hochschule mannheim

Entwurf und Anwendung von Verfahren zur Optimierung der Parameter in Moduro-Modellen

Karin Adler

Master-Thesis

zur Erlangung des akademischen Grades Master of Science (M.Sc.)

Studiengang Informatik

Fakultät für Informatik

Hochschule Mannheim

28.02.2017

Betreuer

Prof. Dr. Markus Gumbel, Hochschule Mannheim

Prof. Dr. Jörn Fischer, Hochschule Mannheim

, :
Entwurf und Anwendung von Verfahren zur Optimierung der Parameter in Moduro-Modellen / Karin Adler. –
Master-Thesis, Mannheim : Hochschule Mannheim, 2017. **??** Seiten.

, :
/ Karin Adler. –
Master-Thesis, Mannheim : University of Applied Sciences Mannheim, 2017. **??** pages.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mannheim, 28.02.2017

Karin Adler

Abstract

Inhalt dieser Masterthesis umfasst das Parameterhandling und die Optimierung von Urothelsimlationen durch ein automatisiertes Optimierungstool. Als Ausgangspunkt dient das Moduro-Projekt der Hochschule Mannheim in Kooperation der Urologie am Universitätsklinikum Mannheim der Universität Heidelberg. Die in Moduro erstellten CompuCell3D-Modelle dienen als Basis zur Optimierung durch ein Optimierungsframework. Die Ergebnisse liefern einen umfassenden Einblick in die Welt der Optimierungs- und biologischen Simulationsverfahren, sowie umfangreiche Erläuterungen zum aktuellen Stand und technischen Machbarkeit.

Inhaltsverzeichnis

Kapitel 1

Einleitung

„Das Optimum beschreibt den für die Organismenart günstigsten Wert, in dem auch Lebensvorgänge ihren höchsten Gewinn erreichen. Höhere oder niedrigere Werte bedeuten eine Verschlechterung der Lebensbedingungen für den Organismus.“ [4]

Das ist eine von vielen Definitionen, welche ein biologisches Optimum beschreiben. Diese Optima zu finden, ist oftmals keine Herausforderung. Diese erreicht die Natur schon selbst - über viele Generationen und über Millionen von Jahren hinweg. Doch wie so ein biologisches Präferendum entsteht und welche Faktoren dabei eine Rolle spielen, bleibt nicht selten unerforscht.

Um die Biologie von Lebewesen, Organen oder Zellverbänden gänzlich zu verstehen, reichen Beobachtungen, Sektionen oder mikrobiologische Untersuchungen nicht immer aus. Oftmals bleiben vielerlei Faktoren aus den unterschiedlichsten Gründen unentdeckt. Um diese Probleme zu umgehen, werden in der Biologie Simulationen eingesetzt, um biologische Prozesse abzubilden und nachzustellen. Doch selbst dafür wird ein gewisses Wissensspektrum über die Funktionalitäten und Eigenschaften eines zu modellierenden Körpers vorausgesetzt.

Dies trifft auf das Urothelgewebe, um welches es in dieser Arbeit geht, nicht ganz zu. Viele, aber nicht alle Eigenschaften des Urothels sind bereits erschlossen, was das Erstellen von Simulationen erschwert.

Ziel dieser Arbeit ist es nun, einige dieser fehlenden oder nicht ausreichend erforschten Eigenschaften, des weiteren als Parameter bezeichnet, mittels Optimierung zu ergründen, um die Simulationen des Urothelgewebes zu verbessern. Eine Verbesserung bedeutet eine Annäherung an das Optimum des Urothelgleichgewichts und damit das Erringen von Erkenntnissen über biologische Zustände und Prozesse im Gewebe, die wiederum zur Erkennung von Krankheiten oder zur Abstraktion auf eine andere Gewebe genutzt werden können.

Diese Arbeit stützt sich auf die von [26] erlangten Ergebnisse. Als Ausgangspunkt dienen Urothelmodelle, die bereits beschrieben und implementiert vorliegen und die es bestenfalls zu verbessern gilt.

Um dieses Vorhaben zu verwirklichen, wird zuerst nach einem Optimierungsverfahren gesucht, welches sich auf die gegebene Problemstellung anwenden lässt. Nach der Erörterung verschiedenster Optimierungsverfahren, wird nach der Auswahl vorher definierter Kriterien ein Verfahren ausgesucht und beschrieben. Die Moduro-Toolbox, welche Features zur statistischen Auswertung der Urothelmodelle liefert, wird anschließend mit einer Optimierungserweiterung ausgestattet. Diese Optimierungserweiterung ist in der Lage, die von den Simulationsmodellen generierten Dateien auszulesen, die wichtigen Parameter zu extrahieren, im Anschluss zu optimieren, alle Parameter wieder zusammenzuführen und als neue Datei zu speichern. Diese neu angelegten Dateien dienen nun als Grundlage für neue, bestenfalls verbesserte, Simulationsläufe jener Modelle. Eine Verbesserung tritt ein, wenn sich die Fitness eines Modells mit jedem optimierten Simulationslauf stetig verbessert.

Kapitel 2

Grundlagen

2.1 Biologie des Urothels

Die Harnblase ist ein von Schleimhaut ausgekleidetes sackartiges Organ mit umgebender, glatter Muskulatur und dehnbarer Wand. Zu lokalisieren ist die Blase direkt im kleinen Becken hinter dem Schambein [24]. Diese innere Schleimhaut, das Urothel, ist ein Übergangsepithel und trennt das Blasengewebe vom hochosmolaren Urin. Zu den Aufgaben des Urothels gehört neben der selektiven Permeabilität, die interzelluläre Kommunikation oder die physische Barriere als Schutzfunktion vor Giftstoffen und Krankheitserregern [16].

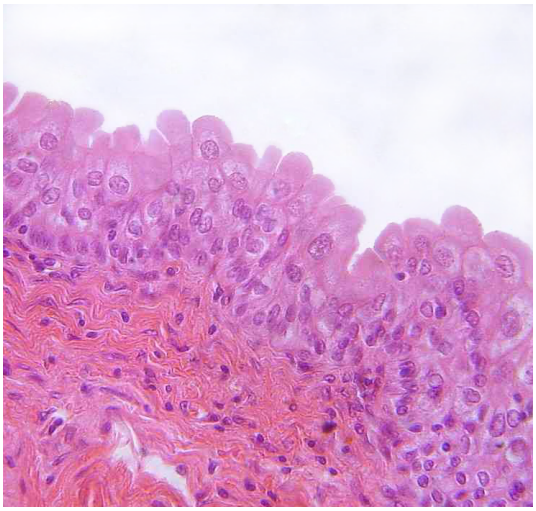


Abbildung 2.1: Gefärbtes Schnittbild des Urothels
Quelle: https://commons.wikimedia.org/wiki/File:Harnblase_Urothel.png

Es besteht ein hoher osmotischer, sowie chemischer Gradient zwischen dem Urin und dem umgebenden Gewebe, so dass das Urothel mehrere Mechanismen entwickelt hat, um den Eintritt von Urin in das Blaseninnere zu verhindern. Über der versiegelnden Deckschicht liegt zusätzlich die GAG-Schicht (Glykosaminoglykan), von der man eine antibakterielle Wirkung annimmt. Unter dem Urothel liegt die

Submukosa oder auch Basalmembran - ein dichtes Bindegewebe mit Gefäßplexus [19].

Das Urothel selbst ist mehrschichtig mit maximal sieben Zelllagen, welche in drei Schichten gegliedert werden können: Basalzellschicht, Zwischenzellschicht (Intermediärzellen) und Deckzellschicht (Umbrellazellen) [16], die als luminale Oberfläche dient [19]. Die Zellteilung findet in der Basalschicht statt. Die Basalmembran als unterste Schicht stützt das Gewebe und gibt den oberen Schichten den nötigen Halt. Zwischen den Basalzellen finden sich einzelne Stammzellen, die die Proliferation des Urothels vorantreiben. Teilt sich eine Stammzelle, erzeugt sie eine neue Basalzelle. Die Zellen wandern mit der Zeit luminal und differenzieren zu Intermediärzellen, sobald der Kontakt zur Basalmembran unterbricht. Die Intermediärzellschicht ist die dickste Schicht mit bis zu fünf übereinanderliegenden Zellen und macht den größten Anteil der gesamten Zellmasse aus. Durch das ständige Abtragen der obersten Zellschicht, gelangen neue Intermediärzellen an die Oberfläche und differenzieren durch die Verbindung mit dem Urin zu Umbrellazellen.

Wie beispielsweise auch die Haut oder der Verdauungstrakt ist ebenso die Harnblase gezwungen, sich vor reizenden Bestandteilen des Mediums oder toxischen Abfallprodukten zu schützen. Wie eben beschrieben, zeigt das ganze System der harnleitenden Organe auch hier wie ausgefeilt der anatomische Schutzmechanismus vor körperfremden Stoffen aufgebaut ist. Trotz all dieser biologischen Sicherheitsvorkehrungen, können einzelne Faktoren aus dem Gleichgewicht geraten. Die Deregulation der Homöostase durch die hohe Zellerneuerungsrate, sowie die reaktive Umgebung macht das Urothelgewebe anfällig für Mutationen und fehlerbehaftete Gewebeneubildungen: Über 90% aller Blasenkrebsbefunde sind Urothelkarzinome. Das gilt vor allem für Männer ab dem 40. Lebensjahr und gipfelt zwischen dem 60. und 70. Lebensjahr. Zudem sind sie fast doppelt so häufig betroffen wie Frauen. In der ganzen Europäischen Gemeinschaft zählt man etwa 104 000 neue Fälle und 36 400 Todesfälle pro Jahr und gilt somit als neunthäufigste Krebserkrankung für beide Geschlechter weltweit [7].

2.2 Simulationsmethoden

Bei einer Simulation werden Experimente an einem Simulationsmodell durchgeführt, die für eine gewöhnliche Experimentalserie zu komplex sind. Dieses Modell stellt eine Abstraktion der Realität in Form einer Struktur oder Funktion dar. Durch das Verändern der gegebenen Variablen lassen sich Unmengen an Szenarien durchspielen, die andernfalls aus Zeit- oder Kostengründen nicht durchführbar sind.

In der Biologie oder Medizin ist die Anwendung von Simulationen ein wichtiger Faktor zur Erforschung unzureichend untersuchter Problemstellungen geworden. Viele Zell- oder Gewebestrukturen lassen sich schwer oder gar nicht kultivieren. Andere haben zu lange Zellzykluszeiten, um natürliche Abläufe kontinuierlich zu

erforschen, sind zu teuer oder ethisch nicht vertretbar. In solchen Fällen macht man sich Simulationen zu Nutze, die die Realität durch rechnerisch entwickelte Modelle interpretierbar machen.

Es gibt viele Möglichkeiten eine Simulation zweckmäßig abzubilden, die wiederum unterteilt werden können. Grob werden folgende Simulationsmethoden unterschieden:

Statische Simulationen sind Simulationen, die nur einen bestimmten Zeitpunkt abbilden, also zeitliche Komponenten nicht berücksichtigen [10].

Anders bei den **dynamischen Simulationen**. Wie die Bezeichnung schon verrät, spielt hier die Zeit eine große Rolle. Diese Simulationen bilden in der Regel Prozesse und Abläufe ab. Aber auch die dynamischen Simulationen lassen sich in diskrete und kontinuierliche Simulationen unterteilen. Kontinuierliche Simulationen bilden stetige Prozesse ab, wohingegen diskrete die Zeit nutzen, um bestimmte Ereignisse hervorzurufen und damit Zustände bestimmen [10].

Außerdem gibt es noch die **Monte-Carlo-Simulationen**. Diese Art von Simulation nutzt stochastische Mittel, indem Zufallszahlen erzeugt werden, um Näherungen in einem System zu erreichen [17].

2.2.1 Glazier-Graner-Hogeweg und CC3D

Die letzten Jahre zeigten, dass es bereits sehr eindrucksvolle und auch realistische Simulationen einzelner Zellen oder kleiner Zellverbände und derer Eigenschaften als Reagenzglas-Ersatz gibt. Aspekte wie die Interaktion zwischen einzelnen Zelltypen, deren Dynamik oder Verhalten werden allerdings meist nicht berücksichtigt, was Ergebnisse auch nur kleiner Zellkultur-Simulationen durchaus verfälschen kann. Aus diesem Grund nutzt das unten vorgestellte Tool, CompuCell3D, das Glazier-Graner-Hogeweg-Verfahren, um Zellverbände authentisch simulieren zu lassen [25].

2.2.1.1 Glazier-Graner-Hogeweg (GGH)

Das GGH-Modell ist ein auf dem Cellular-Potts-Modell [17] basierender Algorithmus zum Simulieren von kollektivem Zellstrukturverhalten. Alle Objekte, in diesem Fall Zellen, sind Punkte auf einem Gitter, die als Gruppe ein Volumen oder eine Oberfläche ergeben. Jede Zelle hat einen Zustand σ , der über die Zugehörigkeit eines bestimmten Gewebetyps bestimmt. Zudem bekommt jeder Zustand eine Energie E zugeteilt. Da sich lebendes Gewebe bewegt und verändert, können sich auch die Gitterpunkte in der Simulation verändern.

Das System wird stets versuchen die Gesamtenergie zu minimieren. Eine Darstellung der Minimierungsfunktion sieht aus wie folgt:

$$E = \sum_{Zellen\, i} (\lambda_{Volume}(V_i - V_{0,i})^2 + \lambda_{Surface}(S_i - S_{0,i})^2) \tag{2.1}$$

V_i und S_i sind dabei Ist-Volumina und -oberflächen und λ gewichtet die gegebenen Zustände. Diese sind zu minimieren und werden deshalb mit den Sollwerten für Volumen und Oberfläche verglichen. Bei solch einem Schritt wird i zufällig gewählt. Zusätzlich wird ein Nachbarpunkt von i, in diesem Fall j untersucht und die temporäre Energie p übertragen. Ist die Differenz ΔE der neuen Nachbarschaft kleiner als die vorherige, wird der Zustand der Zelle getauscht. Ist $\Delta E < 0$ verringert sich durch den Austausch des Zustands die Gesamtenergie. Wäre $\Delta E > 0$, nimmt die Gesamtenergie des Systems zu und die Zustände sollten nicht gewechselt werden. Das Prinzip wird in Abbildung 2.2 veranschaulicht.

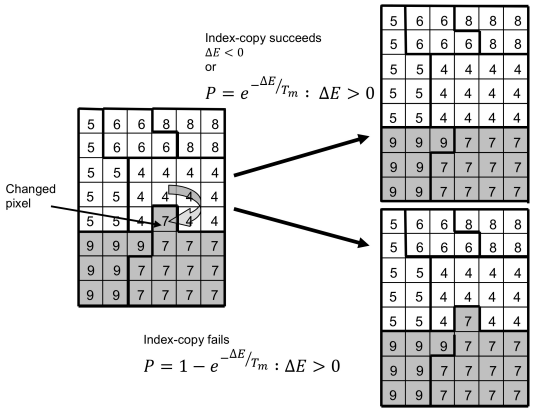


Abbildung 2.2: Darstellung des Glazier-Graner-Honeweg-Verfahrens
Quelle: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0033726#pone-0033726-g013

Alle GGH-Modelle beinhalten eine Liste von Objekten, eine Interaktionsbeschreibung, sowie geeignete Anfangsbedingungen [25].

2.2.1.2 CompuCell3D (CC3D)

CompuCell3D ist eine Open-Source Entwicklungsumgebung zum Erstellen und Ausführen von Simulation von Zellen, Zellverbänden, Geweben oder ganzen Organen.

Betrachtet man Zellen phänomenologisch, ungeachtet der unendlichen Reaktionsmöglichkeiten, Interaktionen oder diverser Genomverhalten, haben sie alle jene Eigenschaften beziehungsweise Fähigkeiten inne: Bewegung, Teilung, Apoptose, Differenzierung, Formveränderungsvermögen, Ausübung von Kräften, Absondern oder Absorption chemischer Substanzen, sowie elektrischer Ladungen und zuletzt

das Verändern der Volumen- oder Oberflächenstruktur. CC3D macht sich die Eigenschaften des GGH-Prinzips zu Nutze, indem die Parameter auf die Modelle übertragen werden. Ebenso werden Gitterdimensionen, Zelltypen, biologische Mechanismen und weitere Hilfs- und Nebenbedingungen in einem Python-Skript spezifiziert [25]. Wie all diese Faktoren zur Simulation ausgearbeitet und implementiert werden, ist in Abbildung 2.3 zu sehen.

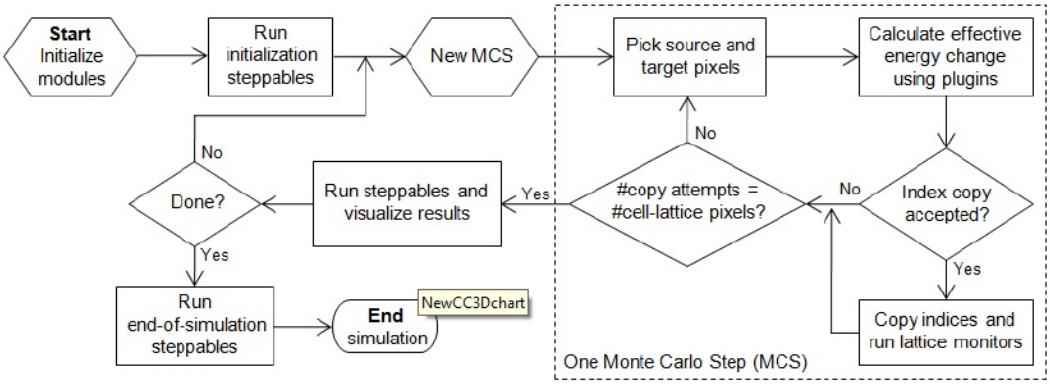


Abbildung 2.3: Flussdiagramm des GGH-Algorithmus in CC3D
Quelle: Introduction to CompuCell3D [25]

Primär bewachen die Python-Skripte Veränderungen in der Implementierung, einschließlich dem Zellverhalten oder dem Status der Simulation [25].

2.2.2 Optimierungsverfahren

Das folgende Kapitel soll einen kurzen und allgemeinen Überblick über den Bereich der Optimierung liefern, um Grundlagenwissen zum Verständnis dieser Arbeit zu gewährleisten.

Optimierungsverfahren sind gängige Methoden zum Lösen trivialer, aber auch komplexer, praktischer Systeme der angewandten Mathematik, die mit unbekannten Parametern arbeiten. Eine wesentliche Aufgabe ist es also, optimale Lösungen für jenen Parametersatz zu finden, der den maximalen Deckungsbeitrag liefert. Jedoch ist eine Modellierung fast immer mit einer Idealisierung verknüpft. Das Problem für das ein Modell oder eine mathematische Formel konstruiert wird, spiegelt nur eine Annäherung wieder und lässt Ergebnisse nur vorsichtig interpretieren [12].

In allgemeiner Form beschrieben, suche man also Lösungen der Optimierungsaufgabe

$$f(\vec{x}) \rightarrow max!$$

$$\vec{x} \in W,$$

wobei $W \in \mathbb{R}^n$ der sogenannte zulässige Bereich, abhängig von n Einflussparametern, und $f\colon W \rightarrow \mathbb{R}$ die stetige und differenzierbare oder undifferenzierbare Zielfunktion der Optimierungsaufgabe ist. Meistens ist der zulässige Bereich W zusätzlich durch Nebenbedingungen definiert [3]. Ist $W = \mathbb{R}^n$, so handelt es sich um ein unrestringiertes Optimierungsproblem. Ist die zulässige Menge W beschränkt, so spricht man von restringierten Problemen, die als Gleichungen oder Ungleichungen der Form: $x \in W \iff g_i(x) = 0\ (i = 1, ..., p)$ und $h_j(x) \leq 0\ (j, ..., q)$ dargestellt werden.

