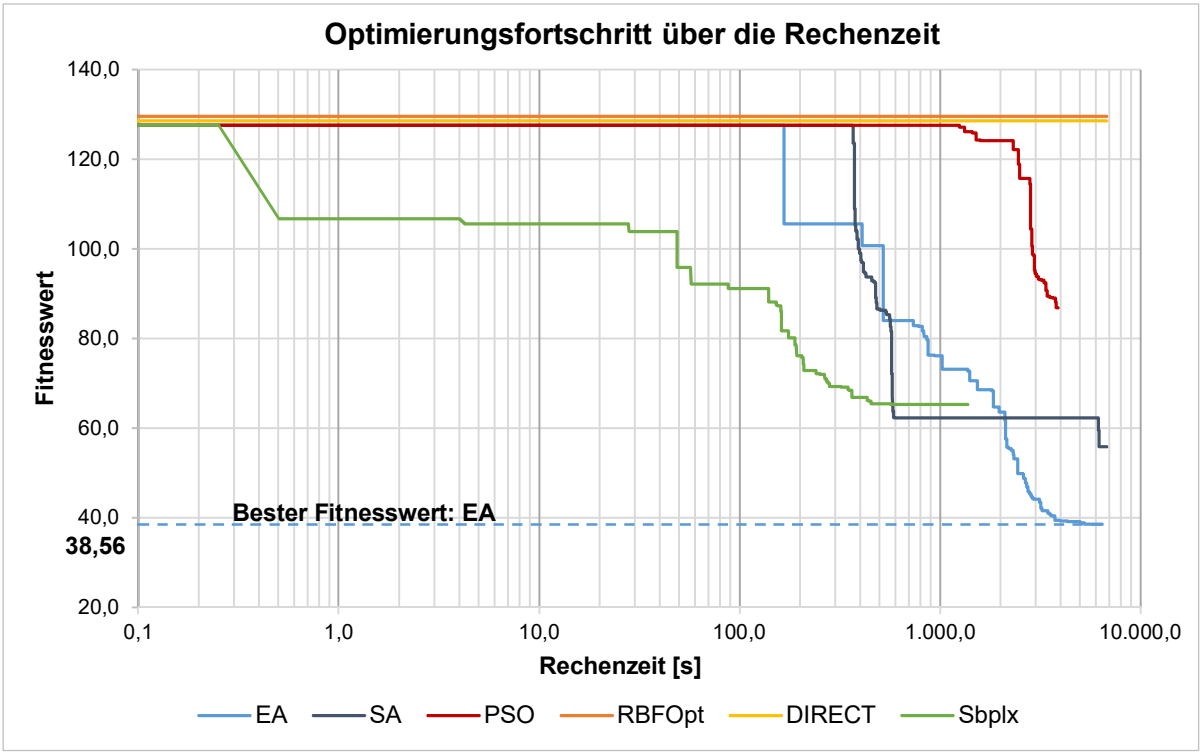


Abbildung 62: Optimierungsfortschritt: Pfahlgründung

7.5.2 Eingrenzung des Variablenraums

Aufgrund der großen Anzahl an lokalen Extremwerten ist zu erwarten, dass die Typologie der Fitnesslandschaft eine Vielzahl an Bergen und Hügeln aufweist. Abbildung 63 zeigt in abstrahierter Form die zu erwartende Fitnesslandschaft.

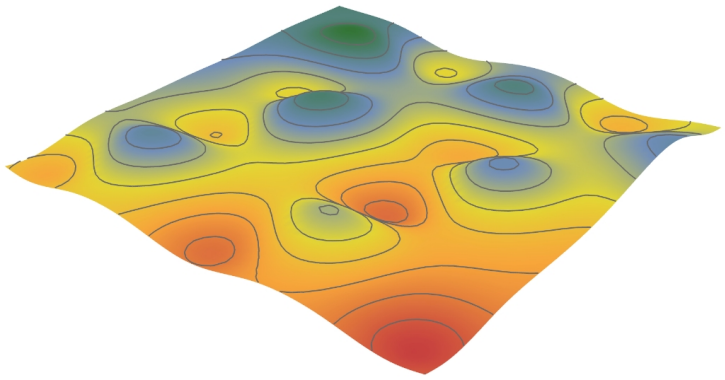


Abbildung 63: Fitnesslandschaft eines lokalen Optimierungsproblems

Analog zur Fitnesslandschaft verhält sich die Typologie des Variablenraums. Zur Verdeutlichung wurde der Variablenraum des Optimierungsproblems gemäß der Methode der n-1-dimensionalen Gruppierung visualisiert, siehe Abbildung 64. Jedes farbige Quadrat der Karte spiegelt ein 20-dimensionales Variablenset mit berechnetem Fitnesswert wieder. Hier ist, analog zur Fitnesslandschaft, eine Vielzahl an lokalen Extremwerten zu erkennen (grün), die nahezu gleichmäßig über den gesamten Variablenraum verteilt sind.