2.2.2.1 Begriffe

Um eine gegebene Aufgabenstellung zu optimieren, ist eine **Fitness-Funktion** vonnöten, die eine Abbildung der Realität darstellt. Diese Fitness-Funktion kann beispielsweise ein Abstand, eine Menge oder eine Bilanz sein. Je nach Szenario gilt es anhand gegebener **Parameter** oder **Variablen** diese Fitness-Funktion zu maximieren oder zu minimieren. Ab einer Parameterzahl $n \geq 2$ spricht man von einer **mehrdimensionalen** Optimierungsaufgabe. In solchen Fällen können die Parameter, sofern sie voneinander unabhängig sind, an den Achsen aufgespannt werden. Die Zielfunktion bildet so ein Gebirge mit Bergen und Tälern, wie in Abbildung 2.4 aufgeführt ist. Entsprechend ist das tiefste Tal das **Minimum** und der höchste Gipfel das **Maximum**.

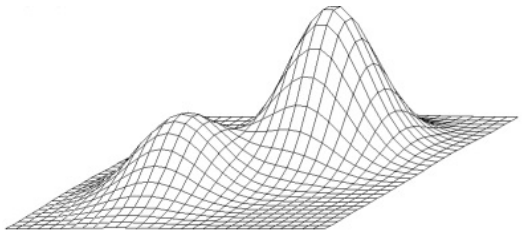


Abbildung 2.4: Schaubild einer zweidimensionalen Oberfläche
Quelle: https://commons.wikimedia.org/wiki/File:Local_maximum.png#/media/File:Local_maximum.png

Bei hochdimensionaler Optimierung wird diese Gebirgslandschaft sehr schnell unübersichtlich und zeichnet sich durch viele Maxima bzw. Minima aus. Manche Optimierungsaufgaben suchen, vom Ausgangspunkt gesehen, nur das nächste (lokale) Minimum oder Maximum, was als **lokale Optimierung** bezeichnet wird. Wird

die ganze Zielfunktionstopologie nach dem absoluten Hoch- oder Tiefpunkt durchsucht, spricht man von **globaler Optimierung**, die in der Regel einen höheren Rechenaufwand voraussetzt.

Häufig gelten für realistische Systeme sogenannte **Nebenbedingungen**. Diese Nebenbedingungen grenzen den zulässigen Wertebereich der Parameter ab und werden meist als Gleichungen oder Ungleichungen dargestellt. Die Menge aller Parameter, die die zugelassenen Werte erfüllen, nennt man **Paretomege** oder **zulässige Menge** und beschränkt die zu durchsuchende Fläche der Gebirgslandschaft [3].

2.2.2.2 Gliederung

Es gibt viele Wege Problemstellungen zu benennen oder zu katalogisieren - auch bei der Optimierung ist dies der Fall. Von der mathematischen Modellierung ausgehend, lassen sich Optimierungsverfahren in zwei Kategorien einteilen: Lineare und nichtlineare Optimierungsverfahren.

Bei **linearer Optimierung** ist die Fitness-Funktion stets linear und besitzt die allgemeine Form: $f(x) : W \rightarrow \mathbb{R}^1$. In der Regel werden weitere Fitness-Funktionen als Nebenbedingungen definiert, um eine Menge oder eine Fläche einzugrenzen. Eine Darstellungsform sei $f(\vec{a} + \vec{b}) = f(\vec{a}) + f(\vec{b})$ und $f(s \cdot \vec{a}) = s \cdot f(\vec{a})$.

Das heißt, es gibt nur ein (globales) Optimum und entsprechend beinhaltet die Lösungsmenge nur ein Element im Lösungsraum $\mathbb{R}^n$.

Die Herangehensweise **nichtlinearer Optimierungsverfahren** unterscheidet sich deutlich und behandelt Problemstellungen, deren Fitness-Funktion nicht linear ist. Für ein Optimierungsproblem kann es demnach auch mehrere Lösungen geben, aus denen man ein Optimum bestimmen kann. Nichtlineare Funktionen leiten sich in der Regel aus multikriteriellen Problemstellungen ab und sind allgemein folgendermaßen zu beschreiben: $f_i(x) : W \rightarrow \mathbb{R}\ (i = 1, ..., m)$.

Weitere Gliederungen sind möglich und in Schaubild 2.5 dargestellt. Zur Optimierung können also verschiedene Ansätze verwendet werden. Primär entscheiden aber Informationen über $f(x)$, sowie die Nebenbedingungen, welche Herangehensweise genutzt wird.

Deterministische Verfahren nutzen beliebig gesetzte Startpunkte auf der Gebirgslandschaft und berechnen die Werte anhand gegebener Eigenschaften. Die Auswertung dieser Lösungen legt die neuen Startpunkte fest. Dieses Verfahren wird iterativ solange wiederholt bis ein Abbruchkriterium erfüllt wird. Hierbei handelt es sich um White-Box-Verfahren, die Informationen über die Gestalt von $f(x)$ nutzen, um zur Orientierung Suchpunkte entlang der Gradienten zu setzen. Ist ein Punkt erreicht, an dem keine Verbesserung zum letzten Durchlauf vorliegt, ist der Gradient an diesem Punkt 0 und entspricht einem, in der Regel, lokalem Optimum, da

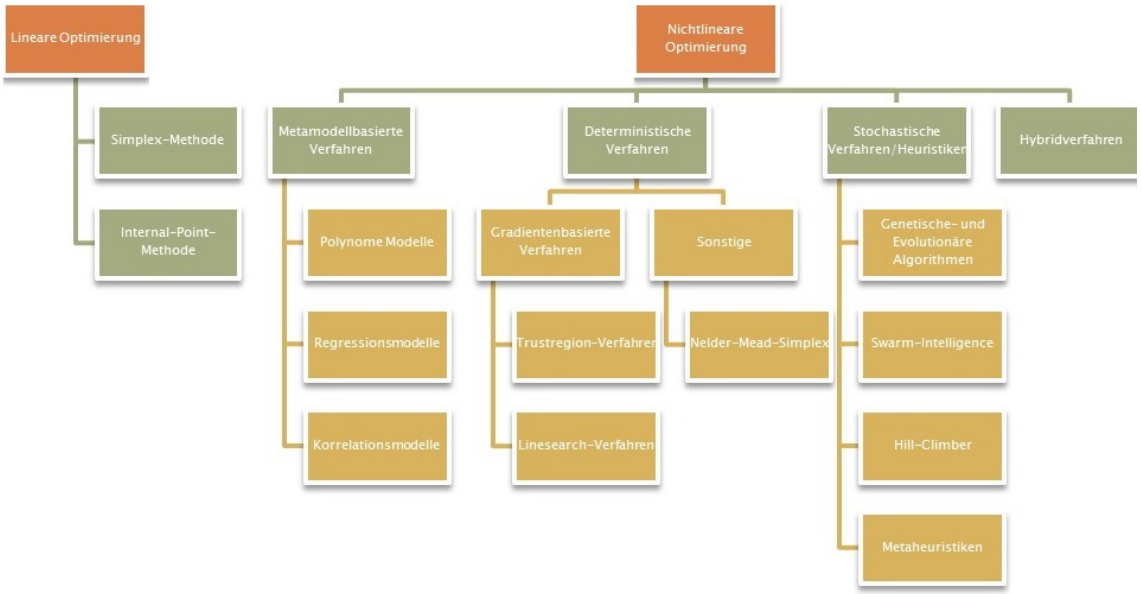


Abbildung 2.5: Hierarchie von Optimierungsmethoden

mit dem Berechnen der Gradienten nur die Nachbarschaft durchsucht werden kann [14].

Dahingegen werden bei Black-Box-Verfahren, die keinerlei Informationen über die Zielfunktion besitzen, Zufallspunkte als Startpunkte gewählt. In Abhängigkeit der Auswertung werden neue, verbesserte Suchpunkte gesetzt. Die Verbesserung der Suchpunkte erfolgt deterministisch oder stochastisch. Diese Handlungsweise praktizieren **stochastische Verfahren** und **Heuristiken**. Sie sind im Gegensatz zu deterministischen Verfahren weniger effizient, jedoch ist das Finden eines globalen Optimums durch die verteilte Anordnung der Startpunkte einfacher [14].

Auch beim Ansatz der **metamodellbasierten Verfahren** ist ein zumindest bruchstückhaftes Wissen über $f(x)$ notwendig. Hierbei wird die Funktion an einigen Punkten ausgewertet, mit deren Hilfe eine neue, trivialere Funktion $g(x)$ modelliert wird. Diese vereinfachte Funktion nennt man Metamodell. Metamodelle können Regressionsmodelle oder einfach nur Funktionen niedrigerer Polynome sein. Durch Metamodelle können Suchrichtungen ermittelt werden, mit denen man Optima approximiert [20].

Kapitel 3

Problemstellung

Die Aufgabe dieser Arbeit ist nun, aufbauend auf den gegebenen Mitteln, mittels Moduro-CC3D und der Moduro-Toolbox geeignete Verfahren zu entwickeln, Zellen des Urothels zu simulieren, zu optimieren und auszuwerten.

Wie in den Grundlagen beschrieben, ist der Aufbau und die Funktionalität des Urothels weitestgehend bekannt. Das Verständnis wie sich dieses Gewebe in vivo verhält - also wie es differenziert und proliferiert - ist weniger erforscht. Hierzu wurden Modelle (Hypothesen) entwickelt, die mögliche Szenarien der Dynamik und des Verhaltens des Urothels abbilden sollen. Um diese Modelle quantitativ vergleichen zu können, sind Metriken ausgearbeitet worden, die Werte wie Zellvolumen, Zellanordnung und die Gesamtfitness darstellen. Die Hypothesen erweisen sich allerdings als unzureichend und es ist bisher ist es noch nicht gelungen, ein dauerhaft intaktes Gewebe darzustellen. Das heißt, die Simulationsläufe ergeben nach unbestimmten Zeitintervallen pathologisches Gewebe, unrealistische Konstruktionen oder erreichen just nicht die gewünschten Werte der Zielfunktion.

Nun ist es daran herauszufinden, ob die zu untersuchenden Hypothesen gänzlich ungeeignet sind oder sich nur die Auswahl der Parameter als unvorteilhaft erweist. Gesucht wird demnach ein geeigneter Parametervektor, der die Zielfunktion maximiert.

Die Simulation von biologischem Gewebe hängt nun aber von vielen Faktoren und deren Wechselwirkungen untereinander ab.

Wie Zellen im Urothel proliferieren und differenzieren, kann weitestgehend eingegrenzt werden. Jedes Proliferations- und Differenzierschema kann mit den gegebenen Zelltypen kombiniert werden. Demnach ergibt sich kombinatorisch eine überschaubare Anzahl an Modellen, die auf ihre Fitness untersucht werden müssen.

Jedes Modell selbst verfügt zusätzlich noch über eine Anzahl an Faktoren, die im Kontext dieser Arbeit als Parameter bezeichnet werden. Die Anzahl der Parameter innerhalb eines Modells beläuft sich momentan auf 129. Davon sind etwa die Hälfte biologisch relevante Faktoren. Einige von ihnen sind erforscht und dokumentiert, das heißt, sie obliegen einem festen Wert oder einem gewissen Messbereich und

dürfen diesen nicht über- oder unterschreiten. Ein Beispiel dafür wäre die Größe eines bestimmten Zelltyps, die sich recht einfach unter dem Mikroskop nachweisen lässt. Um unter realistischen Bedingungen zu simulieren, bekommen solche Faktoren, oder eben Parameter, einen unabänderlichen Default-Wert zugewiesen. Andere Faktoren sind nicht bekannt oder können nur schwer belegt werden. Vorbildlich dafür ist die Apoptose-Wahrscheinlichkeit eines Zelltyps oder die Wachstumsrate. Diese Faktoren sind in jedem Szenario variabel. Momentan befinden in jedem Modell unter den biologischen Faktoren etwa 20 Parameter, deren Einfluss auf das biologische System unbekannt ist.

Wie all diese Parameter, definiert als $\varphi_1, \ldots, \varphi_n$, interagieren und berechnet werden, ist für den Benutzer nicht nachzuvollziehen - zu sehen ist allein der Output als Funktionswert, der mit Funktionsargumenten versehen, eine unbekannte Funktion, des weiteren als **UroFunktion** bezeichnet, $f(x) = (\varphi_1, \ldots, \varphi_n, t)$ zu einem Zeitpunkt t durch eine Simulation darstellt.

Zusammengefasst wird schnell klar, dass ein Problem von diesem Ausmaß nicht trivial zu lösen ist und ein maximaler Fitnesswert weder durch das systematische Austauschen von Parametern, noch auf analytischem Wege erreicht werden kann.

Verwendet werden hierfür Optimierungsmethoden, welche etabliert und bei Tauglichkeit zur Optimierung der Hypothesen und den dazugehörenden Parametern herangezogen werden können. Problemstellungen mit ähnlichen Ausgangsbedingungen werden hierfür als Referenz herangezogen. Im Anschluss gilt es zu überprüfen, in wie weit sich die berufenen Optimierungsverfahren eignen und ob sich ein Mehrwert quantitativ abschätzen lässt. Dazu gehören zusätzliche Aspekte als eine Reihe von Nebenbedingungen, wie Aufwandsabschätzungen über Laufzeit, Sampling einzelner Parameter, die Menge an Iterationen pro Durchlauf, ebenso wie eine Anzahl an Abbruchbedingungen, genutzt als natürliche oder technische Restriktionen.

Die Realisierung soll durch die Erweiterung der Moduro-Toolbox mit einem Optimierungsmodul erfolgen. Dieses Optimierungsmodul dient als Blackbox, die möglichst nur die reine Optimierung der Parameter übernimmt. Der Output dieser Optimierung gilt somit als Ausgangspunkt für weitere Iterationen und Verbesserungsläufe für das ausgewählte Modell und zur weiteren Berechnung in Moduro-CC3D. Verfeinert wird die Funktion durch die Demonstration durch Schaubilder und Optimierungskurven.

Das Ziel dieser Arbeit ist demnach das Design und die Implementierung zur Erweiterung mit eben vorgestelltem Optimierungs-Feature für alle aktuellen und künftigen Modellierungshypothesen zur Gestaltung repräsentativer Fitness-Kurven inklusive benutzerfreundlichem Workflow für die Eingabe gewünschter Parameter mit Hilfe einer gekapselten, generischen Schnittstelle.

Zweck ist das Verständnis über biologische Prozesse zu vertiefen, was nicht schrittweise durch Beobachtungen und Forschung zustande kommen kann, sondern indem vorhandenes Wissen explorativ genutzt wird, reflexiv intakte Gewebestrukturen zu schaffen.

Kapitel 4

Stand der Dinge

4.1 Darstellung des Urothels als Simulation

Wenn man von Simulationen spricht, ist in der Regel eine Abbildung der Wirklichkeit gemeint, die möglichst realitätsgetreu agiert. Je mehr Eigenschaften einer Vorlage die Simulation annimmt, desto realistischer wirkt sie für den Benutzer und liefert entsprechend authentischere Ergebnisse. Doch was bedeutet realitätsgetreu in diesem Zusammenhang? Fakt ist, dass eine Simulation eine Nachahmung ist und einen lebenden Organismus nie vollends kopieren kann. Man kann allerdings versuchen, alle Eigenschaften eines Systems nachzustellen.

Abbildung 4.1: Zelllinie des Kontaktmodells (Urothel)
Quelle: [26]

Dafür muss man sich zuerst mit der Dynamik eines Systems befassen. Im Falles des Urothels stellt sich die Frage nach den Bestandteilen: Welche Zelltypen sind vorhanden? Welche Funktionen nehmen die einzelnen Zelltypen an und welche Eigenschaften besitzen sie? Im Anschluss werden größere Zellverbände betrachtet, das Verhalten der Zellen untereinander, sowie ganze Gewebe und alle Gesichtspunkte, die es ausmacht.

Abbildung 4.1 zeigt den Lebenszyklus einer Urothelzelle, im Genaueren den Proliferationszylus und an welche Regel sich eine Urothelsimulation halten muss.

4.2 Moduro

„ModUro“ (Modellierung und Simulation in der Urologie) ist ein Projekt des Instituts für Medizinische Informatik der Fakultät für Informatik der Hochschule Mannheim in Kooperation mit der Medizinischen Fakultät Mannheim der Universität Heidelberg. Moduro ist ein Set an Modulen, die zusammen das Simulieren des Urothelgewebes mit Hilfe von Modellen ermöglichen. Das realistische Abbilden der Zellen und deren Wechselbeziehungen im lebenden Organismus, soll durch das möglichst authentische Setzen biologischer Parameter erreicht werden. Ziel ist es, das Wissen über alle interagierenden Faktoren, die für das Gleichgewicht innerhalb des Gewebes von Bedeutung sind, zu vergrößern und somit Aufschlüsse über Anatomie und Verhalten des Urothels zu erlangen.

4.2.1 Moduro-CC3D

Die Umgebung für eine Urothel-Simulation entspricht einem Ausschnitt von $800\mu m$ Breite und $150\mu m$ Höhe. Die Dauer beträgt 720 simulierte Tage. Erreicht das Modell diesen Zeitpunkt, wird die Simulation beendet. Beginnend mit den initialen Stammzellen zum Zeitpunkt $t = 0$, wird verletztes Gewebe nachgestellt, das nur aus den eben genannten Stammzellen besteht. Im Laufe der Simulation heilt das Urothel und erreicht, je nach Modell, nach etwa 100 Tagen den **Steady-State**. Dieser Status beschreibt den gewöhnlichen Fluss des Urothelgewebes, siehe Abbildung 4.2.

Abbildung 4.2: Urothelsimulationen: Initialzustand (links) und im Steady-State (rechts)

Folgende biologische Prozesse müssen ebenso nachgeahmt werden [26]:

Urinieren

Alle sechs Stunden wird eine Harnabgang simuliert, indem 2% der obersten Zellschichten abgetragen und ausgespült werden.

Zellentstehung

Eine Tochterzelle entsteht, wenn sich eine Elternzelle teilt. Die Mitose-Phase wird eingeleitet, wenn eine Zelle ihre maximal initiierte Größe erreicht und impliziert dadurch gleichzeitig einen Zellzyklus.

Zelltod

Eine Zelle kann entweder durch Apoptose oder durch Harnentleerung absterben. Die Apoptose hat im Simulationskontext allerdings nicht die gleiche Bedeutung wie im lebenden Gewebe: Eine echte Apoptose ereilt nur die Umbrellazellen. Nach Ablauf ihrer Lebenszeit werden diese ausgewaschen oder anderweitig abgebaut. Basal- und Intermediärzellen sterben nach abgelaufener Lebensdauer nicht direkt ab, sondern simulieren, wie oben erwähnt eine Mitose und beginnen einen neuen Zyklus als differenzierter Zelltyp. Jeder Zelltyp besitzt eine andere Lebenserwartung, die zusätzlich um einen Mittelwert randomisiert wird.

Differenzierung/Proliferation

Erreicht eine Zelle ihren Lebensabend, so kann sie sich, je nach Modelldefinition, entweder teilen und ihren Zyklus neu beginnen oder zu einem anderen Zelltyp differenzieren. Ob eine Zelle proliferiert oder differenziert, hängt von einigen Parametern ab, wie beispielsweise der Zeit, der Position im Gewebe oder anderen äußeren Einflüssen wie der Berührung mit dem Medium. Die Abbildungen 4.2 bis 4.5 zeigen, welche Proliferations- und Differenzierungsschemata denkbar sind.