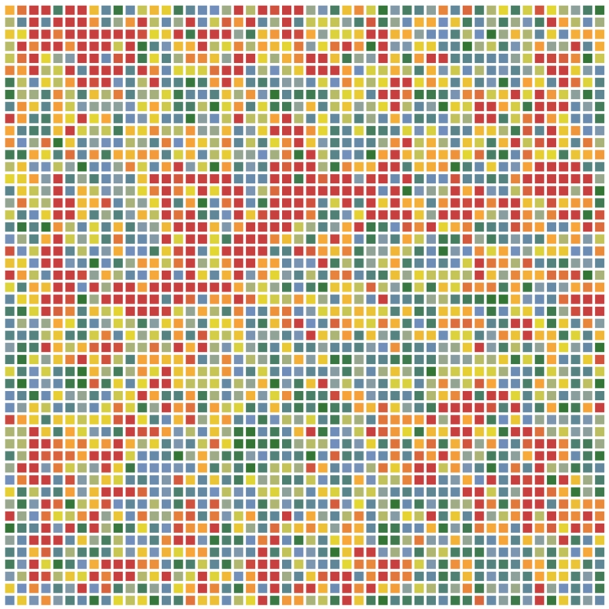


Abbildung 64: Visualisierter Variablenraum: Pfahlgründung

An dieser Stelle ist es wichtig zu erwähnen, dass, obwohl es auf den ersten Blick anders scheint, die obige Visualisierung des Variablenraums erfolgreich war. Die vermeintliche Unordnung der Karte war zu erwarten, sie spiegelt die vielen lokalen Extremwerte des Optimierungsproblems wieder. Anders als bei den Benchmark Tests in Absatz 7.3 und 7.4 können in diesem Variablenraum keine Regionen definiert und analysiert werden, die ausschließlich gute Ergebnisse liefern. Eine Reduzierung der Problemgröße durch eine Eingrenzung des Variablenraums mit der Methode der n-1-dimensionalen Gruppierung ist deshalb nicht möglich. Um den Benchmark Test dennoch mit einer reduzierten Problemgröße durchführen zu können, wird der Variablenraum konstruktiv sinnvoll eingegrenzt.

Die Grundlage dafür ist die Überlegung, dass vermutlich ein relativ gutes Ergebnis erzielt werden kann, wenn alle 20 Pfähle gleichmäßig unter den Pfahlkopfbalken angeordnet werden. In diesem Fall würde damit alle $84\,m\,/\,20 = 4{,}2\,m$ ein Pfahl angeordnet werden. Ausgehend von dieser Startposition werden die Variablen so eingegrenzt, dass sich die Position eines Pfahls um maximal plus/minus 4,2 Meter verändern lässt. Für den Variablenraum bedeutet das folgendes:

Tabelle 22: Eingegrenzter Variablenraum

Variable *i*	Startwert	Intervall / Wertebereich	Äquivalenter Bereich	Auspräung
1	*0,050*	*0,000 – 0,100*	0,0 – 8,4 m	100
2	*0,100*	*0,050 – 0,150*	4,2 – 12,6 m	100
⋮	⋮	⋮	⋮	⋮
20	*0,950*	*0,900 – 1,000*	75,6 – 84,0 m	100

Die Größe des Optimierungsproblems wurde durch die Einschränkung des Variablenraums von $1000^{20} = \mathbf{1 \cdot 10^{60}}$ mögliche Lösungen auf $100^{20} = \mathbf{1 \cdot 10^{40}}$ mögliche Lösungen reduziert, was einer Reduktion des Problems um $99,\overline{9}\ \%$ entspricht.

7.5.3 Optimierungsergebnis mit eingegrenztem Variablenraum

Die Optimierung des Problems mit eingegrenztem Variablenraum wird sowohl mit dem EA als auch mit dem Sbplx Algorithmus durchgeführt. Die Ergebnisse der Optimierungen sind in folgender Tabelle zu finden.

Tabelle 23: Vergleich: Ergebnisse mit vollständigem und reduziertem Variablenraum

	Vollständiger Variablenraum	Reduzierter Variablenraum	Verbesserung
Algorithmus	**EA**		
Iterationen	40102	11567	
Rechenzeit	1:47:43	40:31	**62,36 %**
Fitnesswert	38,56 [-]	37,16 [-]	**3,63 %**
Algorithmus	**Sbplx**		
Iterationen	2292	278	
Rechenzeit	22:58	12:44	**44,56 %**
Fitnesswert	65,30 [-]	37,38 [-]	**42,76 %**

Durch die Reduzierung der Problemgröße infolge eines eingegrenzten Variablenraums verkürzt sich die Rechenzeit des EA Algorithmus um rund zwei Drittel, während sich der Fitnesswert nur geringfügig verbessert. Anders ist es bei den Ergebnissen des Sbplx Algorithmus, hier verbessert sich sowohl die Rechenzeit als auch der Fitnesswert um rund 40 %.

7.5.4 Schlussfolgerung

Das Problem der optimalen Pfahlanordnung wurde zunächst so generiert, dass es mit $1 \cdot 10^{60}$ möglichen Lösungen sehr komplex und mit $2{,}43 \cdot 10^{18}$ Extremwerten hochgradig lokal ist. Diese Problemeigenschaften spiegeln sich in den Ergebnissen der Optimierungsprozesse wieder. Die global suchenden RBFOpt und DIRECT Algorithmen konnten mit den lokalen Fitnesslandschaften nicht umgehen und haben keine Lösungen gefunden. Die Suche wurde jeweils nach drei Stunden manuell abgebrochen.

Die Verwendung von metaheuristischen Algorithmen garantiert aufgrund ihrer Struktur und Suchmethode, dass schlussendlich ein Ergebnis gefunden wird. Jedoch geht diese „Erfolgsgarantie“ zu Lasten der Ergebnisqualität beziehungsweise der Rechenzeit. Der EA Algorithmus erreicht bei diesem Test den bei weitem besten Fitnesswert von 38,56, allerdings benötigt der Algorithmus für die Optimierung 1:47:43 Stunden. Rechenzeiten in dieser Größenordnung sind für eine Anwendung im Ingenieursalltag nicht praktikabel.

Für die Problemkategorie ist der Sbplx Algorithmus grundsätzlich am besten geeignet, da er auf die Suche lokaler Extremwerte spezialisiert ist. Mit nur 22:58 Minuten konvergiert der Sbplx Algorithmus zwar am schnellsten, allerdings gehört der Fitnesswert von 65,30 zu den schlechtesten in diesem Test.

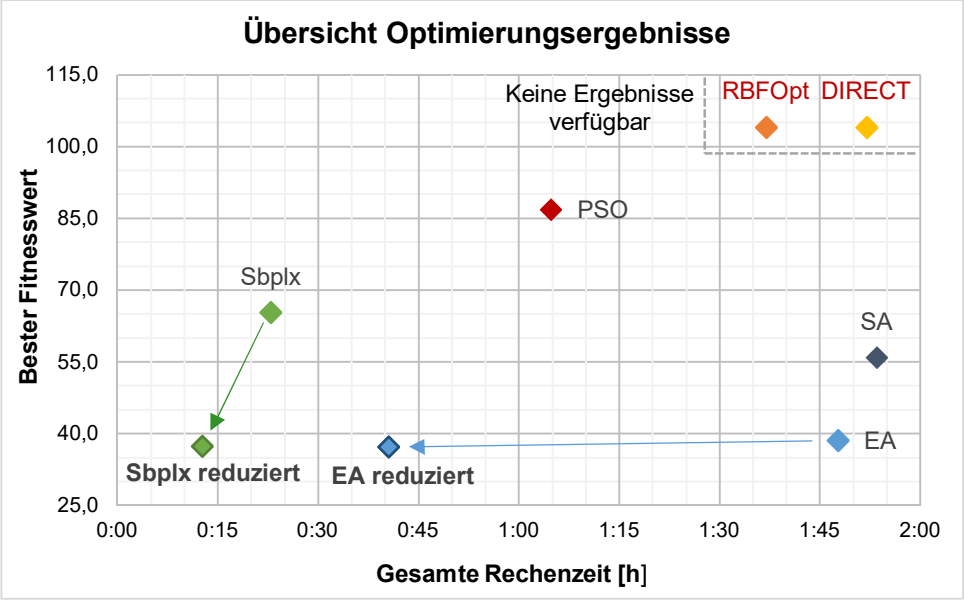


Abbildung 65:Übersicht Optimierungsergebnisse: Pfahlgründung

Nach der Reduzierung der Problemgröße durch die Eingrenzung des Variablenraums wurde das Problem nochmals mit dem EA und dem Sbplx Algorithmus optimiert. Wie sich die

Performance der Algorithmen verbessert hat, ist in Abbildung 65 zu sehen. Hervorzuheben ist vor allem der Sbplx Algorithmus. Hier konnte sowohl die Rechenzeit als auch der Fitnesswert so weit verbessert werden, dass sie die Besten des gesamten Tests sind.

Für eine detaillierteres Verständnis werden die Verläufe des EA und Sbplx Optimierungsprozesses mit vollständigem und reduziertem Variablenraum in folgender Abbildung miteinander verglichen. Hier fällt auf, dass die Optimierung des EA Algorithmus mit vollständigem und reduziertem Variablenraum nahezu gleich verläuft (blau), der Prozess ist lediglich auf der Zeitachse parallel verschoben. Das lässt darauf schließen, dass bei den Metaheuristiken ein linearer Zusammenhang zwischen der Problemgröße und der Rechenzeit besteht. Im Umkehrschluss bedeutet die Eingrenzung des Variablenraums eine Reduktion der Rechenzeit, jedoch keine deutliche Verbesserung des Fitnesswertes. Anders verhält es sich bei den Optimierungsverläufen des Sbplx Algorithmus. Hier verbessert sich, wie in Abbildung 66 zu sehen ist, sowohl die Rechenzeit als auch der Fitnesswert.