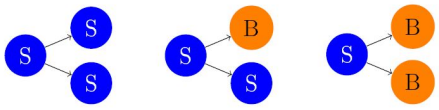


Abbildung 4.3: Zelllinien-Modellage der Stammzelle

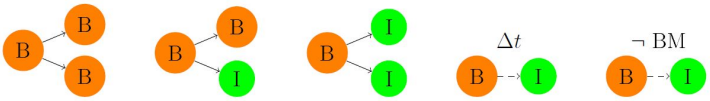


Abbildung 4.4: Zelllinien-Modellage der Basalzelle

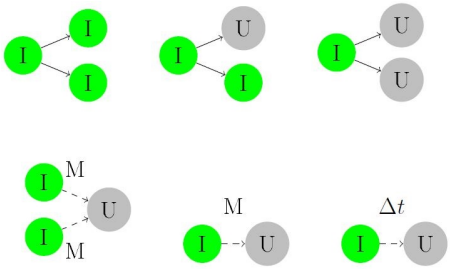


Abbildung 4.5: Zelllinien-Modellage der Intermediärzelle
Quelle: [26]

Um alle Zelllinien abzudecken, sind die Eigenschaften kombinatorisch verknüpft worden, was eine Menge an Urothel-Modellen hervorbringt. Die Einteilung und Beschreibung der Modelle folgt in Tabelle 4.2.

Zellsortierung

Oberflächenspannungen sind Zelleigenschaften, die Bewegung und Sortierung beeinflussen. Durch das Setzen unterschiedlicher Oberflächenspannungen zwischen den Zellverbänden können Anordnungen konstruiert werden, um die Gewebedynamik nachzustellen. Tabelle 4.1 listet die möglichen Kombinationen zwischen den einzelnen Zelltypen, der Basalmembran und dem Medium und zeigt 21 verschiedene Oberflächenspannungen für den Gewebekomplex.

Zelltypen		M	BM	S	B	I	U
Medium	M	0	14	14	14	14	4
Basalmembran	BM		-1	1	3	12	12
Stammzelle	S			6	4	8	14
Basalzelle	B				5	8	12
Intermediärzelle	I					6	4
Umbrellazelle	U						2

Tabelle 4.1: Oberflächenspannungen für die einzelnen Zelltypen, sowie Membran und Medium

Moduro-CC3D setzt zusätzlich für jedes Modell einen Satz an Parametern, die nicht direkt durch CC3D realisierbar sind oder für die von CC3D benötigten Funktionalitäten von Nöten sind. Die beschriebenen Parameter definieren die Struktur eines Modells, sind je nach Definition variabel, und bleiben für die Zeit einer Modellsimulation konstant [26].

Ein kurzer Überblick über die Simulationsstruktur gibt Aufschluss über Aufbau und Vorgehensweise von Moduro-CC3D:

Jedes Modell setzt in der eigens implementierten Klasse alle Parameter fest. Bei denen handelt es sich jedoch streng genommen um Python-Skripte, die als Wrapper für die C++-Umgebung von CC3D fungieren. Die Struktur für die Modelle gibt die Klasse **ModelConfig** vor. Dazu gehören unter anderem Parameter, die nicht modellspezifisch sind, sondern für die ganze Simulationsumgebung gelten, wie beispielsweise die Adhäsionsenergie oder der Adhäsionsfaktor. Darüber hinaus wird jedem Modell ein Array an Zelltypen übergeben, welche in der Klasse **CellType** initiiert werden. Jeder Zelltyp wird als Enum dargestellt und besitzt charakteristische Eigenschaften, wie Größe, Durchmesser oder Sterbewahrscheinlichkeit. All diese Eckpunkte fließen in die Modelle ein, bevor sie ausgeführt werden und ergeben ein fertiges Simulationskonstrukt.

Moduro-CC3D ist ein Open-Source-Projekt und unter GitHub einsehbar.

4.2.2 Urothel-Modelle

Aus der Proliferations- und Differenzierungstabelle, siehe Tabelle 4.2, gehen acht unterschiedliche Regeln hervor, aus denen momentan 16 Kombinationen resultieren. Jede Kombination entspricht einem Modell und ist autonom und separat implementiert, verfügt aber über den gleichen Parametersatz. Die Namen der Modelle setzen sich aus den IDs der Teilungsoptionen zusammen; zum Beispiel **SdSdbCdi**. Das bedeutet, dass sich das Modell aus einer symmetrischen Stammzellenteilung, einer symmetrischen Basalzellenteilung und schließlich aus der Kontaktdifferenzierung der Intermediärzellen zusammensetzt.

ID		Beschreibung	Model
Stammzelle	SD	Symmetrische Stammzellenteilung	$S \rightarrow S + S$ und $S \rightarrow S + B$
	SPA	Stammzellenpopulation erfolgt asymmetrisch	$S \rightarrow \begin{cases} S + S & p = P_{SSS} \\ S + B & p = P_{SSB} \\ B + B & p = P_{SBB} \end{cases}$
Basalzelle	SDB	Symmetrische Basalzellenteilung	$B \rightarrow B + I$
	BPA	Basalzellenpopulation erfolgt asymmetrisch	$B \rightarrow \begin{cases} B + B & p = P_{BBB} \\ B + I & P = P_{BBI} \\ I + I & p = P_{BBI} \end{cases}$
	PCDB	Proliferations- und Kontaktdifferenzierung der Basalzellen	$B \rightarrow B + B$ und $B \overset{\neg BM}{\rightarrow} I$
	CDB	Kontaktdifferenzierung der Basalzellen	$B \overset{\neg BM}{\rightarrow} I$
Intermediärzelle	PCDI	Proliferations- und Kontaktdifferenzierung der Intermediärzellen	$I \rightarrow I + I$ und $I \overset{M}{\rightarrow} U$
	CDI	Kontaktdifferenzierung der Intermediärzellen	$I \overset{M}{\rightarrow} U$

Tabelle 4.2: Mögliche Proliferations- und Differenzierungsregeln
Quelle: interne Kommunikation, [6]

Ist eine Zellpopulation asymmetrisch, wird für jeden eintretenden Fall eine Teilungs- oder Differenzierungswahrscheinlichkeit angegeben. Folglich würde eine Stammzelle mit asymmetrischer Zellteilung und einer Fallunterscheidung von

$$S \rightarrow \begin{cases} S + S & p = 5\% \\ S + B & p = 90\% \\ B + B & p = 5\% \end{cases}$$

bei einer (simulierten) Teilung zu 90% eine Stammzelle bleiben und eine weitere Basalzelle erzeugen, mit einer 5% - Wahrscheinlichkeit eine weitere Stammzelle zeugen oder eine Basalzelle zeugen und sich selbst in eine Basalzelle umwandeln.

4.2.3 Fitness-Funktion

Nach all den Variablen bemessen, die es zum Simulieren des Urothels gibt, muss festgelegt werden, welche Güte ein Modell überhaupt aufweist, um quantitative Vergleiche zu stellen. Hierfür reicht die visuelle Auswertung nicht aus, da es keinen messbaren Richtwert oder Anhaltspunkt gibt.

Dazu entwickelten [26] eine Fitness-Funktion, mit der sich die Fitness eines Simulationslaufs und der generierten Daten bewerten lässt. Sie setzt sich gleichermaßen zusammen aus der Qualität der Zellanordnung, sowie dem vorhandenem Gewebevolumen zu einem Zeitpunkt t_i. Zusammengesetzt erfolgt die Bewertung der Fitness der Funktion:

$$f = \frac{\bar{f}_a + f_v}{2} \tag{4.1}$$

wobei a die Anordnung (Arrangement) und v das Volumen symbolisiert. Die Fitnesswerte kongruieren mit $f(x_i)$ und werden zweimal pro Tag für 720 Tage gemessen und obendrein dokumentiert.

Fitnessarrangement

Die Fitnessfunktion f_a bewertet im Genaueren die Verteilung der Zellschichten. Ob jede Schicht vorhanden und messbar ist und ob die Reihenfolge der Schichten konstant korrekt bleibt. Um die Reihenfolge zu ermitteln werden vertikale Stichproben entnommen, aus denen wiederum kleine Voxel alle $7\mu m$ abgelesen werden. Die häufigste Zellprobe wird ermittelt und auf Richtigkeit geprüft. Eine maximale Fitness von 1 ergibt sich, wie unten ausführlicher beschrieben, wenn auf der Basalmembran alle Stamm- und Basalzellen liegen, auf diesen sich wiederum die Intermediärzellen aufbauen und schließlich die luminale Schicht aus Umbrellazellen besteht. Zusammengefasst ergibt sich daraus folgender Term:

$$f_a = \begin{cases} \frac{1}{(1-L_B)+(L-L_I)+(1-L_U)+1} & if\ L_B + L + L_U > 0 \\ 0 & else \end{cases} \tag{4.2}$$

Dabei handelt es sich bei $(1-L_B)+(L-L_I)+(1-L_U)$ um die Anzahl der Zellen, die sich von ihrem Bestimmungsort wegbewegen, wie beschrieben:

- $L_B = 1$, wenn die erste Schicht ausschließlich aus Stamm- oder Basalzellen besteht. Der Alternativfall ergibt 0.
- $L_U = 1$, wenn die oberste Schicht ausschließlich aus Umbrellazellen besteht. Der Alternativfall ergibt 0.
- L ist die Anzahl der Zellschichten zwischen den äußersten Schichten, wobei L_I eine Untermenge von L darstellt und aus Intermediärzellen besteht.

Die Summe der gemittelten Stichproben ergibt

$$\bar{f}_a = \frac{1}{n} \sum_{i=1}^{n} f_a(i)\,. \tag{4.3}$$

Fitnessvolumen

Wie groß eine Urothelzelle zu sein hat, ist nicht genaustens beschrieben. Aus der Literatur [23; 8] ist jedoch eine Gesamthöhe des Urothels zu entnehmen, die sich auf etwa $85\mu m$ beläuft. Den Simulationsbildausschnitt betrachtend, ergibt sich ein optimales Sollvolumen V_{opt} also aus $150\mu m \cdot 85\mu m \cdot 1\mu m$. Davon belegt ca. 23% des Volumens die Menge an Umbrellazellen, 67% machen Intermediärzellen aus und die restlichen 10% sind Stamm- und Basalzellen. Daraus ergibt sich die optimale Verteilung V_{opt} aus den Werten $S_B = 0,1 \cdot V_{opt}$, $S_I = 0,67 \cdot V_{opt}$ und $S_U = 0,23 \cdot V_{opt}$. Die tatsächliche Volumenokkupation zu einem Zeitpunkt t_i ist definiert als I_B, I_I und I_U. Ergo ergeben sich daraus drei Fitnessfunktionen, definiert als:

$$f_B = \frac{1}{\frac{(S_B - I_B)^2}{a \cdot V_{max}} + 1}\,,\ f_I = \frac{1}{\frac{(S_I - I_I)^2}{a \cdot V_{max}} + 1}\,,\ f_U = \frac{1}{\frac{(S_U - I_U)^2}{a \cdot V_{max}} + 1}\,, \tag{4.4}$$

mit $a = 3\mu m$ als regulierendem Faktor zur Beobachtung der Breite der Fitnessfunktion und einem maximalen Volumen $V_{max} = 3 \cdot 10^4 \mu m^3$, die zusammengefasst eine Volumenfitness von

$$f_v = \begin{cases} \frac{(f_B + f_I + f_U)}{3} & if\ 0 < I_B,\ I_I\ or\ I_U < V_{max} \\ 0 & else \end{cases} \tag{4.5}$$

festlegen.

4.2.4 Moduro-Toolbox

Die Moduro-Toolbox ist ein Software-Tool, das die wesentlichen Bestandteile zur statistischen Analyse von Moduro-CC3D-Modellen bereithält. Die Auswertung geschieht über die graphische Oberfläche der Moduro-Toolbox: alle bereits abgeschlossenen Simulationen werden aus dem verwiesenen Dateisystem gelesen und anschaulich gelistet. Durch Auswahl eines Modells ist es möglich einzelne Simulationen, aber auch gruppierte Simulationsläufe anzuzeigen. Box-Whisker-Plots und Liniendiagramme vereinfachen eine statistische Auswertung je nach Anwendungsfall und Vorlieben. Die Auswertung umfasst die Daten, die von Moduro-CC3D ausgegeben werden.

Um ein Modell anschaulich zu analysieren, muss erst definiert werden, welche Kriterien überhaupt untersucht und wie sie verarbeitet werden sollen. Zu diesem

Zweck wird die Fitness-Funktion als Referenz verwendet und anhand dieser alle Daten ausgewertet. Die Moduro-Toolbox beinhaltet sowohl die Veranschaulichung der Fitness-Funktion selbst, als auch die einzelnen Fitness-Funktion-Komponenten wie Volumen und Anordnung des simulierten Gewebes [21].

Erweitert durch das Optimierungs-Feature könnte die Moduro-Toolbox auch den Anstieg der Fitness im Verlauf eines Optimierungsprozesses kartieren.

Die Moduro-Toolbox ist ein Open-Source-Projekt und unter GitHub einsehbar.

4.3 Selektion der Optimierungsverfahren

Die Suche nach geeigneten Optimierungsalgorithmen gestaltet sich nicht selten als schwierig. Am sinnvollsten ist hierbei eine Vorab-Analyse der gegebenen Problematik und des gesuchten Ziels. Daran gemessen, können nun verschiedene Optimierungsmethoden abgewogen und auf Tauglichkeit getestet werden.

Aufgabenstellung: Maximieren einer Fitnessfunktion für einen kontinuierlichen Definitionsbereich durch das Abbilden eines Funktionsurprungs auf einen Funktionswert. Beispielhaft hierfür ist die Kurvenoptimierung oder, wie in gegebenem Fall, eine Modellkalibrierung, wie sie häufig in einem physikalischem oder biologischem Kontext zu finden ist:

$$f : \chi \subseteq \mathbb{R}^n \to \mathbb{R}$$

Ziel: eine möglichst schnelle Annäherung an ein globales Maximum für den Parametersatz eines definierten Modells, was in einem möglichst hohen und robusten Fitnesswert $f(x)$ resultiert.

Nach der Definition der Aufgabenstellung und des Ziels ist nun wichtig ein geeignetes Optimierungsverfahren zu finden. Die Eigenschaften der gesuchten Funktion sind hierfür essenziell für die Auswahl eines geeigneten Verfahrens. Um einen genauen Überblick zu schaffen, mit welchen Funktionseigenschaften die Suche konfrontiert wird, zeigt folgende Aufzählung sämtliche Eigenschaften der Funktion mit entsprechender Begründung.

Grundlegend ist festzuhalten, dass die gesuchte Formel $f(x)$ mit einfachen Mitteln nicht zu erörtern ist. Die Berechnung der Funktion geschieht über Moduro-CC3D und ist vorerst zu komplex zur Definition.

Daraus resultiert ein **Black-Box-Szenario** wie folgt:

Abbildung 4.6: Funktionsprinzip eines Black-Box-Szenarios
Quelle: [9]

Das Problem erfordert domänenspezifisches Wissen und kann nur innerhalb der Black-Box erfolgen. Demnach ergeben sich folgende Herausforderungen [9]:

Linearität

Ausgehend von der Tatsache, dass $f(x)$ weitgehend unbekannt und von komplexer Struktur ist, ist das Vorhandensein einer linearen oder konvexen Funktion nicht auszuschließen, aufgrund der Parameterzahl jedoch unwahrscheinlich. Es sollte eine Methode gewählt werden, die universell, und auch für nicht-lineare Funktionen, einsetzbar ist.

Gradienten

Da es sich um Blackboxing handelt, sind Gradienten nicht auffindbar und somit für die Optimierung nicht von Bedeutung.

Multimodalität

Zu erwarten ist, dass bei der Berechnung von $f(x)$ viele lokale Minima und Maxima zu finden sind. Je rauer bzw. zerklüfteter eine Funktionslandschaft ist, desto aufwändiger gestaltet sich die Suche nach einem Optimum. Infrage kommen demnach nur Optimierungsverfahren, die eine globale Suche abdecken.

Räumlichkeit

Eine er wenigen bekannten Faktoren ist die Anzahl der festen, sowie variablen Parameter, die bei der Lösung der Funktion eine Rolle spielen. Die Variation dieser Parameter ermöglicht erst das Optimieren der Funktion in einem bestimmten Umfang. Der Rahmen der variablen Parameteranzahl bewegt sich zwischen 15 und 20 und bedarf einem Optimierungsverfahren, welches diese Eigenschaften zu adäquater Zeit mit einem gewissen Maß an Genauigkeit handeln kann.

Trennbarkeit

Eine Funktion heißt trennbar (separabel), wenn

$$arg\ min\ f(\varphi_1, \ldots, \varphi_n) = (arg\ min\ f(\varphi_1, \ldots), arg\ min\ f(\ldots, \varphi_n))$$

gilt. Erst das Sampeln der einzelnen variablen Parameter gibt einen Einblicks über die Wechselwirkung unter den Parametern selbst. Eine Betrachtung von φ eindimensionalen unabhängigen Problemen ist wesentlich einfacher, in diesem Szenario jedoch nicht nachweisbar.

Alle oben genannten Eigenschaften gestalten die Optimierungsumgebung als recht schwierig und schränken die möglichen Optimierungsmethoden stark ein. Alle gelisteten Kriterien betrachtend, ist es trotzdem auf Anhieb möglich eine Vielzahl an Optimierungsverfahren auszuschließen. Aufgrund der vielen initialen Parameter und der eingeschränkten Nutzbarkeit, verbleiben alle **linearen Optimierungsmethoden** prinzipiell ungeachtet.

Wie oben bereits beschreiben, gibt es unzureichend Informationen über Form und Verhalten der Funktion. Die Approximation über Ableitungen ist in diesem Fall nicht möglich und alle **gradientenbasierten Verfahren** unter den nichtlinearen Optimierungsverfahren fallen infolgedessen raus. Darunter fallen auch alle Methoden, die zur Verwendung der Suche nach lokalen Optima genutzt werden. Diese agieren zumeist ebenfalls gradientenbasiert. Ein weiteres Ausschlusskriterium ist die Präferenz eines globalen Optimums.

Nach Erkundung der etwaigen Anzahl der variablen, zu optimierenden Parameter, eignen sich klassische Methoden wie **Nelder-Mead** [2] ebenfalls nicht für diese Problemstellung. Genutzt werden diese in der Regel für Funktionen mit einer Dimensionszahl von $n \ll 10$.

Modellbasierte Methoden [3; 15] sind bei Funktionen mit absehbar geringen Laufzeiten gebräuchlich, bei denen die Zahl der Iterationen selten zweistellig wird. Iterations- und Evaluationszahl sind in gegebenem Szenario nicht abschätzbar und damit ergeben sich modellbasierte Verfahren als ungeeignet [9].

Angesichts aller genannten Ausschlusskriterien ist bewiesen, dass genetische- und evolutionäre Algorithmen den Anforderungen entsprechen könnten und stellen mögliche Kandidaten zur Optimierung von Urothelsimulationen dar.