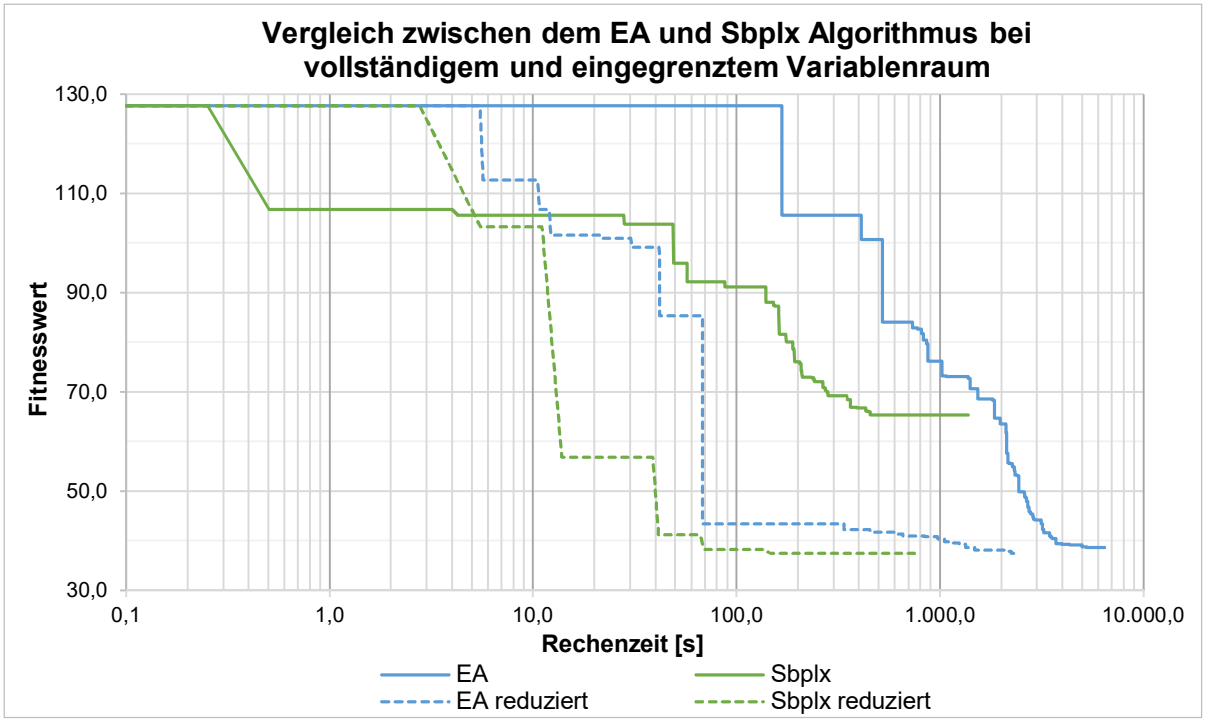


Abbildung 66: Detaillierter Vergleich der Optimierungsprozesse mit eingegrenztem Variablenraum

Die Analyse der Ergebnisse zeigt deutlich, dass bei einem derart großen und komplexen Optimierungsproblem eine vorgeschaltete Eingrenzung des Variablenraums nahezu unerlässlich ist. Vergleicht man die Ergebnisse unabhängig des verwendeten Algorithmus miteinander, wird die Rechenzeit durch die Eingrenzung von 1:47:43 Stunden (EA) auf 14:44 Minuten (Sbplx) verkürzt und dabei der Fitnesswert um rund 3 % verbessert.

7.6 Fazit der Benchmark Tests

Eine detaillierte Auswertung und Analyse der Optimierungsergebnisse ist jeweils in den Absätzen der Benchmark Tests zu finden. An dieser Stelle werden die Erkenntnisse aus den Analysen lediglich zusammengefasst und darauf basierend eine Empfehlung für die am besten anzuwendende Algorithmus-Art in Abhängigkeit der Problemkategorien ausgesprochen.

Die Auswertung aller drei Benchmark Tests hat gezeigt, dass eine Reduzierung der Problemgröße vor dem Optimierungsprozess einen großen Einfluss auf die Ergebnisse hat. Es kann sowohl die Qualität der Ergebnisse deutlich verbessert als auch die benötigte Rechenzeit um bis zu 80 % verkürzt werden. Auf dieser Grundlage wird empfohlen, die Größe und damit auch Komplexität jedes Optimierungsproblems vorab zu reduzieren. Es hat sich herausgestellt, dass die im Rahmen dieser Arbeit entwickelte Methode der n-1-dimensionalen Gruppierung ein leistungsstarkes Tool zur Reduzierung von Problemgrößen ist. Durch die Anwendung dieser Methode kann der Variablenraum eines n-dimensionalen Problems vor der Optimierung visualisiert und anschließend ohne einen Verlust an möglichen guten Lösungen eingegrenzt werden. Die Optimierung der Benchmark Tests in dieser Arbeit wurden sowohl mit vollständigem als auch mit reduziertem Variablenraum durchgeführt. Die folgenden Auswertungen und Schlussfolgerungen beziehen sich jedoch immer auf die Ergebnisse der Optimierungen mit reduziertem Variablenraum.

Die Auswertungen der Benchmark Tests haben weiterhin ergeben, dass die Algorithmen entsprechend ihrer Methoden in zwei Kategorien einzuteilen sind. Die erste Kategorie beinhaltet alle metaheuristischen Algorithmen, in dieser Arbeit ist das der EA, SA und PSO Algorithmus. Alle spezifischeren Algorithmen der Ersatzmodell- und Downhill-Simplex-Methode sind der zweiten Kategorie zuzuordnen. Der Grund für diese Unterscheidung ist, dass Metaheuristiken prinzipiell für die Optimierung jeder Problemkategorie angewendet werden können, allerdings müssen dann in puncto Ergebnisqualität und Rechenzeit Abstriche in Kauf genommen werden. Welcher metaheuristische Algorithmus angewendet wird, spielt dabei nur eine untergeordnete Rolle, da die Suche nach dem Optimum nicht auf einem deterministischen Muster basiert. Ergebnisse sind somit nicht reproduzierbar und dadurch auch untereinander nur bedingt vergleichbar. Auf der anderen Seite stehen die spezifischeren Algorithmen der Ersatzmodell- und Downhill-Simplex-Methode. Unter ihnen existiert mindestens ein Algorithmus, der gleichgut oder sogar besser performt als jede Metaheuristik. Hier gilt das No-Free-Lunch-Theorem, nachdem keine allgemeine Metaheuristik besser ist, als jeder spezifischere Optimierungsalgorithmus [4].

Aus diesem Grund werden in der untenstehenden Tabelle die Metaheuristiken bei der Wahl des jeweils geeignetsten Optimierungsalgorithmus außenvor gelassen. Stattdessen wird das Verhältnis zwischen dem besten spezifischen Algorithmus und den Metaheuristiken in Bezug auf Fitnesswert und absolute Rechenzeiten angegeben.

Tabelle 24: Übersicht über die Analyse der Benchmark Test Ergebnisse

Problemkategorie	**Geeignetster Optimierungsalgorithmus**	**Ergebnis im Vergleich zur Optimierung mit einer Metaheuristik**	
		Fitnesswert	Rechenzeit
groß **1.** stetig global	**RBFOpt** Ersatzmodel-Methode global Extremwertsuche	**≈ 0 %**	**- 5:56 Minuten** Metaheuristik = 2:29 min RBFOpt = 8:25 min
mittelgroß **2.** diskret global	**DIRECT** Downhill-Simplex Methode global Extremwertsuche	**+ 2,4 %**	**+ 2:29 Minuten** Metaheuristik = 9:04 min DIRECT = 6:35 min
sehr groß **3.** stetig lokal	**Sbplx** Downhill-Simplex Methode lokale Extremwertsuche	**≈ 0 %**	**+ 27:47 Minuten** Metaheuristik = 40:31 min Sbplx = 12:44 min

Problemkategorie 1

Optimierungsprobleme, deren Fitnesslandschaft als stetig und global kategorisiert werden kann, gelten als die am einfachsten zu optimierenden Probleme. Die Benchmark Tests haben gezeigt, dass sich für die Optimierung solcher Probleme der RBFOpt Algorithmus am besten eignet. Aufgrund der stetigen Typologie der Fitnesslandschaft lässt sich das nichtlineare Problem gut durch differenzierbare Funktionen approximieren. Diese Methode nutzt der RBFOpt Algorithmus indem ein geeignetes differenzierbares Ersatzmodel des eigentlichen Problems erstellt und anschließend ausgewertet wird. Bei den entsprechenden Benchmark Tests konnten die metaheuristischen Algorithmen den Fitnesswert des RBFOpt Algorithmus nicht verbessern, kamen allerdings mit einer Rechenzeit von knapp sechs Minuten etwas schneller zum Ziel. Daraus kann jedoch nicht abgeleitet werden, dass Metaheuristiken im Allgemeinen schneller sind, da die Rechenzeit nahezu linear von der Problemgröße abhängig ist, was bei dem RBFOpt Algorithmus nicht der Fall ist.

Problemkategorie 2

Bei globalen und diskreten Problemeigenschaften hat sich der DIRECT Algorithmus der Downhill-Simplex-Methode am leistungsstärksten erwiesen. Während dem Optimierungsprozess zerlegt der Algorithmus das Problem in immer kleiner werdende Rechtecke, bis das Problem direkt gelöst werden kann (*DIRECT Kurzform für DIviding RECTangles*). Aufgrund dieses Suchmusters geben die Entwickler von DIRECT an, dass die

beste Performance bei Problemen niedriger Dimensionalität zu erwarten ist [8]. Das hier getestete Optimierungsproblem (siehe Absatz 7.4) zählt mit der Dimension fünf noch zu den Problemen niedriger Dimensionalität. Ob der DIRECT Algorithmus bei höheren Dimensionen immer noch gut performt, müsste in weiteren Tests untersucht werden. An dieser Stelle kann der DIRECT Algorithmus für die Optimierung globaler und diskreter Probleme niedriger Dimensionalität empfohlen werden.

Problemkategorie 3

Die dritte Problemkategorie wurde durch ein 20-dimensionales Problem mit lokalen Extremwerten und stetiger Fitnesslandschaft repräsentiert. Sowohl der RBFOpt als auch der DIRECT Algorithmus konnten während dem Optimierungsprozess den Fitnesswert nicht verbessern, was zu erwarten war, da beide Algorithmen auf eine rein globale Extremwertsuche ausgelegt sind. Im Gegensatz dazu betreibt der Sbplx Algorithmus mit Hilfe des Nelder-Mead-Verfahrens eine streng lokale Suche und liefert deshalb das mit Abstand beste Optimierungsergebnis. Die relativ kurze Rechenzeit von nur 12:44 Minuten zeigt, dass auch im Hinblick auf die hohe Dimension und den großen Variablenraum der Sbplx Algorithmus sehr gut performt. Der beste metaheuristische Algorithmus erreicht in diesem Test zwar den gleichen Fitnesswert, benötigt für den Optimierungsprozess allerdings über 40 Minuten.

Zusammenfassung

Abschließend kann festgehalten werden, dass von der Benutzung metaheuristischer Algorithmen nicht generell abgeraten werden kann. Spielt die Rechenzeit keine Rolle oder sind keinerlei Informationen über das Problem vorhanden, sind Metaheuristiken durchaus zielführend. Sie liefern die gleiche Ergebnisqualität wie die spezifischeren Algorithmen der Ersatzmodell- und Downhill-Simplex-Methode. Spielt die Rechenzeit jedoch eine Rolle, ist vor allem bei der Optimierung großer und komplexer Probleme die Anwendung von Metaheuristiken nicht zu empfehlen.

Die spezifischeren Algorithmen der Ersatzmodell- und Downhill-Simplex-Methode gelangen in den meisten Fällen schneller ans Ziel, benötigen allerdings Informationen über die Beschaffenheit des Problems beziehungsweise der Fitnesslandschaft. Die Eigenschaften stetig oder diskret, global oder lokal sind dabei die wichtigsten Faktoren bei der Auswahl eines geeigneten Algorithmus.