4.3.1 Optimierungsverfahren: Evolutionsstrategien

Die Problemstellung betrachtend, empfiehlt es sich nun auf einen Evolutionären Algorithmus (EA) zurückzugreifen. Diese werden in vier Hauptströme gegliedert, die jedoch ineinander übergehen können und keiner strikten Abgrenzungen unterliegen:

- Genetische Algorithmen
- Evolutionsstrategien
- Genetische Programmierung
- Evolutionäre Programmierung

Evolutionsstrategien liefern anwendbare Problemrepräsentationen mit identischem Suchraum basierend auf der nach Charles Darwin aufgestellten Theorie des *„Survival of the fittest“*. Inspiriert von der Natur lassen sich grundlegende Schemata auch auf Optimierungsprobleme übertragen. Beginnend mit der natürlichen Umgebung, kann in einem bestimmten **Lebensraum** - abhängig von Ressourcenknappheit, räumlicher Ausdehnung oder Fressfeinden - nur eine gewisse Anzahl an **Individuen** (über-)leben. Je besser also ein Individuum an seine natürliche Umgebung angepasst ist, desto wahrscheinlicher ist seine Überlebenschance und das Zeugen von Nachkommen, genannt **Selektion**.

Um den Genpool einer **Population** immer weiter zu verbessern, hat sich in der Natur die DNA-**Rekombination** zweier gesunder und gut angepasster Individuen als sehr nützlich bewährt. Zufällige **Mutationen** in der DNA neu gezeugter Nachkommen ermöglichen zusätzlich neue Evolutionspfade, was sich als ungemeiner Vorteil oder auch gravierender Nachteil gegenüber anderen Konkurrenten erweisen kann.

Daran angelehnt nutzt die CMA-ES das oben beschriebene Prinzip im optimierungsorientierten Kontext:

Bei einem Individuum handelt es sich um einen einzelnen Punkt, welches Teil einer Population ist. Die Population lässt sich als eine Menge an Individuen beschreiben, die über einen Lebensraum, hier die n-dimensionale Landschaft, verteilt ist. Anders als die Individuen selbst, ändert sich die Menge der Individuen im Laufe der Generationen nicht. Die Landschaft entspricht der Fitness-Funktion, welche Abbild einer natürlichen Umgebung ist und Anforderungen an die Individuen definiert. Demnach ist die Fitness eines Individuums ein Qualitätswert, der gewichtet und bei der Rekombination, je nach Mittlung, bevorzugt wird.

Innerhalb einer Population wird anhand des Fitnesswerts ein Rang der am besten angepassten Individuen ermittelt, welche dann zu Eltern der nächsten Generation werden. Die Elternselektion erfolgt allerdings stochastisch, sodass die Individuen mit der höchsten Fitness nicht zwangsläufig zu Eltern werden, jedoch durch das Gewichten die Auswahlwahrscheinlichkeit entsprechend erhöht wird.

Das Erzeugen von Nachkommen verläuft sowohl deterministisch und im Anschluss auch stochastisch gleichermaßen und wird von Variationsoperatoren gesteuert. Dabei werden die Eigenschaften der zufällig gepaarten Elternteile ebenso zufällig kombiniert und durch einen Zufallsfaktor (Mutation) ergänzt. Die Mutation ist ein einstelliger Variationsoperator und erzeugt stochastisch jedes Mal ein anderes leicht modifiziertes Kind [13].

Mathematisch ausgedrückt werden oben genannte Eigenschaften wie in folgendem Muster realisiert. Zum Optimieren einer Funktion $f : \mathbb{R}^n \to \mathbb{R}$ müssen Verteilungsparameter θ innerhalb der Funktionslandschaft und eine Populationsgröße $\lambda \in \mathbb{N}$ initialisiert werden. Wo diese Stichproben liegen ist nicht relevant, da sich der Algorithmus am Funktionswert der Stichprobenergebnisse orientiert. Mit jedem Durchlauf einer Evolutionsstrategie wird folgendes gesetzt:

Algorithmus 4.1 Allgemeiner Ablauf einer Evolutionsstrategie

Gegeben: Optimierungsproblem mit Suchraum und Fitnessfunktion
Lösung: Generiere zufällige Startpopulation θ mit der Populationsgröße λ
while Abbruchbedingung nicht erfüllt **do**
  $P(x|\theta) \to x_1, ..., x_\lambda \in \mathbb{R}^n$ ▷ Setze Individuen
  Evaluiere $x_1, ..., x_\lambda$ auf f ▷ Selektiere nach Fitness
  $\theta \leftarrow F_\theta(\theta, x_1, ..., x_\lambda, f(x^1), ..., f(x_\lambda))$ ▷ Wende genetische Operatoren an
end while

Quelle: [2]

Nach der Beurteilung der einzeln gesetzten Punkte, wird ein Mittelwert-Vektor $m \in \mathbb{R}$ der Punkte ausgewertet. Der **Mittelwert-Vektor** m entspricht der momentan besten Lösung. Um diesen Mittelwert werden die Suchpunkte der nachfolgenden Generation normal verteilt festgemacht. Das Setzen der Suchpunkte innerhalb der Normalverteilung ist eine bewährte Methode, die auch in der Natur beispielsweise bei der Ausprägung phänotypischer Merkmale zu beobachten ist. Zudem ist es in der Statistik die bequemste Methode isotrope Suchpunkte zu generieren:

$$x_i \sim m + \sigma\mathcal{N}_i(0,\, C) \quad für\, i = 1,\, \ldots,\, \lambda.$$

Die **Kovarianzmatrix** $C \in \mathbb{R}^{nxn}$ entscheidet über die Form des Normalverteilungs-Ellipsoids. Die sogenannte **Schrittweitenkontrolle** $\sigma \in \mathbb{R}_+$ überwacht die Schrittlänge. Eine gut angepasste Schrittweitenkontrolle kann die Anzahl der Evaluationen um ein Drittel reduzieren [9]. Das Anpassen von m, C und σ variiert je nach Evolutionsstrategie.

4.3.2 Optimierungsverfahren: CMA-ES

Aufgrund ausführlicher Dokumentation und aktuellem State-of-the-Art-Konzept wird im Folgenden die CMA-ES (Covariance-Matrix-Adaptation Evolution-Strategy) herangezogen. Es handelt sich um eine Methode der stochastischen Optimierung nichtlinearer und nicht konvexer Funktionen. Sofern nicht erläutert, finden sich alle Symbole und Abkürzungen in Tabelle 4.3.

Betrachtet man ein Szenario mit einer Anzahl $n \in \mathbb{N}$ Dimensionen und einer Population g mit einzelnen Individuen x_1^g, ..., x_λ^g, wobei $\lambda \in \mathbb{N}$ der Populationsgröße entspricht. Die erste Bewertung erfolgt und ein gewichtetes Mittel m wird innerhalb

Algorithmus 4.2 CMA-ES-Algorithmus

Gegeben $m \in \mathbb{R}^n, \sigma \in \mathbb{R}_+, \lambda$
Initialisiere $C = I, p_c = 0, p_\sigma = 0$
while Abbruchbedingung nicht erfüllt **do**
$m \leftarrow \sum_{i=1}^{\mu} w_i x_{i:\lambda}$ ▷ Update Mittelwert
$\sigma \leftarrow \sigma \cdot exp(\frac{c_\sigma}{d_\sigma}(\frac{||P_\sigma||}{EN(0,I)} - 1))$ ▷ Update σ
$C \leftarrow (1 - c_1 - c_\mu) \cdot C + c_1 \cdot p_c p_c^T + c_\mu \cdot \sum_{i=1}^{\mu} w_i (\frac{x_i - m}{\sigma})(\frac{x_i - m}{\sigma})^T$ ▷ Update Kovarianz
end while

Quelle: [2; 9]

der Populationsverteilung berechnet. Um diesen n-dimensionalen Vektor wird eine Standardnormalverteilung $\mathcal{N} \sim (m_0, C_0)$ mit $C_0 = I_{nxn}$ gesetzt, um die Stichprobenelemente, zufällig gesetzte Individuen, zu ziehen. Dieser Schwerpunkt dient als Ausgangspunkt zur Entstehung einer neuen Generation $g + 1$

$$m^{(g)} = \sum_{i=1}^{\mu} \omega_i \, x_{i:\lambda}^{(g)} \tag{4.6}$$

Die einzelnen Individuen werden nun nach ihrer Fitness sortiert, sodass Individuen mit guter Fitness ein hoher Gewichts- bzw. Qualitätswert zugeteilt wird. Im Anschluss erfolgt eine Selektion, indem über die μ besten Elemente, einem Rang entsprechend, ein neues gewichtetes Mittel m_1 gesetzt wird. Die Verwendung des Evolutionspfades p_σ mit dem Schrittweiten-Skalierungskoeffizient σ schaffen den Ausgangspunkt der Kovarianzmatrix C_1. Anschließend werden die Elemente der nächsten Generation anhand einer neuen Normalverteilung $N \sim (m_1, C_1)$ gezogen (Mutation des Mittelwertes) [15]. Die Veränderung für jeden Funktionsaufruf finden sich in Algorithmus 4.2.

Kovarianzadaption

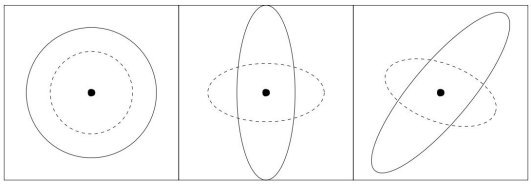


Abbildung 4.7: Normalverteilungs-Ellipsoide
Quelle: [2]

Abbildung 4.7 zeigt mögliche Anordnungen entlang der Kovarianzen abhängig von unterschiedlichen Faktoren:

Die linke Abbildung zeigt eine Standardnormalverteilung mit $\mathcal{N}(m, \sigma^2 I) \sim m + \sigma\mathcal{N}(0, I)$ mit einem Freiheitsgrad σ und unabhängigen Komponenten. Die mittlere

Abbildung 4.7 zeigt ebenfalls n unabhängige Freiheitsgrade mit $\mathcal{N}(m,\, D^2) \sim m + D\mathcal{N}(0,\, I)$ entlang der Diagonalmatrix D. Die rechte Abbildung zeigt korrelierte Verteilungen mit $\mathcal{N}(m,\, C) \sim m + C^{\frac{1}{2}}\mathcal{N}(0,\, I)$.

Die Eigenschaften der Kovarianz bieten die Möglichkeit durch die Gewichtung guter Fitnesswerte das Suchgebiet entlang eines Spats in Richtung eines wahrscheinlichen Optimums zu lenken. Hat ein Szenario n Dimensionen, so hat auch die dazugehörige Kovarianzmatrix n auszufüllende Varianzen in Richtung der einzelnen Koordinatenachsen, welche sich mit einer Hauptkomponentenanalyse berechnen lassen. Betrachtet man die Kovarianzmatrix und subtrahiert die Einheitsmatrix, setzt sich die Diagonale D (siehe Abbildung) aus einzelnen Eigenwerten zusammen. Diese Eigenwerte gehören jeweils zu einer Dimension n innerhalb des $\mathbb{R}^n$. Nun kann jeder dieser Eigenwerte mit einem Faktor γ kombiniert werden, der der Fitness eines Individuums entspricht, sodass daraus ein Eigenvektor resultiert. Diese vom Mittelpunkt m_i ausgehenden Eigenvektoren zu den einzelnen Eigenwerten sind die eben genannten Varianzen der Zufallsvariablen und sind den Spalten der Ladungsmatrix B zu entnehmen.

Zur Anschauung dienen Hyperellipsen, welche die Orientierung der Hauptachsen beschreiben.

Kumulation

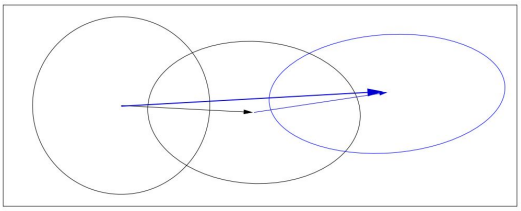


Abbildung 4.8: Konstruktion des Evolutionspfades (Kumulation)
Quelle: [2]

Der Evolutionspfad p_σ und der Kumulationspfad p_c definieren die Summe aller Mittelwertdifferenzen und geben Länge sowie Richtung der Evolutionsschritte beginnend mit $p_\sigma = 0$ bei x_0 an. Der dadurch erreichte Informationsgewinn erlaubt die Optimierung weiterer Iterationen durch den Koeffizienten σ, sollte der Abstand von m_i und m_{i-1} zu groß oder zu klein sein.

$$p_\sigma^{(g+1)} = (1 - C_\sigma)\, p_\sigma + \sqrt{1 - (1 - C_\sigma)^2} * \sqrt{\mu} * \frac{m - m_{old}}{\sigma} \tag{4.7}$$

Schrittweitenkontrolle

σ ist die kumulative Schrittweite (step-size), vergleichbar mit einer Streuung, in Generation g. Sie gibt an, wie hoch der Mutationsfaktor für die nächste Generation

ausfallen soll. Abbildung 4.7 zeigt den Einfluss der Schrittweite auf den Suchraum und wie sich der Hyperellipsoid dadurch vergrößert oder verkleinert. Ist der Evolutionspfad (dauerhaft) kurz, wird σ vergrößert, um das Gebiet auszudehnen. Schwanken die Mittelwertdifferenzen sehr stark, weil sie möglicherweise korrelieren, wird σ verkleinert.

$$\sigma^{(g+1)} = \sigma^g \, exp\,(\frac{c_\sigma}{d_\sigma}\,(\frac{\mid p_\sigma^{(g+1)} \mid}{E \mid N(0,I) \mid} - 1)) \tag{4.8}$$

Parameter/Formel	Beschreibung
$\sigma^g \in \mathbb{R}_+$	Schrittweite
λ	Populationsgröße
μ	Rang/Wertung der Individuen
$B \in \mathbb{R}^n$	Orthogonalmatrix. Die Reihen von B sind Eigenvektoren von C und korrespondieren mit D
$C^g \in \mathbb{R}^{nxn}$	Kovarianzmatrix zur Generation g
$C\sigma$	Lernrate zur Kumulation der Schrittweitenkontrolle
$D \in \mathbb{R}^n$	Diagonalmatrix. Die Elemente von D sind Quadratwurzeln der Eigenwerte von C und korrespondieren zu B
$f : \mathbb{R}^n \to \mathbb{R},\ x \mapsto f(x)$	Fitnessfunktion zum Optimieren
$g \in \mathbb{N}$	Generationsnummer
$m^g \in \mathbb{R}$	Mittelwert einer Suchverteilung in Generation g
$I \in \mathbb{R}^{nxn}$	Einheitsmatrix
$n \in \mathbb{N}$	Suchraum
$N(0,1)$	Multivariate Normalverteilung mit dem Mittelwert 0 und Einheitsmatrix
$N(m,C)$	Multivariate Normalverteilung mit Mittelwert $m \in \mathbb{R}^n$ und Kovarianzmatrix $C \in \mathbb{R}^{nxn}$
$p \in \mathbb{R}^n$/$p\sigma$	Evolutionspfad als Folge normalisierter Evolutionsschritte über eine Abfolge von Generationen
$\theta \in \mathbb{N}$	Verteilungsparameter - Anfangsverteilug der zu optimierenden Parameter
ω_i	Gewichtung mit $i = 1, ..., \mu$
$x_{i:\lambda}$	Anzahl der i besten Individuen aus $x_1^{(g+1)}, ..., x_\lambda^{(g+1)}$

Tabelle 4.3: Abkürzungs- und Formeltabelle

Kapitel 5

Lösungsansatz

5.1 Infrastruktur

5.1.1 Simulationsserver

Die Funktionsberechnung der unbekannten Fitnessfunktion erfordert einen hohen Rechenaufwand. Um möglichst schnell, auswertbare Ergebnisse zu erlangen, ist es von Vorteil, die Berechnungen auszulagern.

Für die praktische Ausführung dieser Arbeit wird die Rechenleistung eines Servers, der von der Fakultät für Informatik der Hochschule Mannheim, zu Verfügung gestellt. Der Server mit dem Namen Cuda läuft mit einer Windows-Server 2008 HPC-Edition. Ausgerüstet ist er mit 2 Intel Xeon Prozessoren mit jeweils zwölf Kernen und einem 96 Gigabyte Arbeitsspeicher.

5.1.2 Analyse der Bibliotheken und Frameworks

Um sich auf die definierten Ziele der Arbeit zu konzentrieren, wurde davon abgesehen die CMA-ES neu zu implementieren. Stattdessen wird das ApacheCommonsMath-Framework in der Version 3.6.1 genutzt, welches auf der von Nikolaus Hansen basierenden Matlab-Implementierung beruht.

5.1.2.1 Apache Commons Math Optimization

Apache Commons ist ein von der Apache Software Foundation ins Leben gerufene Projekt, das Codefragmente und Klassenbibliotheken für die Java-Programmierung zur gezielten Wiederverwendung bereitstellt. Der Nutzen eines robusten und ausgereiften Frameworks erleichtert das Implementieren und Evaluieren, wie es es sich im Laufe der Arbeit zeigt, ungemein. Für diesen Spezialfall wird die Java-API-Erweiterung *Math* genutzt. Und aus diesem Framework das Optimierungs-Paket *org.apache.commons.math4.optim* [1].

Dieses Paket bietet eine Vielzahl an Optimierungsalgorithmen, die zur Berechnung eines Maximums oder Minimums einer Skalarfunktion, im üblichen *ObjectiveFunction* genannt, dienen. Es ist nochmals in einzelne Pakete unterteilt, die speziell an die gegebene Optimierungsumgebung angepasst sind:

Univariate

Der *UnivariateOptimizer* optimiert univariate und reellwertige Problemstellungen.

Linear

Dieses Paket stellt eine Ausführung des Simplex-Algorithmus zur Verfügung zur Berechnung linearer Problemstellungen und linearen Nebenbedingungen.

Direct

Die Algorithmen in diesem Paket sind an Funktionen angepasst, die keine Kostenfunktionen oder Gradienten zur Verfügung stellen. Sie werden genutzt, wenn Gradienten oder Ableitungen nicht oder mit äußerst hohem Aufwand berechenbar sind. Das Paket bietet folgende Algorithmen zur Lösung eines Optimierungsproblems an:

- Classical Nelder-Mead-Algorithmus
- Virginia Torczon's Multi-Directional-Methode
- Nikolaus Hansen's Covariance Matrix Adaptation Evolution Strategy (CMA-ES)
- Mike Powell's BOBYQA

Wie in Kapitel 4 erläutert, wird die CMA-ES für die Optimierung von CC3D-Simulationen ausgewählt. Interessanterweise können mit diesem Framework und nur wenigen Code-Änderungen die Algorithmen ausgetauscht und möglicherweise zum Vergleich herangezogen werden.

General

Das General-Paket behandelt nichtlineare und vektorbezogene Optimierungsprobleme, sofern (partiell) Gradienten oder Ableitungen vorhanden sind. Das Paket enthält zwei Algorithmen:

- Gauss-Newton-Methode
- Levenberg-Marquardt-Methode

Fitting

Das *CuveFitting*-Paket ist primär für Problemstellungen geeignet, die weniger eine *ObjectiveFunction*, sondern eine Vielzahl an Beobachtungen benötigen, um Optimierungsprobleme zu lösen.

5.2 Prototyp

5.2.1 Referenzproblem

Um Machbarkeit, sowie Aufwand und Nutzen abzuschätzen, ist es gebräuchlich sich auf eine Problem ähnlichen Ausmaßes zu beziehen, was als Referenzproblem bezeichnet wird. Dabei werden Testfunktionen benutzt, um Aufschlüsse über Robustheit, Konvergenzverhalten, Performanz und Güte eines Optimierungsalgorithmus zu erlangen. Ein Problem zu referenzieren, von dem weder Funktion noch Fitnesskurve bekannt sind, ist zugegebenermaßen schwierig.

Wie oben bereits erwähnt, ist das Auseinandersetzen mit der Fitnessfunktion erforderlich. Diese Kriterien miteinbezogen, sollte eine Referenzfunktion in diesem Falle möglichst hoch dimensional, sowie nichtlinear sein und über ein globales Optimum verfügen. Ebenso handelt es sich um eine nicht-multi-objektive Problemstellung, da nur eine Fitnessfunktion betrachtet werden soll.

Ein überschaubare Testfunktion ergibt sich bei der Rosenbrock-Funktion [22], die von Howard H. Rosenbrock 1960 vorgestellt und häufig bei Optimierungsproblemen oder zum Testen von Optimierungsumgebungen herangezogen wird.

Die Rosenbrock-Funktion ist flexibel angesichts der Anzahl der Dimensionen und ergibt sich aus der Funktionsgleichung:

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)] \; mit \; x = [x_1, \ldots, x_n]. \qquad (5.1)$$

Die Funktion besitzt mehrere lokale und ein globales Minimum bei $(x_1, \ldots, x_n) = (-1, 1, \ldots, 1)$, was sie leicht und schnell für alle Dimensionen auswertbar macht, siehe Abbildung 5.1.

Die Optimierung erfolgt zuverlässig anhand geeigneter Koordinaten ohne Gradienteninformationen und ohne Approximation entlang einer Geraden oder Funktionskurve [22].

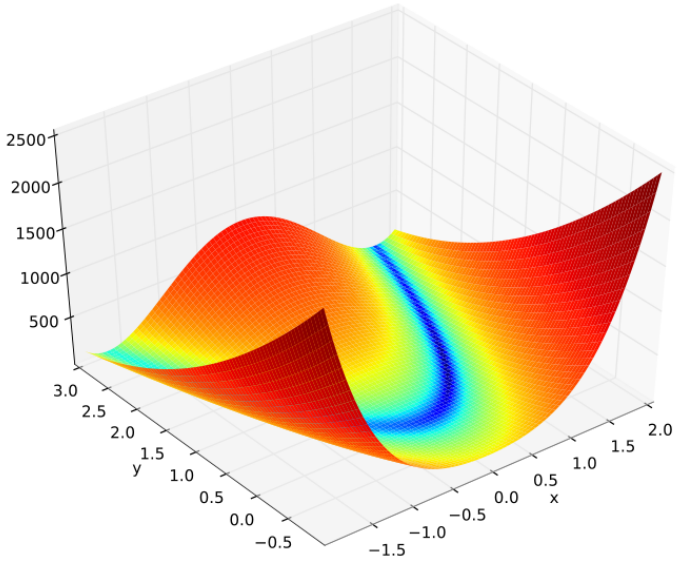


Abbildung 5.1: Entwurf einer 2D-Rosenbrock-Funktion mit a = 1 und b = 100
Quelle: https://en.wikipedia.org/wiki/Rosenbrock_function#/media/File:Rosenbrock_function.svg

Diese Funktion eignet sich besonders gut zum Testen eines Optimierungsalgorithmus und wurde in diesem Fall auch so verwendet.

Allerdings hat die Rosenbrock-Funktion wenig mit der von CC3D generierten Uro-Funktion gemein. Sie ist relativ leicht auswertbar, die Funktionsaufrufe sind schnell berechnet und mit Blackboxing hat das wenig zu tun. Ein anderes Beispiel, das dem Problem näher kommt findet sich in einer Arbeit zur Optimierung von Rennsimulationen [13]. Ungeachtet der Tatsachen, dass auch in dieser Abschlussarbeit die CMA-ES genutzt wird, findet sich eine sehr ähnliche Problemstellung wieder. Das Simulationstool, welches die Simulationen generiert heißt TORCS (The Open Racing Car Simulator) [5]. TORCS simuliert Autorennen mit verschiedenen Fahrzeugen, die wiederum unterschiedliche Zusammenstellungen haben können. Diese variablen Autos können dann auf einer Auswahl an Strecken getestet werden. All diese Komponenten ergeben eine hohe Anzahl diverser Parameter, die parallel in einer Simulation gesetzt und über einen langen Zeitraum „gefahren“ werden.

In der Forschung und Lehre, sowie bei vielen Konferenzen, unter anderem der GECCO (The Genetic and Evolutionary Computation Conference) wird TORCS des öfteren mitunter wegen seiner Stabilität, Modularität und Erweiterbarkeit im Rahmen von Workshops oder Championships genutzt. Ebenfalls wurde für solch eine Challenge eine Optimizer-Erweiterung namens *Car Setup Optimization Competition* implementiert [5]. Diese Erweiterung scheint auf den ersten Blick recht unkompliziert zu sein und optimiert mit einem rudimentären genetischen Algorithmus die Fitness einer TORCS-Simulation. Die Optimizer-Erweiterung kommuniziert mit TORCS mittels UDP-Verbindung über eine Schnittstelle. Ebenfalls ist TORCS primär in C++ implementiert, wo auch die Berechnungen stattfinden. Allerdings erschwert auch hier die Kapselung der Berechnungen die Auswertbarkeit, sowie die Nutzung

externer Optimierungs-Frameworks. Was heißt, dass die Optimierung in TORCS selbst durchgeführt werden muss.

Es gibt demnach viele Gemeinsamkeiten mit CC3D; von den Problemen bis hin zur möglichen Realisierung. Ob und inwieweit sich eine solche Architektur auch für CC3D anwenden lässt, wird in Kapitel 6 näher erläutert.

5.2.2 Aufwandsabschätzung durch Prototyp

CMA-ES

Die Rosenbrock-Funktion braucht mit der CMA-ES zwischen 5000 und 6000 Funktionsevaluationen, um ein endgültiges Optimum von 1e-8 zu für 21 Dimensionen zu erreichen. Die Berechnungen dauern nur wenige Sekunden [11].

UroFunktion

Die UroFunktion ist die unbekannte Funktion, die intern von CC3D berechnet wird. Der Input wird durch die Modelle und die dazugehörigen Parameter gesetzt. Der Output entspricht den jeweiligen Fitnesswerten, die zum Zeitpunkt t_i ausgegeben werden. Die UroFunktion ist das, was mittels ParameterDump darzustellen versucht wird.

Der Vorteil an Evolutionsstrategien ist, dass sie auch auf diesen Anwendungsfall anwendbar sind. Es ist also möglich, die CMA-ES nicht nur auf eine auswertbare und definierte Funktion wie die Rosenbrock-Funktion anzuwenden, sondern auch auf eine Unbekannte.

Die UroFunktion muss aber erst definiert werden, um sie überhaupt optimieren zu können. Was ein Black-Box-Szenario ist und warum es sich hierbei um eines handelt, wird in Kapitel 4 näher erläutert. Gesucht wird eine Fitnessfunktion, die ein möglichst großes $f(x)$ mit möglichst wenigen Black-Box-Aufrufen findet. Wie viele Funktionsaufrufe für die UroFunktion notwendig sind, kann zum Zeitpunkt der Probleminitialisierung nicht abgeschätzt werden.

Doch nun steht die Frage im Raum, wie die UroFunktion aussehen mag:

Zuerst sollte man sich vor Augen führen, dass jede Funktion und jeder Algorithmus eine beliebige Anzahl an Parametern aufweist, die miteinbezogen werden müssen und welche Werte diese einnehmen können. Im Anschluss sollte man sich ebenso überlegen, ob es Nebenbedingungen gibt, die eingehalten werden sollten, beispielsweise ob bestimmte Parameter nur einen bestimmten Wertebereich annehmen dürfen. Näheres zur Approximation der UroFunktion und der Fitness-Funktion ist in Kapitel 6 beschrieben.

Parameter

Wie bereits erwähnt, sind nicht alle Parameter, die für die Simulation am Urothelgewebe angewandt werden, unbekannt. Einige von ihnen sind biologisch nachgewiesen und sollen ihren Default-Wert beibehalten. Folgende Tabelle dokumentiert ein **ParameterDump**. ParameterDumps sind Dateien, die zu Beginn einer Simulationsausführung von Moduro-CC3D ausgegeben werden. Diese Datei beinhaltet Informationen über Modell und Parametersatz einer Simulation. Anhand dieser Datei wird tabellarisch aufgeführt, welche Parameter zur Optimierung sinnvoll erscheinen.

Jeder Zelltyp enthält den gleichen Parametersatz mit unterschiedlichen Werten. Dieser wird deshalb nur einmal notiert. Auch Medium und Basalmembran sind als Zelltypen aufgeführt, werden aber nicht optimiert, da sie im engeren Sinne nicht dem Urothelgewebe angehören.

Parameter	Datentyp	Eignung für Optimierung	Begründung
startTime	TimeStamp	nein	Zeitstempel, der bei der Ausführung einer Simulation neu gesetzt wird
ExecConfig	-	nein	Einführung: Execution-Configuration Block. Dieser Block setzt Attribute zur Ausführung einer Simulation und enthält keine biologisch relevanten Werte
MCSperDay	Integer	nein	Anzahl der Monte-Carlo-Steps pro Tag
SEED	Integer	nein	Initialpunkt für Zelle
boundary_x	String	nein	Grenzenergie zwischen Zellen und Zelltypen
debugOutputFrequency	Integer	nein	Ausgabe der Frequenz
dimensions	Integer	nein	Anzahl der Dimensionen, in der eine Simulation aufgeführt wird
flip2DimRatio	Double	nein	Verhältnis in der Ausgabe der Ansichten
fluctuationAmplitude	Double	nein	Beschreibung der Membranbewegung
initNutrientDiffusion	Bool	nein	Initialisiert einen Nährstoffaustausch im Gewebe

latticeSizeInVoxel	Integer	nein	Initialisiert die Gittergröße
maxSteps	Integer	nein	Anzahl der maximalen Steps in einem Simulationsdurchlauf
neighborOrder	Integer	nein	Anzahl der Nachbarschaftsverhältnisse eines definierten Pixel
sampleIntervalInDays	Double	nein	Häufigkeit einer Stichprobenentnahme pro Tag
sampleIntervalInMCS	Integer	nein	Häufigkeit einer Stichprobenentnahme in Monte-Carlo-Steps
simDurationDays	Integer	nein	Anzahl der simulierten Tage
voxelDensity	Double	nein	Voxel-Dichte
xDimension	Integer	nein	Anzahl der Gitter-Dimensionen in x-Richtung
xLength	Integer	nein	Länge eines Gitterpunktes in x-Richtung
yDimension	Integer	nein	Anzahl der Gitter-Dimensionen in y-Richtung
yLength	Integer	nein	Länge eines Gitterpunktes in y-Richtung
zDimension	Integer	nein	Anzahl der Gitter-Dimensionen in z-Richtung
zLength	Integer	nein	Länge eines Gitterpunktes in z-Richtung

Tabelle 5.2: ParameterDump: Laufzeit-Parameter

Parameter	Datentyp	Eignung für Optimierung	Begründung
ModellName	-		Einführung: Model-Configuration Block. Dieser Block setzt Attribute, die relevant für das Modell als Einheit sind
adhEnergy	Double	ja	Adhäsionsenergie in einem Modell
adhFactor	Double	ja	Faktor zum Multiplizieren der Energiematrix. Wird dieser Parameter optimiert, bleibt die Energiematrix konstant
cellID	Integer	nein	Eindeutige ID zur Erkennung
cellLifeCycleLogger	String	nein	Wird zum Loggen des Objekts verwendet
energyMatrix	Array[Double]	nein	Die Energiematrix wird nicht optimiert, wenn der Adhäsionsfaktor als Multiplikator bereits optimiert wird
name	String	nein	Name des zu optimierenden Modells
necrosisProbBasal	Double	nein	Redundanter Wert, der in der Zelltypbeschreibung nochmal zu finden ist und dort optimiert wird
necrosisProbIntermediate	Double	nein	Redundanter Wert, der in der Zelltypbeschreibung nochmal zu finden ist und dort optimiert wird
necrosisProbStem	Double	nein	Redundanter Wert, der in der Zelltypbeschreibung nochmal zu finden ist und dort optimiert wird
necrosisProbUmbrella	Double	nein	Redundanter Wert, der in der Zelltypbeschreibung nochmal zu finden ist und dort optimiert wird

CellType	-	nein	Einführung: Zelltyp-Configuration Block. Dieser Block setzt Attribute, die für die einzelnen zelltyp-spezifsch sind.
apoptosisTimeInDays	Double	ja, außer Medium und Basal-membran	Die Lebenszeit, die eine Zelle durchschnittlich bis zur eintretenden Apoptose besitzt.
consumPerCell	Double	ja, außer Medium und Basal-membran	Prinzipiell kan dieser Parameter für die Optimierung eingesetzt werden, wird jedoch im aktuellen Kontext nicht genutzt
descendants	Array[Double]	nein	Die Matrix beschreibt mit Hilfe von IDs die Zelltypenreihenfolge innerhalb des Gewebes fest. Die Struktur der Reihenfolge soll dringend beibehalten und nicht optimiert werden
divides	Bool	nein	Teilungseigenschaft eines Zelltyps oder Gewebes
frozen	Bool	nein	Definiert den Zustand einer Zelle
growthVolumePerDay	Double	ja, außer Medium und Basal-membran	Absolutes Wachstum bis zur Mitose
id	Integer	nein	Reihenfolge des Zelltyps innerhalb des Urothelgewebes
maxDiameter	Double	nein	In der Literatur beschriebener und konstanter Wert für den maximalen Durchmesser eines Zelltyps

maxVol	Double	nein	In der Literatur beschriebener und konstanter Wert für das maximale Volumen eines Zelltyps
minDiameter	Double	nein	In der Literatur beschriebener und konstanter Wert für den minimalen Durchmesser eines Zelltyps
minVol	Double	nein	In der Literatur beschriebener und konstanter Wert für das minimale Volumen eines Zelltyps
name	String	nein	Name des Zelltyps
necrosisProb	Double	ja, außer Medium und Basal-membran	Wahrscheinlichkeit des (spontanen) Zelltods
nutrientRequirement	Double	ja, außer Medium und Basal-membran	Gibt an, ob und wieviele Nährstoffe zur Zellteilung notwendig sind. Dieser Parameter wird im aktuellen Kontext nicht genutzt
surFit	Double	ja, außer Medium und Basal-membran	Oberflächen-Fitness eines Zelltyps
volFit	Double	ja, außer Medium und Basal-membran	Volumen-Fitness eines Zelltyps

Tabelle 5.4: ParameterDump: Modell-Parameter

5.3 Gestalt der Fitness-Kurve

Um einen Überblick über die Gestalt der gesuchten Fitness-Kurve zu bekommen, ist Sampling eine Möglichkeit Informationen herauszufiltern. Es gibt viele Methoden

des Samplings und eine geeignete zu finden ist problemspezifisch. In der Statistik bezeichnet das Sampling eine Stichprobenanalyse, die auf die Grundgesamtheit übertragbar ist; in der Nachrichtentechnik ist Sampling das Darstellen eines Kanals eines multiplen Signals.

Biologisch betrachtet, ermöglicht das Samplen das Ausloten der ökologischen Potenz [4], Abbildung 5.2. Das Optimum kommt nur zustande, wenn alle Umweltfaktoren eine perfekte Übereinstimmung erreichen. Bestenfalls bewegt sich der Messbereich des Samplingszenarios innerhalb der Amplitude und hilft beim Erörtern des Optimums.

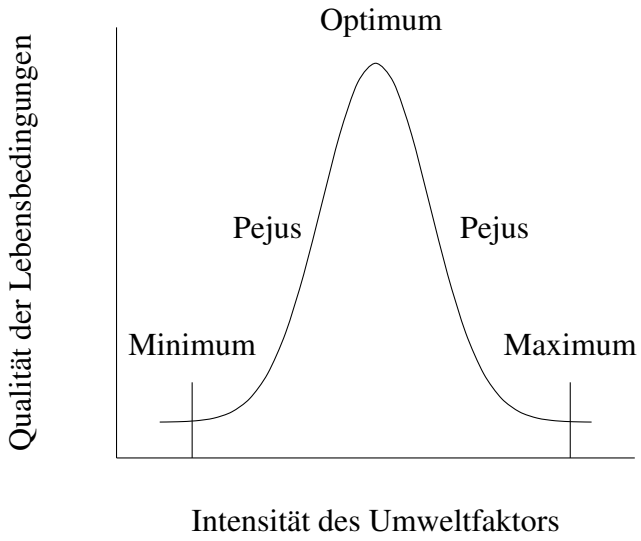


Abbildung 5.2: Ökologische Potenz: Idealisierte Toleranzkurve eines Organismus

Im Falle der Simulation des Urothelgewebes, wird das Sampling zur Approximation genutzt, um Form und Struktur der unbekannten Fitness-Kurve zu erlangen. Ziel ist es dabei, Regressionsmodelle zu erstellen, die sich an die Funktion $f : x \rightarrow y$ mit dem Input $X \subset \mathbb{R}^n$ annähern. Der Output entspricht $Y \subset \mathbb{R}^m$ mit einem Trainingsset aus $T = \{x_i, y_i\}$ Samples [18].

Das Sampling selbst entspricht einer Registrierung von Messwerten zu einem diskreten Zeitpunkt. In diesem Prozess ist die Diskretisierung die Umwandlung der Funktion in einen diskreten Satz von Punkten, welcher zu jedem Funktionsargument ein Set an Funktionswerten jeder Dimension zurückgibt.

Die Zerteilung des Problems in n eindimensionale Teilprobleme erleichtert die Berechnung und auch die Veranschaulichung des Trainingssets. Die Parameter-Blacklist (Tabelle 5.6) beschreibt die exemplarischen Parameter.

Für jeden Parameter φ wird ein sinnvoller Messbereich um den originalen Wert initiiert. Der Messbereich beläuft sich, wenn nicht anders angegeben oder möglich, auf Intervalle von jeweils 25 Prozent im ersten Schritt und 50 Prozent im zweiten Schritt in die positive als auch in die negative Richtung, sodass sich fünf Werte für jeweils einen Parameter ergeben sollten. Sofern der Simulationslauf des Mo-

dells beendet wird, werden Stichproben zu den diskreten Zeitpunkten t_{50}, um die Wachstumsphase besser zu dokumentieren, und im Anschluss von t_{100} bis t_{700} in Hunderter-Schritten gezogen.

Um die Effizienz zu verbessern, laufen die Berechnungen zu den Parametern jeweils parallel: Für jedes Intervall im Messbereich wird ein eigenes Projekt gestartet, um Überschneidungen auszuschließen.