8 Zusammenfassung und Ausblick

Zusammenfassung

Der zentrale Lösungsansatz dieser Arbeit, nach dem die Visualisierung ein Bindeglied zwischen der nichtlinearen Optimierung und deren Anwendung auf Probleme im konstruktiven Ingenieurbau ist, wurde in der Erarbeitung aller Kapitel strikt verfolgt. Die Grundlagen wurden zu Beginn umfassend vorgestellt, sodass die Entwicklung der n-1-dimensionalen Methode und die anschließenden Schlussfolgerungen auch ohne Vorkenntnisse zum Thema nichtlineare Optimierung verstanden und nachvollzogen werden können.

Es hat sich herausgestellt, dass für die Visualisierung von Optimierungsproblemen bekannte Verfahren, wie beispielsweise die Methode der multidimensionalen Skalierung, nicht geeignet sind. Die bekannten Verfahren können nicht mit den enormen Datenmengen umgehen, die bei der Optimierung nichtlinearer Probleme entstehen. Aus diesem Grund wurde im Rahmen dieser Arbeit ein neues Verfahren zur Visualisierung von n-dimensionalen Daten entwickelt, welches auf der Verwendung neuronaler Netze basiert. Mit dem Verfahren, welches *n-1-dimensionale Gruppierung* genannt wird, können Daten beliebig hoher Dimension auf die Dimension 2 oder 3 reduziert und visualisiert werden. Das Verfahren zeichnet sich vor allem durch eine schnelle und unkomplizierte Anwendung aus.

Während die Methode der *n-1-dimensionalen Gruppierung* ursprünglich zur Visualisierung von Fitnesslandschaften entwickelt wurde, stellte sich heraus, dass mit ihr auch ganze Variablenräume in Abhängigkeit der Fitnesswerte visualisiert werden können. Mit diesem Ansatz lassen sich bereits vor dem Optimierungsprozess Variablenräume eingrenzen und Problemgrößen reduzieren.

Für die Beantwortung der Frage, welche Arten von Black-Box Algorithmen bezüglich der Optimierung ingenieurtechnischer Probleme gut geeignet sind, wurden im Rahmen dieser Arbeit drei eigens entwickelte Benchmark Tests durchgeführt. Mit den Testergebnissen konnten klare Empfehlungen für die Verwendung bestimmter Algorithmen in Abhängigkeit der Problemkategorie ausgesprochen werden. Des Weiteren konnte mit den Benchmark Tests die Leistungsfähigkeit der Methode der *n-1-dimensionalen Gruppierung* getestet werden. Die Ergebnisse haben gezeigt, dass eine Reduzierung der Problemgröße durch die Methode der *n-1-dimensionalen Gruppierung* sowohl die Rechenzeit als auch den Fitnesswert positiv beeinflusst.

Abschließend kann festgehalten werden, dass die Anwendung nichtlinearer Optimierungen in Verbindung mit der Methode der Visualisierung eine leistungsstarke Kombination ist, durch die zum einen Optimierungsprozesse transparenter und nachvollziehbarer werden und zum anderen qualitativ bessere Ergebnisse in kürzerer Rechenzeit erreicht werden.

Ausblick

Die durchgeführten Benchmark Tests liefern bezüglich der neu entwickelten Methoden und Ansätze bereits vielversprechende Ergebnisse, allerdings sind diese eher als Resultate einer Grundlagenforschung anzusehen. Um die Ergebnisse und Schlussfolgerungen dieser Arbeit zu bestätigen, sind weitere Tests mit gegebenenfalls umfangreicheren Problemstellungen notwendig.

Die Methode der n-1-dimensionalen Gruppierung wurde ausschließlich in der grafischen Entwicklungsumgebung von Grasshopper 3D erstellt. Das bedeutet, alle Modellierungen, Implementierungen und Auswertungen müssen auch in Form der Grasshopper eigenen Programmiersprache erfolgen, was gewisse Programmierkenntnisse voraussetzt. Um diese Hürde zu umgehen, wäre die Entwicklung einer grafischen Benutzeroberfläche (GUI) für die regelmäßige Anwendung in der Praxis hilfreich. In der GUI könnten aufgrund von Problemkategorien geeignete Algorithmen vorgeschlagen, sowie Eingrenzungen von Variablenräumen und Einstellungen zu Optimierungsprozessen vorgenommen werden. Um kein eigenständiges Programm neu entwickeln zu müssen, könnte die GUI sowohl mit Grasshopper für die Modellierung als auch mit einer Onlinebibliothek (beispielsweise NLopt [7]) für die Optimierung kommunizieren.