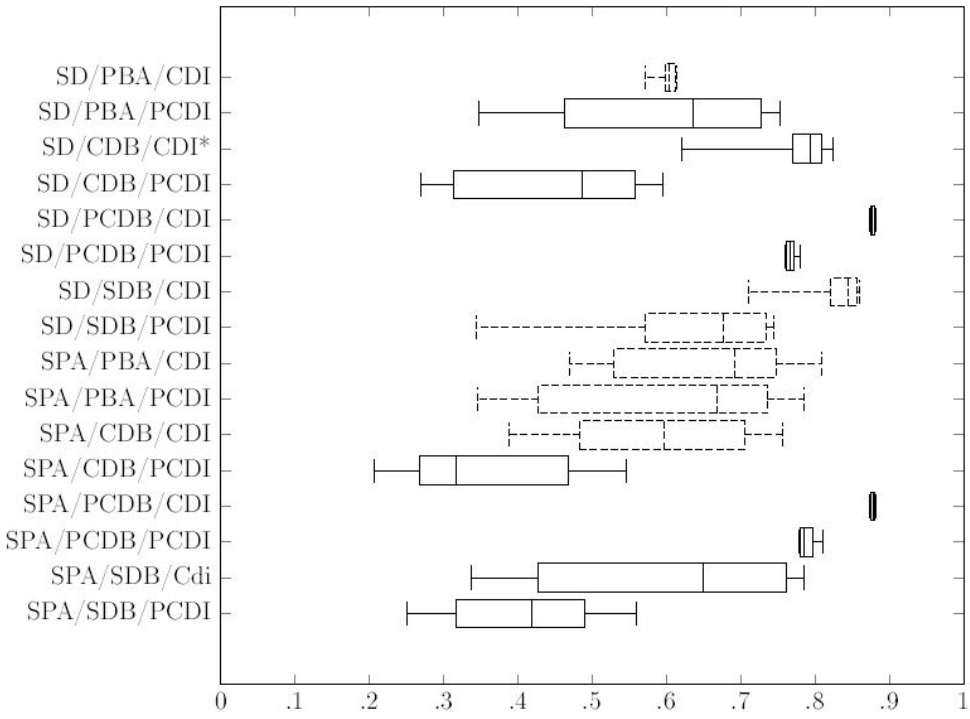


Abbildung 5.3: Modelle im Fitness-Vergleich
Quelle: Generiert aus der Moduro-Toolbox

Aus der oben aufgeführten Abbildung gehen die Verteilungen der Fitnesswerte für jedes Modell hervor. Die Box-Whisker-Plots zeigen jeweils einen Mittelwert, die Standardnormalverteilung, sowie die gesamte Verteilung der dokumentierten Fitnesswerte für jedes Modell auf. Aufgrund begrenzter zeitlicher Ressourcen, wird an dieser Stelle ein Repräsentant für das Parameter-Sampling herangezogen. Sinnvoll ist es hier ein Modell zu selektieren, welches sich im Fitnessranking weder ganz vorne noch ganz hinten befindet. Zudem liegt es nahe, ein Modell auszuwählen, das ein recht breites Spektrum bei der Variation der Parameter anbietet, um mögliche Effekte besser auswerten zu können. Die Wahl fällt damit auf das Modell **SdSdbCdi**. Es erreicht im Schnitt einen passablen Fitnesswert (siehe Abbildung 5.3), zeigt aber dennoch einen gewissen Spielraum nach oben auf. Das Modell wird durch eine symmetrische Stamm- und Basalzellenteilung, sowie durch Kontaktdifferenzierung der Intermediärzellen beschrieben.

Auf das Beispielmodell SdSdbCdi bezogen, ergibt sich folgender Auszug aus dem ParameterDump mit initialen Parameterwerten und zu überprüfendem Messbereich:

Zelltyp	Parameter	- -	-	φ	+	+ +
Model	adhesionEnergy	1.0	1.5	2.0	2.5	3.0
	adhesionFactor	0.15	0.2	0.25	0.3	0.35
Stammzelle	growthVolumePerDay	52.36	54.98	57.59	60.21	62.83
	necrosisProb	/	/	0	0.01	0.1
	nutrientRequirement	0.5	0.75	1.0	1.25	1.5
	surFit	0.25	0.375	0.5	0.625	0.75
	volFit	0.5	0.7	0.9	0.95	0.99
Basalzelle	growthVolumePerDay	5.24	26.18	52.36	78.54	104.72
	necrosisProb	0.00001	0.000015	0.00002	0.000025	0.00003
	nutrientRequirement	0.5	0.75	1.0	1.25	1.5
	surFit	0.25	0.375	0.5	0.625	0.75
	volFit	0.5	0.7	0.9	0.95	0.99
Intermediärzelle	growthVolumePerDay	70.69	123.70	159.04	212.06	265.07
	necrosisProb	0.00001	0.000015	0.00002	0.000025	0.00003
	nutrientRequirement	0.5	0.75	1.0	1.25	1.5
	surFit	0.001	0.05	0.1	0.2	0.3
	volFit	0.5	0.7	0.9	0.95	0.99

Umbrellazelle	growthVolumePerDay	143.65	215.48	287.31	359.14	430.96
	necrosisProb	0.00003	0.000035	0.00004	0.000045	0.00005
	nutrientRequirement	0.5	0.75	1.0	1.25	1.5
	surFit	0.001	0.05	0.1	0.2	0.3
	volFit	0.5	0.7	0.9	0.95	0.99

Tabelle 5.6: Blacklist der variablen Parameter inklusive Messbereich

5.3.1 Durchführung und Erwartungen an die Fitness-Kurven

Um genaue und repräsentative Auswertungen zu erlangen, wird das beschriebene Modell mit dem Default-Parametersatz und jeweils einem verändertem Parameter simuliert. Jede Simulation wird mindestens zwei Mal durchlaufen, um einen halbwegs aussagekräftigen Mittelwert zu erlangen.

Durch das Samplen erhält man ein dreidimensionales Abbild der Kurven, welches man sich folgendermaßen vorstellen kann:

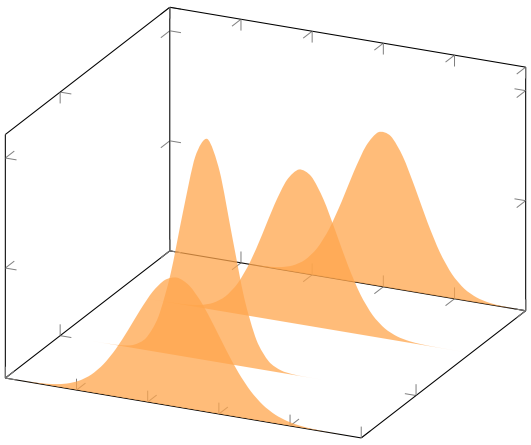


Abbildung 5.4: Beispiel eines Sampling-Szenarios

Die x-Achse entspricht den diskreten Zeitpunkten t_i vom ersten bis zum letzten Tag, die y-Achse dem Fitnesswert und die z-Achse dem variablen Parameter $\varphi_{1,\ldots,n}$. Um allerdings eine übersichtlichere Anschauung auf die Fitness-Kurven zu schaffen,

wird jeder Parameter inklusive Messbereich als eigenes Diagramm dargestellt. Jede Kurve im Diagramm entspricht dem mittlerem Wert einer Versuchsreihe zum Zeitpunkt t_i. So kommt jedes Diagramm auf fünf Kurven, die den initialen, unveränderten Parameter darstellen (schwarz) und jeweils die Adaptionen innerhalb des Messbereichs, sowie die Anzahl der Stichproben n, die zum aufgeführten Mittelwert beitragen und den aktuellen Parameterwert v.

5.3.2 Sampling-Ergebnisse

Das Sampling soll vorab veranschaulichen, ob die ausgesuchten variablen Parameter überhaupt Einfluss auf den Verlauf einer Simulation haben. Verändert sich für den ausgesuchten Messbereich die Fitness einer Kurve nicht, so trägt dieser kaum oder gar nicht zur Veränderung der Güte einer Simulation bei - das heißt: er hat auch optimiert keinen Mehrwert zur Verbesserung der allgemeinen Fitness. Andersherum sind sensible Parameter umso wertvoller zum Erreichen einer möglichst optimalen Fitness.

Die Auswertung des Samplings erfolgt wie oben beschrieben durch den Vergleich der Fitness an ausgesuchten Punkten t_i. Einige Simulationen zeigten bereits während den Läufen starke Schwankungen innerhalb des Messbereichs und deuten demnach auf die Sensibilität eines Parameters hin.

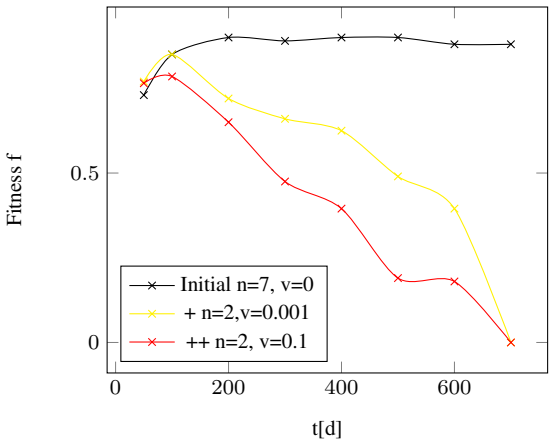
Zur Veranschaulichung dienen die Sampling-Ergebnisse aller Zelltypen, sowie die der Modellparameter.

Tendenziell ist zu sagen, dass die Parameter für das ausgesuchte Modell vorab gut gewählt sind. Kaum ein Parameter konnte nach einer Änderung einen dauerhaft verbesserten Fitnesswert erlangen.

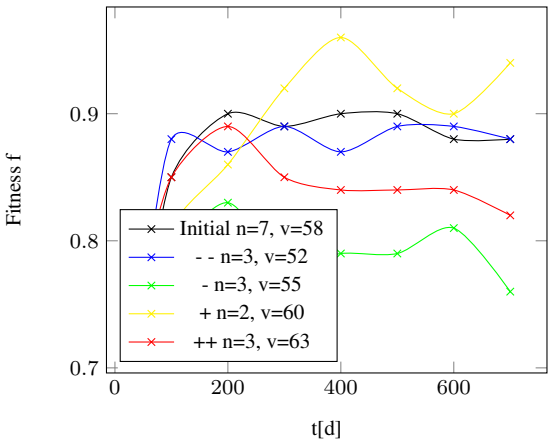
Bei der Datenauswertung fällt auf, dass es innerhalb einer Datenreihe starke Schwankungen gibt, sodass bei einem recht kleinen n Ausreißer umso stärker gewichtet werden und damit den Durchschnitt verzerren. Für künftige Proben sollte unbedingt eine größere Stichprobenmenge eingeplant werden.

Das Samplen soll auch hier als Vorgeschmack zur Optimierung dienen: Optimalerweise sollte das Optimierungsverfahren alle besser ausgefallenen Werte herausfiltern und in diese Richtung optimieren können. So dient das Sampeln zusätzlich als Referenz für das weitere Vorgehen und mögliche Lösungsansätze.

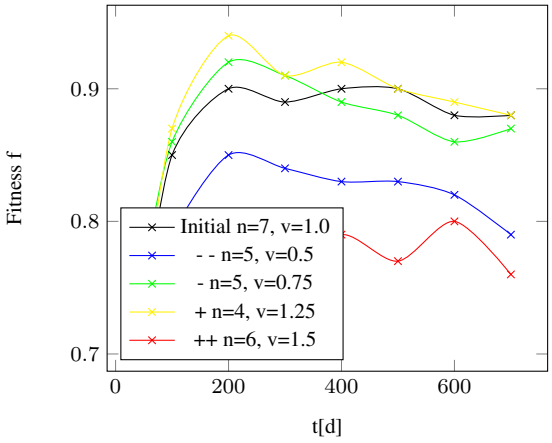
Aufgefallen ist dabei, dass es Parameter gibt, die bei geringer Änderung einen schlechteren Fitness-Wert erlangen konnten als bei größerer Änderung - gemessen an der Initial-Kurve. Eine mögliche Erklärung dafür wäre, dass einige Parameter in Wechselwirkung miteinander stehen und sich womöglich unwissentlich gegenseitig aufheben oder verstärken können.



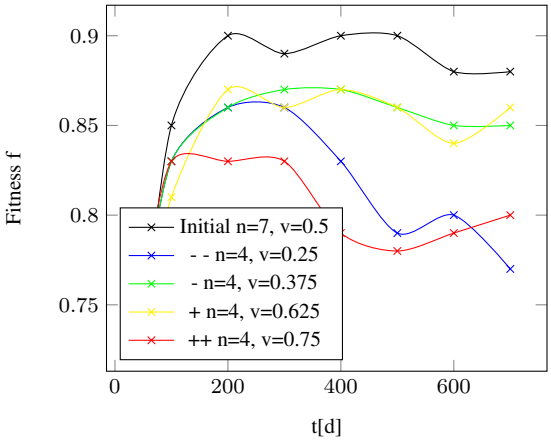
(a) Stammzelle: necrosisProb



(b) Stammzelle: growthRelVolumeDay



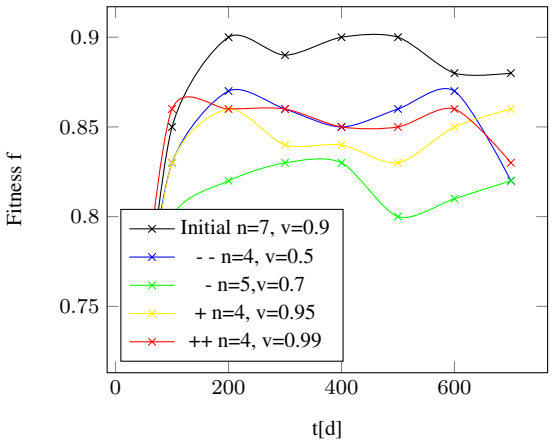
(c) Stammzelle: nutrientRequirement



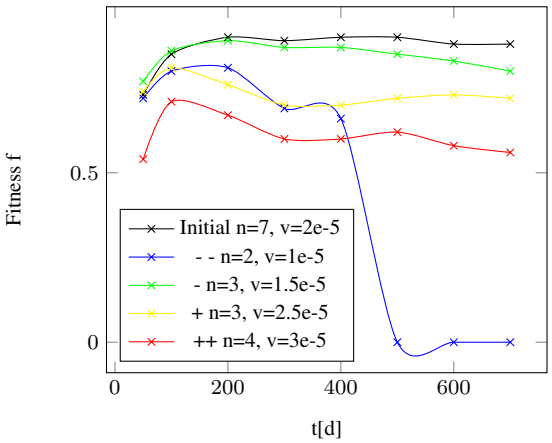
(d) Stammzelle: surFit

Abbildung 5.5: Sampling-Ergebnisse

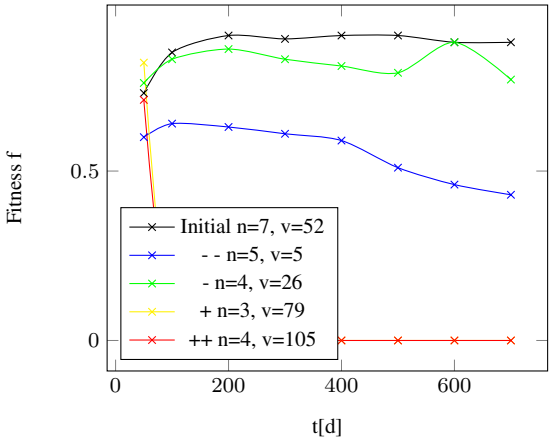
Ein weiteres Kriterium ist das Zellwachstum. Einige Parameter zeigen einen rasanten Anstieg der Fitness innerhalb der Zellwachstumsphase. Das sollte keinesfalls außer Acht gelassen werden, auch wenn es nicht den Steady-State, also den eigentlichen Fluss des Urothels, widerspiegelt. Damit ist anzunehmen, dass sich eine Zelle in der Wachstumshase anders verhält als im gesunden Zustand. Es ist sogar anzunehmen, dass die Wundheilung andere Mechanismen freisetzt, sei es durch äußere Einflüsse oder biologische Signale. Eine Möglichkeit das zu überprüfen, wäre das Verändern entsprechender Parameter nach der Zellwachstumsphase, wenn der Steady-State erreicht ist.