Literaturverzeichnis

Fachliteratur

[1] Sélley / Gyurecz / Janik / Körtélyesi: *Engineering Optimization, Course bulletin*. Budapest University of Technology and Economics, Faculty of Mechanical Engineering, 2012

[2] Albert, Andrej [Hrsg.]: *Schneider Bautabellen für Ingenieure*, 21. Auflage. Köln: Bundesanzeiger Verlag, 2014

[3] Conn, Andrew R. / Scheinberg, Katya / Vicente, Luis N.: *Introduction to Derivative-Free Optimization*. Philadelphia: Society for Industrial and Applied Mathematics, 2009

[4] Wolpert, David H. / Macready, William G.: *No Free Lunch Theorems for Search.* Santa Fe: The Santa Fe Institute, 1996

[5] Robert McNeel & Associates: *Rhinoceros 5: Benutzerhandbuch für Windows*. Veröffentlicht: 13.01.2015

[6] GECCO Black Box Optimization Competition: *https://bbcomp.ini.rub.de/*. Internetseite: Stand August 2018

[7] Onlinebibliothek für nichtlineare Optimierungsalgorithmen: *https://nlopt.readthedocs.io/en/latest/*. Internetseite: Stand September 2018

[8] Finkel, Daniel E.: *DIRECT Optimization Algorithm User Guide*. North Carolina State University, Center for Research in Scientific Computation, 2003

[9] Geiger, Karl / Kanzow, Christian: *Numerische Verfahren zur Lösung unrestringierter Optimierungsaufgaben*. Heidelberg: Springer-Verlag, 1999

[10] Cichocka, Judyta M. / Migalska Agata / Browne, Will N. / Rodriguez Edgar: *SILVEREYE – The Implementation of Particle Swarm Optimization Algorithm in a Design Optimization Tool.* In: Çağdaş / Özkar / Gül / Gürer [Hrsg.]: *Computer-Aided Architectural Design.* Future Trajectories. CAADFutures 2017. Communications in Computer and Information Science, vol 724. Singapore: Springer, 2017

[11] Wortmann, Thomas: *OPOSSUM - Introducing and Evaluating a Model-based Optimization Tool for Grasshopper*. In: Janssen / Loh / Raonic / Schnabel [Hrsg.]: *Protocols, Flows and Glitches, Proceedings of the 22nd International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA).* Hong Kong: The Association for Computer-Aided Architectural Design Research in Asia, 2017

[12] Costa, Alberto / Nannicini, Giacomo: *RBFOpt: an open-source library for black-box optimization with costly function evaluations*. Singapore: ETH Zurich, Future Cities Laboratory program und U.S.A.: IBM T.J. Watson Research Center

Normen und Regelwerke

[13] DIN Deutsches Institut für Normung e.V. [Hrsg.]: *Eurocode 0: Grundlagen der Tragwerksplanung*. Berlin: Beuth Verlag GmbH 2002

[14] DIN Deutsches Institut für Normung e.V. [Hrsg.]: *Eurocode 1: Einwirkungen – Einwirkungen auf Tragwerke –Teil 1-1: Allgemeine Einwirkungen auf Tragwerke – Wichten, Eigengewicht und Nutzlasten im Hochbau* Berlin: Beuth Verlag GmbH 2002

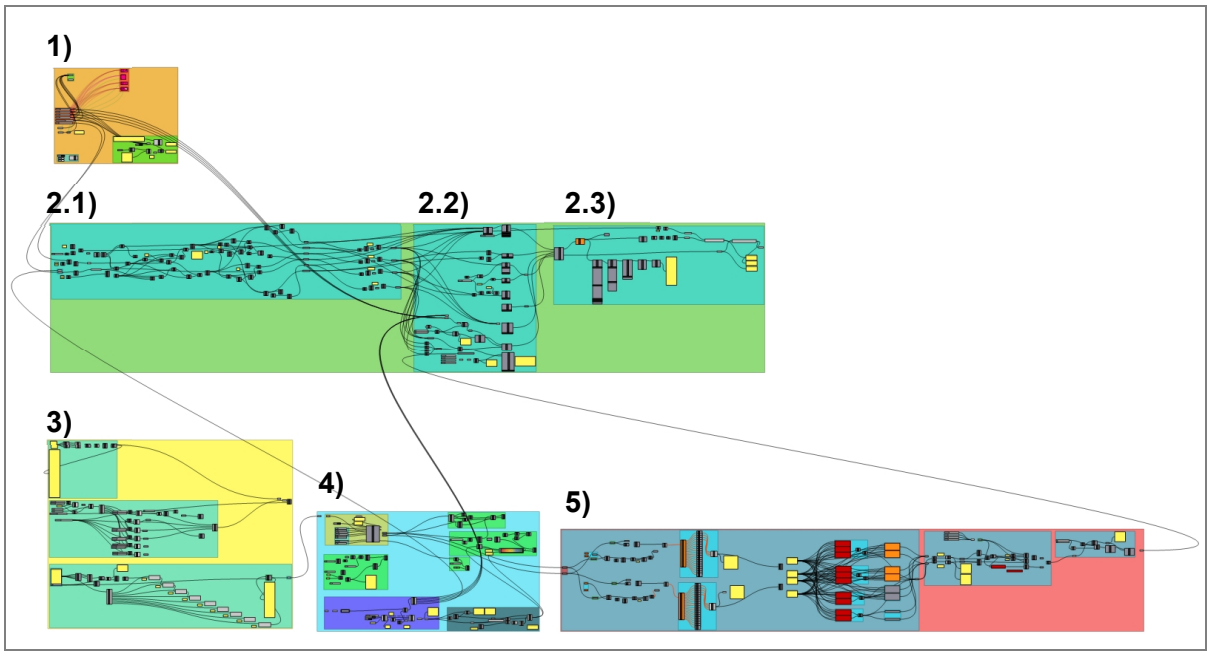
[15] DIN Deutsches Institut für Normung e.V. [Hrsg.]: *DIN 1025-3: Warmgewalzte I-Träger; Breite I-Träger, leichte Ausführung, IPBl-Reihe; Maße, Masse, statische Werte*. Berlin: Beuth Verlag GmbH 1994

Anlagenverzeichnis

Anlage A: Benchmark Test: Schema der Grasshopper Definition

Benchmark Test 1 und 2:

- Fachwerkträger: Variable Querschnittsabmessungen und variable Zugstrebenneigung
- Fachwerkträger: Variable Querschnittsabmessungen und variable Zugstreben-Anzahl



1) Variablen Input, Optimierung und Aufzeichnung der Optimierungsdaten

2) Spezieller Solver

2.1) Modellierung des parametrischen Fachwerkträgers

2.2) Erstellen des Finite-Element-Modells

2.3) Penalty-Funktion / Berechnung des Fitnesswerts

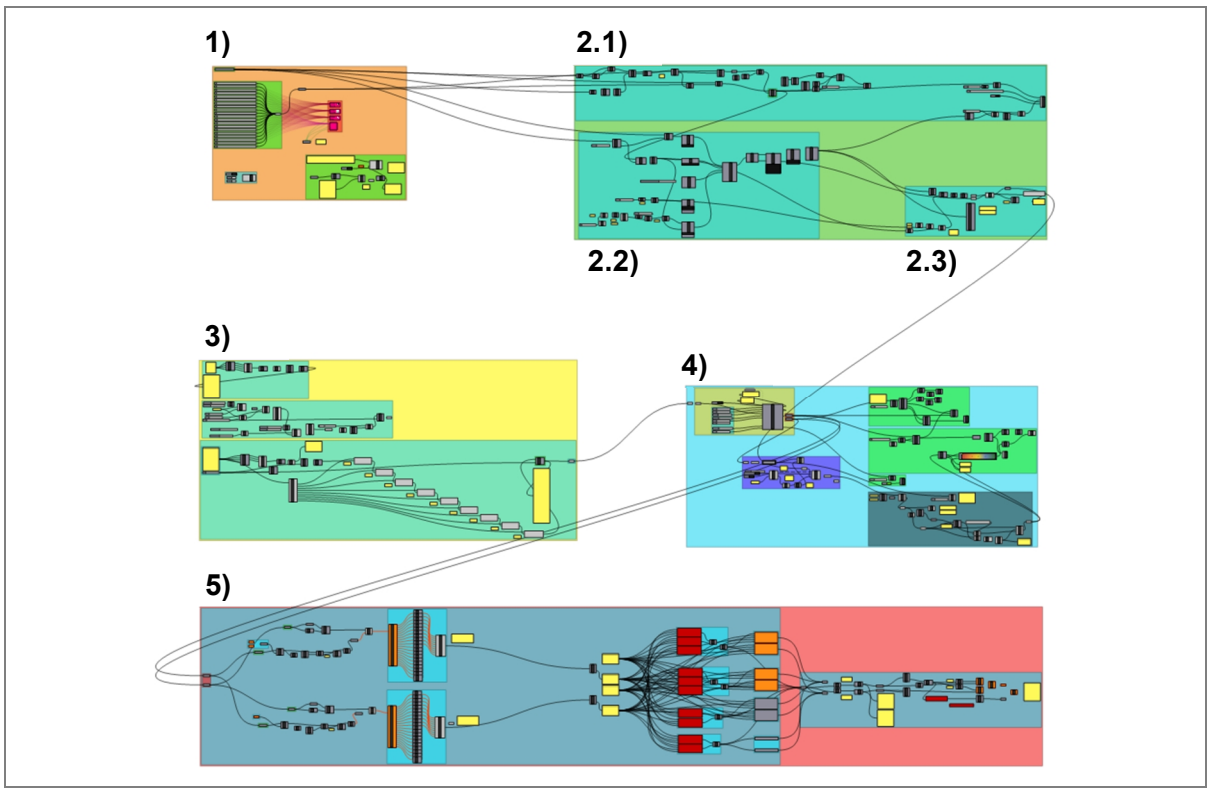
3) Generieren der Variablensets im Raster zur Visualisierung des Variablenraums

4) Visualisierung des Variablenraums (SOM)

5) Eingrenzung des Variablenraums durch die Auswahl guter und schlechter Bereiche des Variablenraums

Benchmark Test 3:

- Pfahlgründung: Anordnung der Pfähle unter den Pfahlkopfbalken



1) Variablen Input, Input der Gründungsgeometrie, Optimierung und Aufzeichnung der Optimierungsdaten

2) Spezieller Solver

2.1) Definition der Pfahlanordnung unter den Pfahlkopfbalken

2.2) Erstellen des Finite-Element-Modells

2.3) Berechnung des Fitnesswerts

3) Generieren der Variablensets im Raster zur Visualisierung des Variablenraums

4) Visualisierung des Variablenraums (SOM)

5) Eingrenzung des Variablenraums durch die Auswahl guter und schlechter Bereiche des Variablenraums

Anmerkung: Digitale Versionen der Grasshopper Definitionen sind in Form von .gh-Dateien auf der Daten-CD (Anlage C) zu finden.

Anlage B: Optimierungsdaten der Benchmark Tests

Die aufgezeichneten Optimierungsdaten aller Benchmark Tests sind ausschließlich in digitaler Form auf der Daten-CD zu finden.

Anlage C: Daten-CD

Enthaltene Dateien:

- Digitale Version der Masterarbeit
- Grasshopper Definitionen der Benchmark Tests inklusive der Definitionen zur Visualisierung von Fitnesslandschaften und Variablenräumen
- Optimierungsdaten der Benchmark Tests als .xlsx-Datei