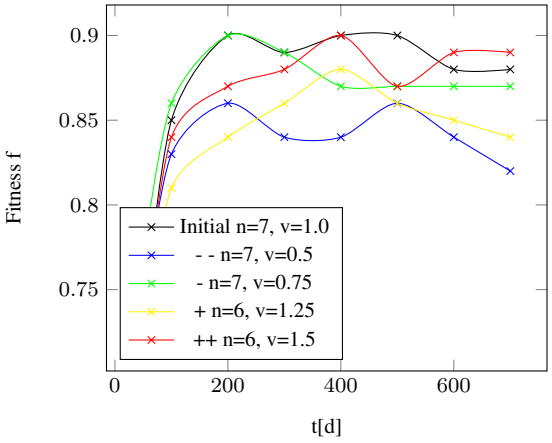
(a) Stammzelle: volFit



(b) Basalzelle: necrosisProb

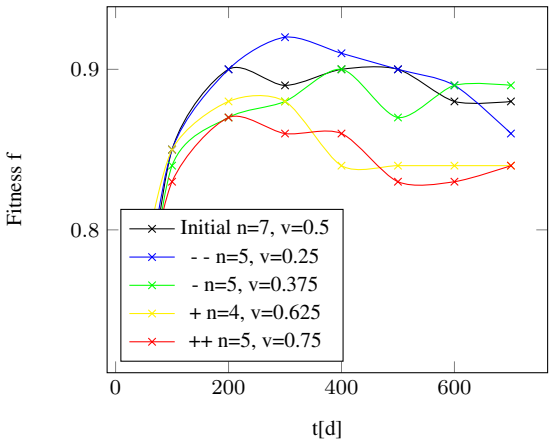


(c) Basalzelle: growthRelVolumeDay

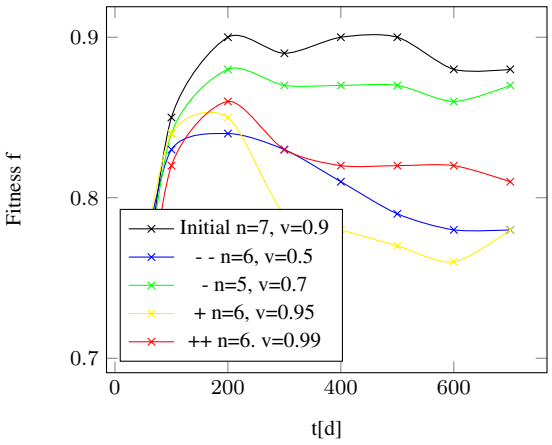


(d) Basalzelle: nutrientRequirement

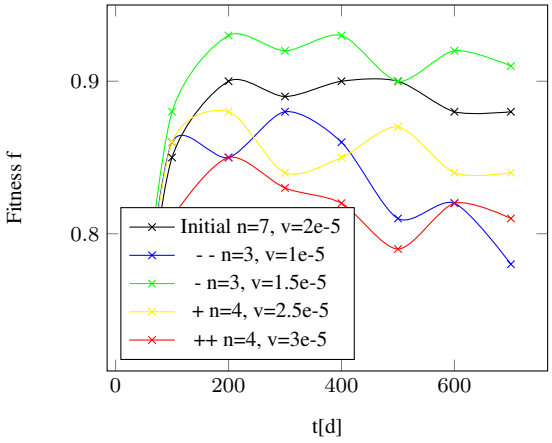
Abbildung 5.6: Sampling-Ergebnisse



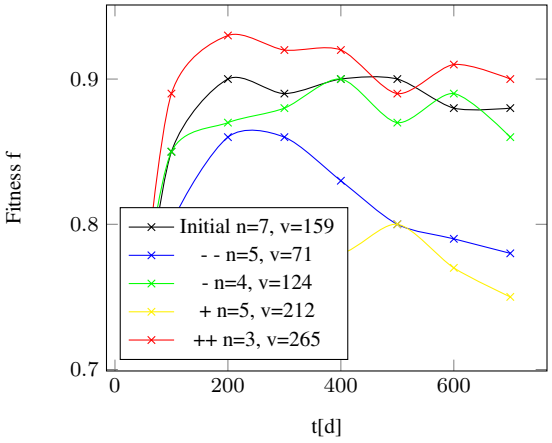
(a) Basalzelle: surFit



(b) Basalzelle: volFit

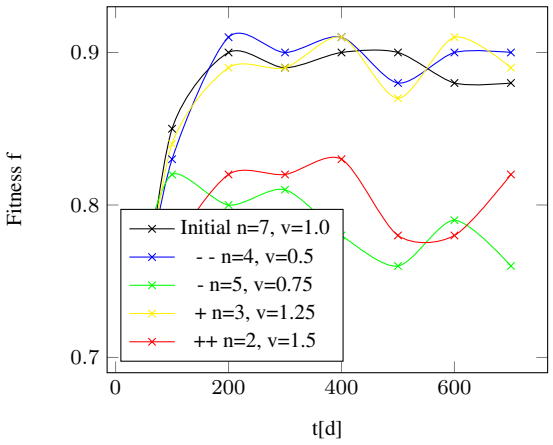


(c) Intermediärzelle: necrosisProb

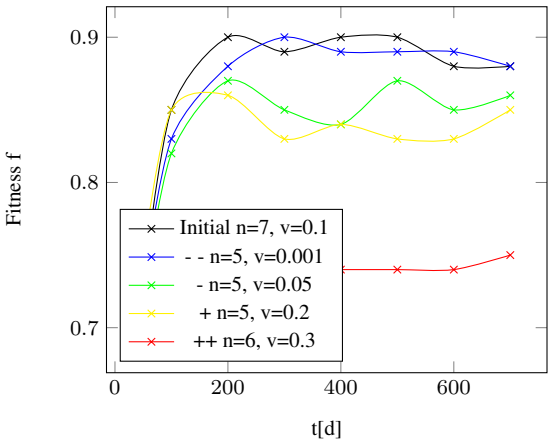


(d) Intermediärzelle: growthRelVolDay

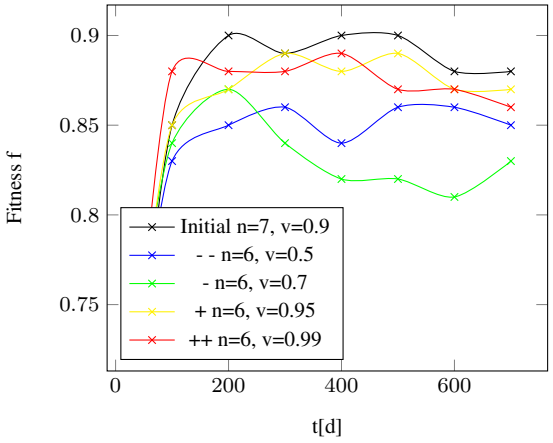
Abbildung 5.7: Sampling-Ergebnisse



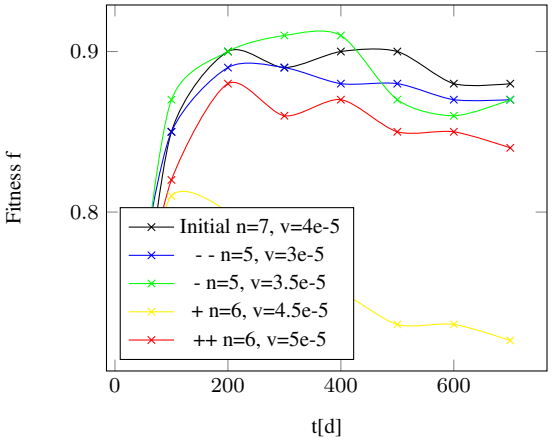
(a) Intermediärzelle: nutrientRequirement



(b) Intermediärzelle: surFit

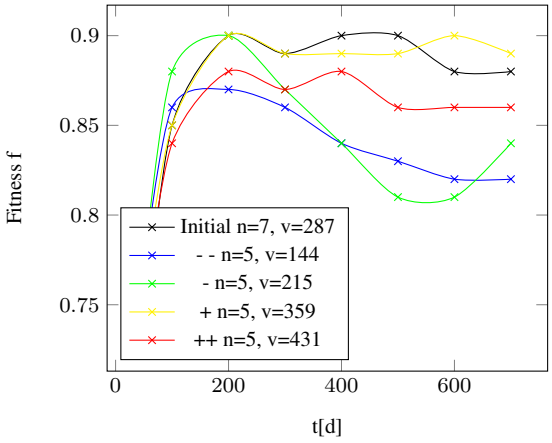


(c) Intermediärzelle: volFit

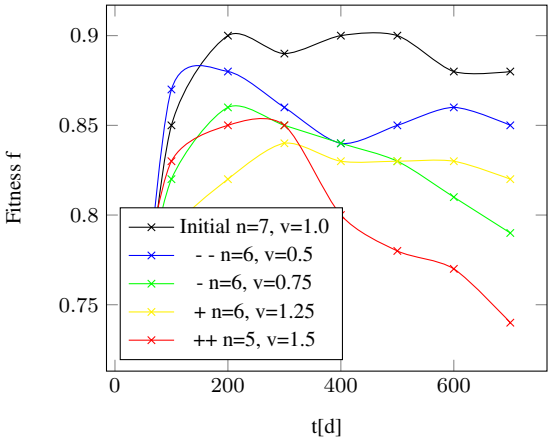


(d) Umbrellazelle: necrosisProb

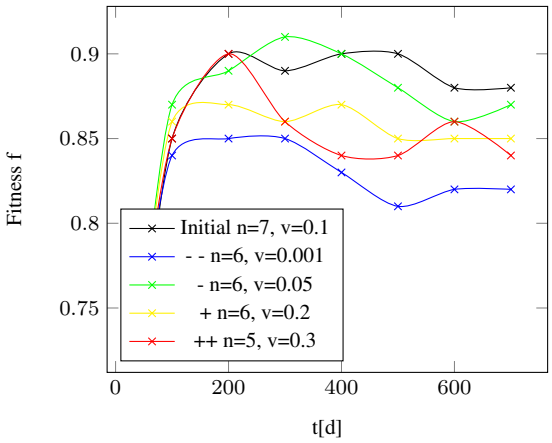
Abbildung 5.8: Sampling-Ergebnisse



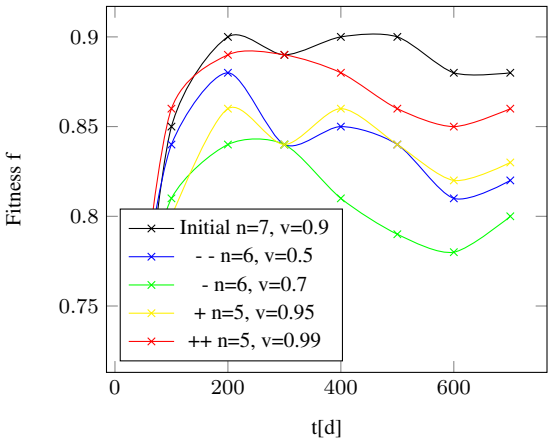
(a) Umbrellazelle: growthRelVolDay



(b) Umbrellazelle: nutrientRequirement

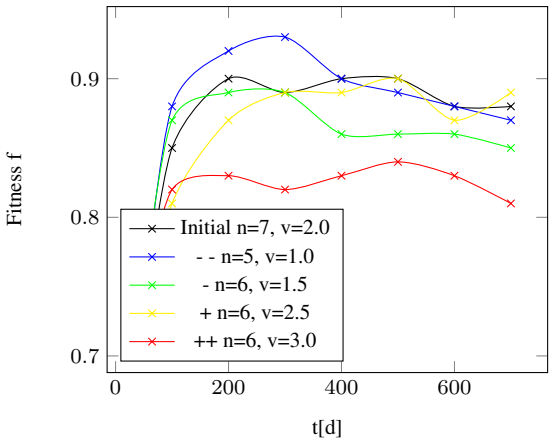


(c) Umbrellazelle: surFit

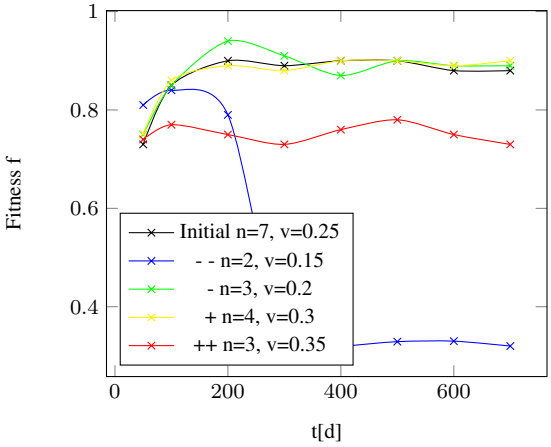


(d) Umbrellazelle: volFit

Abbildung 5.9: Sampling-Ergebnisse



(a) Model: adhEnergy



(b) Model: adhFactor

Abbildung 5.10: Sampling-Ergebnisse

Kapitel 6

Optimierungsergebnisse

6.1 Design und Implementierung von ModuroOptimizationAutomation

6.1.1 Anforderungen

In diesem Abschnitt wird genauer erläutert, welche Anforderungen an das Projekt gestellt worden sind, wie vor allem die technische Machbarkeit ausfällt und ob die tatsächliche Realisierung der Komponenten den Anforderungen entspricht.

- Evaluieren von Optimierungsverfahren zur problemspezifischen Verwendung der Fragestellung

  Durch interne Kommunikation hat man sich auf die CMA-ES geeinigt. Nichtsdestotrotz war eine genaue Recherche und Auflistung der Problemstellung notwendig, um den Nutzen und die Tauglichkeit der Evolutionsstrategien zu erörtern. Nähere Informationen zur Problemstellung und dem Lösungsansatz finden sich in Kapitel 4.

- Ausarbeitung eines Java-Tools, welches einen automatisierten Workflow zum Optimieren von Moduro-Modellen, möglichst in der Moduro-Toolbox, bereitstellt

  Das in Java 8 entwickelte Tool optimiert Urothel-Simulationen, indem einzelne Modelle von CC3D übergeben werden und diese dann von dem ausgesuchten Optimierungs-Framework optimiert und bewertet werden. In jedem Optimierungslauf wird nur ein Modell optimiert. Um ein anderes Modell zu optimieren, muss es explizit übergeben werden. Der Automatisierungsdurchlauf ermöglicht das Starten aufeinanderfolgender Optimierungsläufe, falls aufgrund unzureichender Werte ein Optimierungslauf abbricht. Der ursprüngliche Ansatz das Optimierungstool aus der Moduro-Toolbox zu starten, war lange aufrecht erhalten worden. Hintergrund des Gedankens ist die bessere Wartbarkeit, das Wissen und das Wiederverwenden bereits implementierter Komponenten. Die Implementierung des Automatisierungskonzepts hat bis-

weilen keine stabile Einheit mit der Moduro-Toolbox gebildet. Die Moduro-Toolbox ist eine GUI-Applikation und erschwert den Start beziehungsweise das Handling mit anderen Software-Modulen, wie in diesem Fall CC3D. Zudem macht so ein Konzept im Rahmen der Entwicklung wenig Sinn, da die Automatisierung ohne GUI anzuwenden ist. Technische, praktische und zeitliche Gründe führten daher zu einer unabhängigen Ausarbeitung und so zu einem eigenständigen Tool.

- Möglichst hohe Modularität der Komponenten zur Wiederverwendung und zum möglichen Austausch durch weitere Tools

  Das Optimierungstool unterlag immer wieder Veränderungen aufgrund verbesserter Modularitäten. Zum einen ist das Optimierungstool mit möglichst vielen Interfaces ausgestattet, was dem generischen Plan entgegenkommen soll. Das Parsen und die Verarbeitung der Parameter aus den ParameterDumps erfolgt beispielsweise über eine Konstruktion aus Collections und Hashmaps, was heißt, dass die Gestalt der ParameterDumps nicht zwangsläufig eingehalten werden muss. Es können beliebig Parameter hinzugefügt und entfernt werden ohne Inhalte zu verlieren oder dass Fehlermeldungen geworfen werden. Ebenso verhält es sich mit dem Verändern der ParameterDumps selbst mittels Reflections und dem Einsatz des standardisierten Formats JSON, welches zur Serialisierung herangezogen wird. Weiteres findet sich in den kommenden Abschnitten zur genaueren Beschreibung des Optimierungstools.

6.1.2 ModuroOptimizationAutomation

Um das Optimierungstool zu benennen, um welches es sich in dieser Arbeit handelt, wird der Name **ModuroOptimizationAutomation** gewählt. Die Ausführung des Tools erfolgt, anders als bei den anderen Komponenten, durch die Kommandozeile.

Im folgenden Abschnitt werden alle bisher vorgenommenen Erweiterungen von CC3D, sowie das Grundkonzept und ein Optimierungsablauf vorgestellt.

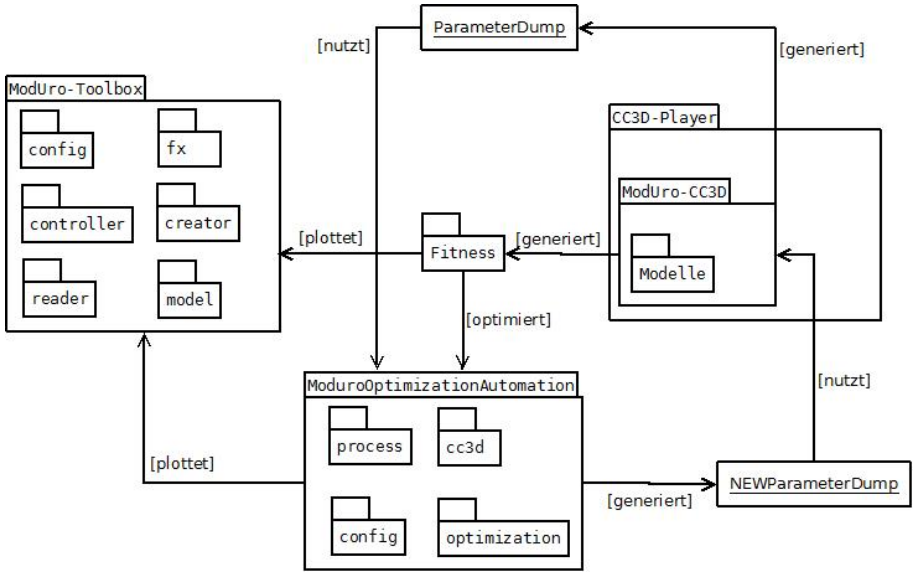


Abbildung 6.1: Design-Idee zur Verwirklichung der Optimierungskomponente

Die Abbildung 6.1 beschriebt den groben Aufbau und die Idee, die hinter dieser Arbeit steckt. Moduro-CC3D liefert statische Modelle, die eine mehr oder weniger gute, durchschnittliche Fitness erreichen. Ziel ist es, eine Verbesserung der Fitness für möglichst alle Modelle zu erreichen. Hierzu ist das ParameterDump das zentrale Dokument in diesem Szenario. Diese Datei liefert alle Parameter und die zugehörigen Default-Werte beim Starten einer Simulation. Diese Parameter sollen nun im ModuroOptimizationAutomation-Tool so angepasst werden, dass die durchschnittliche Fitness eines Modells verbessert wird.

Das Programm beginnt mit der Analyse eines ParameterDumps, das zu Anfang bei der Konfiguration übergeben werden muss. Dieses ParameterDump läuft initial als Simulation durch. Die Fitness wird ausgewertet und durch die CMA-ES optimiert. Nach der Optimierung wird das NEWParameterDump in Form einer JSON-Datei serialisiert. Diese Datei dient nun als Basis für die darauf aufbauende Iteration. Anhand der optimierten Werte wird das Modell erneut gestartet, diesmal jedoch nicht auf Basis eines Modells, sondern durch die im NEWParameterDumps enthaltenen Werte. Die Simulationen starten immer auf Basis der JSON-Dateien.

Die CMA-ES iteriert bis ein optimales Ergebnis erreicht ist. Wird ein Optimierungsvorgang dadurch oder aufgrund anderer Begebenheiten, wie vorzeitiges Abbrechen oder fehlerhaftem Verhalten, beendet, sorgt die Automatisierungsfunktion dafür für da Wiederaufnehmen des Optimierungsprozesses, sobald eine dieser abbricht.

Dieser Vorgang wird wiederholt, bis der CC3D-Simulation-Manager manuell abgebrochen oder ein Abbruchkriterium erreicht wird.

6.1.2.1 Erweiterung von Moduro-CC3D

Wie bereits erläutert, ist Moduro-CC3D ein Python-Projekt, was bedeutet, dass sowohl die Vorgänge als auch die Modelle Python-basierte Skripte sind. Die Aufgabe von Moduro-CC3D ist in diesem Kontext primär das Auslesen optimierter Dateien und das Starten von Simulationen auf Basis dieser Dateien. Dies soll über die von CC3D generierten ParameterDumps geschehen. Diese Dateien haben kein standardisiertes Format, was das Ein- und Auslesen zwischen den Tools erschwert hat. Aus diesem Grund wird das JSON-Datenformat (JavaScript Object Notation) eingesetzt, um die Kommunikation zwischen den beiden Komponenten zu vereinfachen. Ein Parser existiert in fast allen Programmiersprachen und ist zudem durch die einfache Textform lesbar. Ein Import des JSON-Parsers reicht aus, um das Format nutzen zu können.

Der Ausbau der Moduro-CC3D-Schnittstelle basiert auf dem Zusammenstellen alter und neuer Parameter in einem neuen, JSON-basierten Template. Prinzipiell wird hier das Zusammensetzen des Modells wie üblich gehandhabt: Die konventionellen Modelle listen Eigenschaften und die dazugehörigen Parameter aus den übergeordneten Klassen, wie der *CellType*-Klasse, zusammen. Die Struktur für ein optimiertes Modell ist analog, werden die Parameter jedoch aus einem JSON-File, einem optimierten NEWParameterDump, und nicht aus der *ModelConfig* oder aus den *CellTypes* gelesen. Zudem gibt es nur eine JSON-basierte Klasse für alle Modelle. Das Konstrukt ist modellunabhängig, damit sich auch jedes Modell darstellen lässt. Diese triviale Herangehensweise erspart das Warten überflüssiger Modelle und weiterer Klassen.

Bisher muss das Modell manuell übergeben werden, was heißt, dass das optimierte Modell nicht generisch ausgesucht werden kann. Sofern das Modell, in dem Fall der initiale ParameterDump nicht gewechselt wird, läuft genau dieses Modell im Optimierer.

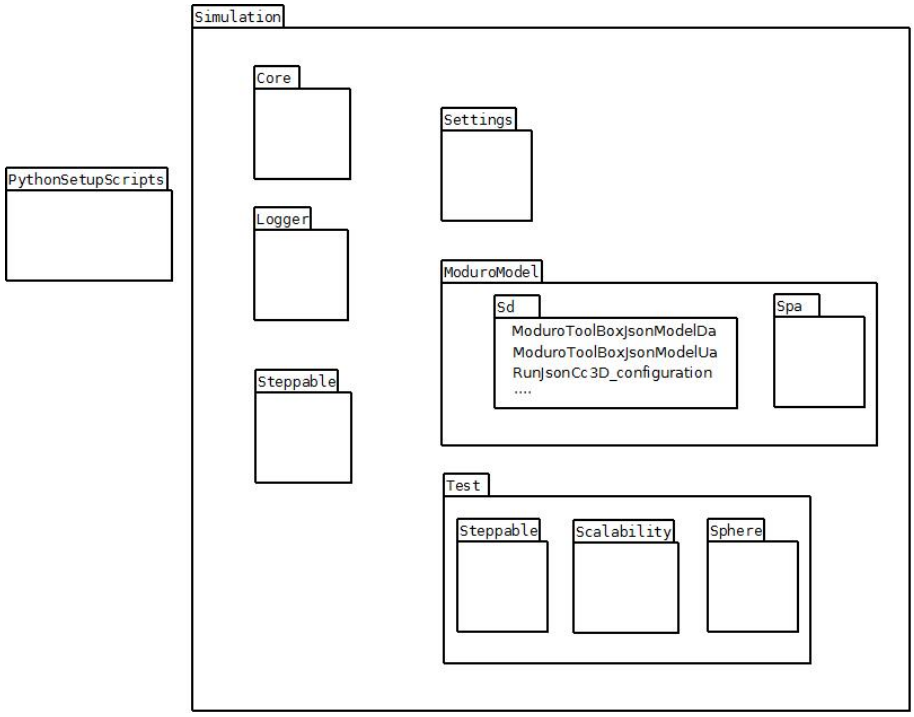


Abbildung 6.2: Moduro-CC3D-Paketdiagramm

Die im *Sd*-Paket hinzugefügten Zeilen entsprechen den neu hinzugefügten Klassen, siehe Abbildung 6.2. Von der Struktur und Anwendung ähneln sie, wie bereits beschrieben, den Modellklassen und unterscheiden sich lediglich in der Ausführung. Analog wie die anderen Modellklassen, befinden sich wegen der Übersichtlichkeit auch die auf JSON-basierten und optimierten Modellklassen im gleichen Verzeichnis.

6.1.2.2 Aufbau von ModuroOptimizationAutomation

ModuroOptimizationAutomation ist ein in Java implementiertes Tool, das die Ergebnisse der CC3D-Simulationen auswerten und optimieren soll. Das Tool liest die von CC3D generierten Verzeichnisse aus und bewertet die gewünschten Simulationsläufe.

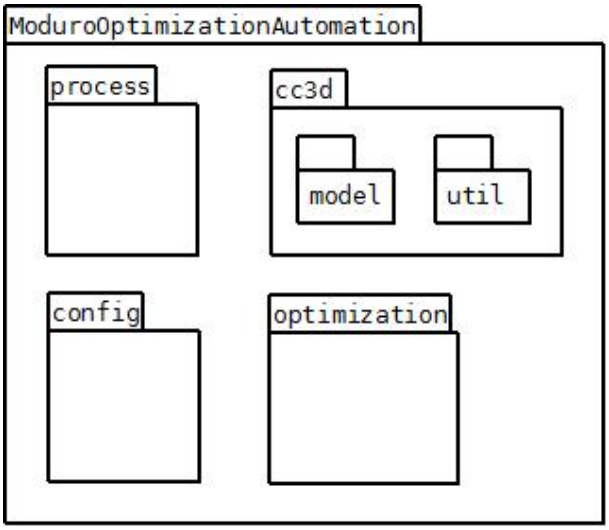


Abbildung 6.3: Paketdiagramm von ModuroOptimizationAutomation

Das dazugehörige ParameterDump wird hierzu herangezogen, um ein Optimierungsmodell zu generieren. Das optimierte ParameterDump wird im Folgenden als JSON-Datei abgelegt und für die laufenden Simulationen verwendet.

Wie in Abbildung 6.3 zu sehen ist, besteht ModuroOptimizationAutomation aus vier Paketen. Die Aufgaben und Funktionsweise der einzelnen Pakete wird wie folgt beschrieben.

process

Dieses Paket (Abbildung 6.4) verwaltet den Automatisierungsprozess des Tools. Die Attribute entsprechen den Konfigurationsparametern, die zu Beginn gesetzt werden müssen. Zuerst werden die angelegten Verzeichnisse auf Vorhandensein überprüft. Im Anschluss wird das referenzierte ParameterDump geladen und weitergegeben. Nachdem das ParameterDump serialisiert wurde, wird es als JSON-File exportiert, indem das Objekt übergeben und als JSON-String in der *ToJson*-Methode formatiert wird. Nun liegt es für CC3D bereit. Ebenso wird ein CC3D-File erstellt, um CC3D zu starten. Sind all diese Dinge erfolgreich abgeschlossen, wird der CC3D-Prozess gestartet und die Simulation beginnt. Das Paket fungiert übergeordnet und steuert die Prozesse nach außen. Bei Abbruch einer Simulation beginnt eine weitere Optimierung.

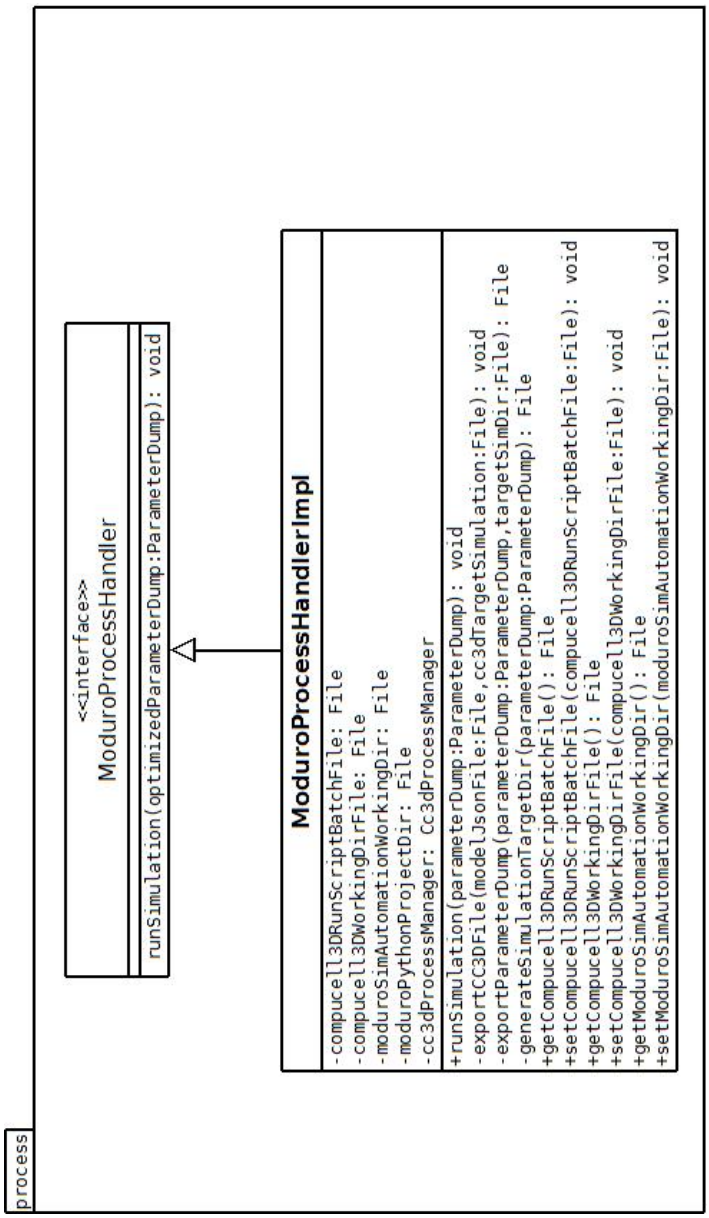


Abbildung 6.4: *process*-Paket

cc3d

Aufgrund der Paketgröße wird dieses Paket aufgeteilt und separat beschrieben. Hier findet der Serialisierungsvorgang statt. Das Paket verarbeitet ParameterDumps, filtert die wichtigen Parameter zum Optimieren und fügt diese nach dem Optimierungsvorgang wieder zusammen.

model

Das Unterpaket *model* beschreibt die Logik innerhalb des Serialisierungsvorgangs. Die einzelnen Klassen wie *ParameterDumpExecConfig*, *ParameterDumpCellType* und *ParameterDumpModel* entsprechen unterschiedlichen Blocktypen, die im *util*-Paket herausgelesen werden. Die einzelnen Typen müssen auf unterschiedliche Weise behandelt werden und sind daher separat zu bearbeiten. Weil die Abwicklung der *descendants*-Attribute innerhalb der Zelltypen besonders komplex war, werden diese nochmals separat behandelt und serialisiert. Innerhalb der parameterverarbeitenden Klassen, werden die einzelnen Parameter in *key* und *type* gesplittet. Unter anderem werden die Werte auf den Namen und korrekte Typauswertung mit Hilfe von Annotations überprüft. Hierbei handelt es sich um Annotations zum Übergeben eines Parameters, bei dem sich der Wert zur Laufzeit ändern kann. Die Werte bleiben so lange in Verwahrung wie die Annotation selbst. Der Optimierer spricht die gesuchten *keys* an und nutzt diese zum Optimieren.

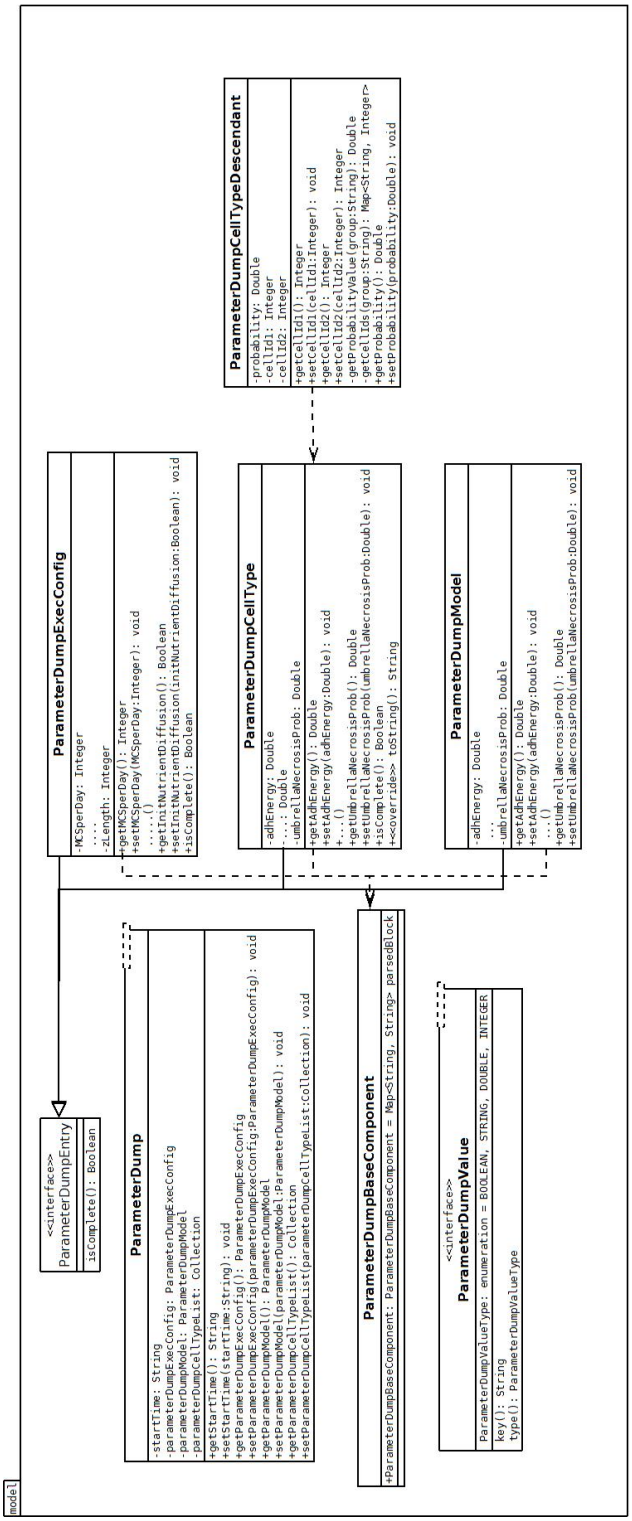


Abbildung 6.5: *model*-Paket

util

Das Unterpaket *util* (Abbildung 6.6) liest das zu verarbeitende ParameterDump oder eine JSON-Datei ein und setzt sie wieder zusammen. Die Verarbeitung selbst geschieht im *model*-Paket. Jeder ParameterDump wird in acht logische Blöcke eingeteilt. Zwei Blöcke beschreiben CC3D-Einstellungen und sind für die Optimierung uninteressant. Nichtsdestotrotz werden diese auch serialisiert, das sie zum Zusammensetzen der JSON-ParameterDumps benötigt werden. Es werden nämlich alle Parameter in die NEWParameterDumps hinzugefügt, sodass sie das Format möglichst beibehalten. Übrig bleiben die Blöcke mit dem Inhalt zu den einzelnen Zelltypen. Jeder Zelltyp besitzt die gleichen Parameter mit unterschiedlichen Wertigkeiten. Die umständliche Handhabung mit den Blöcken ist deshalb notwendig, da einfache Map-Strukturen einen mehrfach vorkommendes Schlüsselattribut überschreiben. Jedes ParameterDump wird daher ausgelesen und auf Schlüsselattribute überprüft. Ein Schlüsselattribut, genannt *MasterKey*. Nachdem alle Zeilen des ParameterDumps ausgelesen und nach den Doppelpunkten gesplittet werden, werden die Zeilen auf Länge und und Vollständigkeit überprüft. Das heißt, Leerzeilen werden ebenso ausgeschlossen und die Blöcke für jeden *MasterKey* separiert. Somit sind die Einträge der einzelnen Blöcke flexibel, solange die Formatierung bestehen bleibt. Diese werden in Collections gespeichert. und erst jetzt in Hashmaps abgelegt werden. Jeder Parameter bekommt einen individuellen Schlüssel mit dazugehörigem Wert zugeschrieben. Der Ansatz hat einen weiteren praktischen Aspekt: Das Hinzufügen und Entfernen von Parametern durch CC3D beeinträchtigt das Auslesen der ParameterDumps nicht.

Der *ParameterDumpWriter* setzt die einzelnen Blöcke zum Ausführen in CC3D wieder zusammen. Das geschieht über den *StringBuilder*, der über die *append*-Methode die einzelnen Bestandteile dem NEWParameterDump hinzufügt. Erst später, im *process*-Paket, wird das ParameterDump-Objekt über *Gson* zu einer JSON-Datei

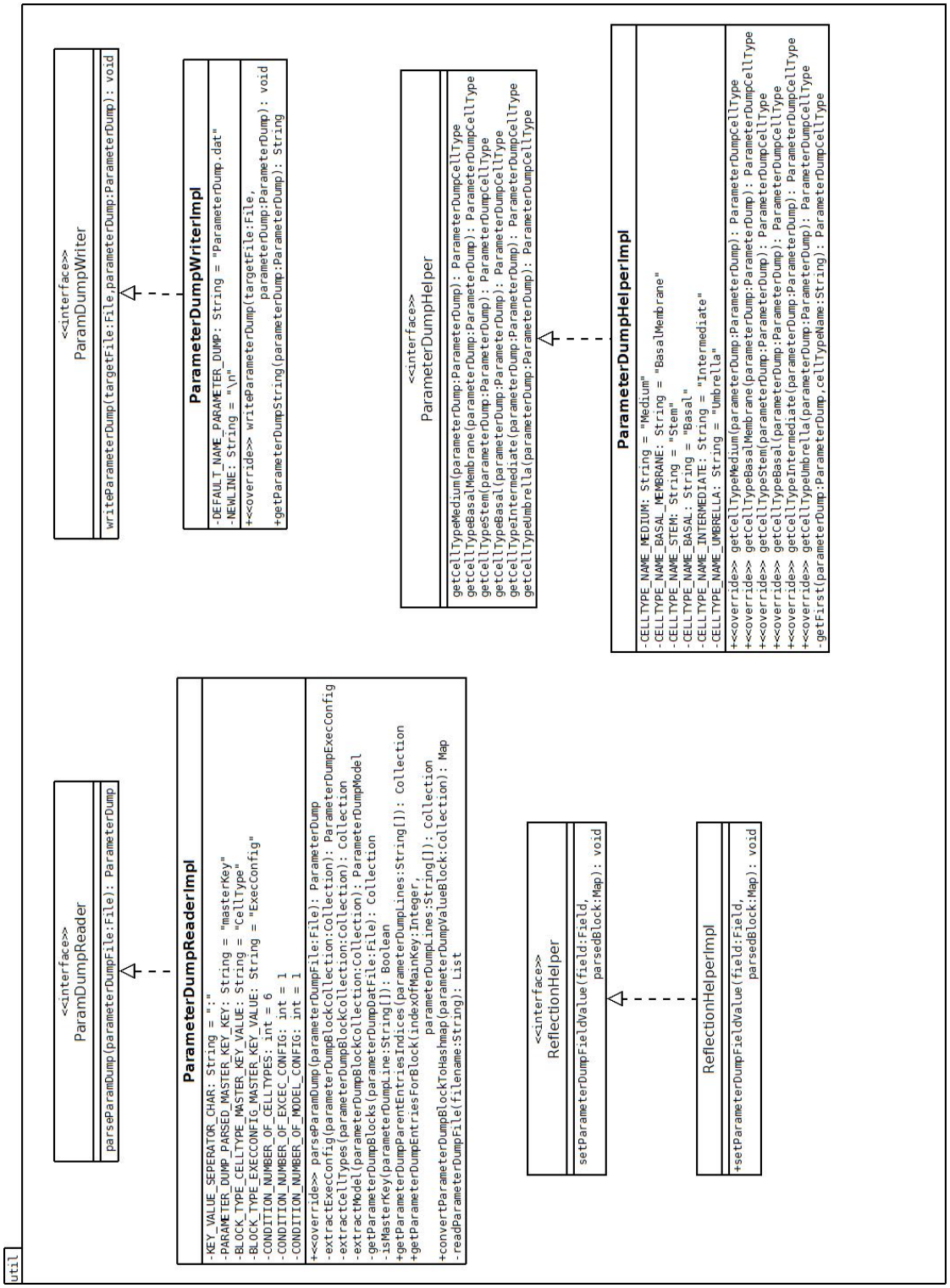


Abbildung 6.6: *util*-Paket

config

Das Paket *config* dient als Hilfe zum Starten eines CC3D-Prozesses aus der ModuroOptimizationAutomation (Abbildung 6.7) . Da in diesem Kontext CC3D nicht nur ausgeführt, sondern auch Parameter wie ein Modell oder andere Einstellungen mitgegeben werden sollen, wird diese Klasse zur Konfigurationshilfe angelegt. Mit Hilfe des CLI-Frameworks können Parameter gesammelt und aus der Kommandozeileneingabe geparst werden, um diese ohne statische Pfadeingaben nutzen zu können. Ebenfalls finden sich hier einige Parameter, die zur Konfiguration des CMA-ES-Optimierers benötigt werden.

optimization

Im *optimization*-Paket (Abbildung 6.8) findet die eigentliche Optimierung der Simulationen statt.

Der Klasse *CompuCell3DParameterOptimizer* werden alle gefilterten Werte aus den zu optimierenden ParameterDumps als Attribute übergeben. Auch die Initialisierung des CMA-ES-Optimierers erfolgt in dieser Klasse. Dem Optimierer berechnet eine interne *ObjectiveFunction*. Die Funktion ist als solches nicht gegeben. Die geforderte ObjectiveFunction verlangt in diesem Fall eine Simulation. Um der Methode, die eigentlich einen mathematischen Term erwartet, einen Prozess unterzujubeln, muss die Simulation gewrappt werden. Die Methode wird aufrecht erhalten, bis der Simulationsprozess beendet ist. Als Rückgabewert wird der Fitnesswert der UroFunction zurückgegeben. Wie sich dieser Fitnesswert berechnet, wird in 6.5 beschrieben.

Der erste Durchlauf ist immer ein initialer Lauf, das heißt, dass hier noch keine Parameter optimiert werden. Erst die zweite Iteration manipuliert die Parameter und setzt diese in das NEWParameterDump ein.

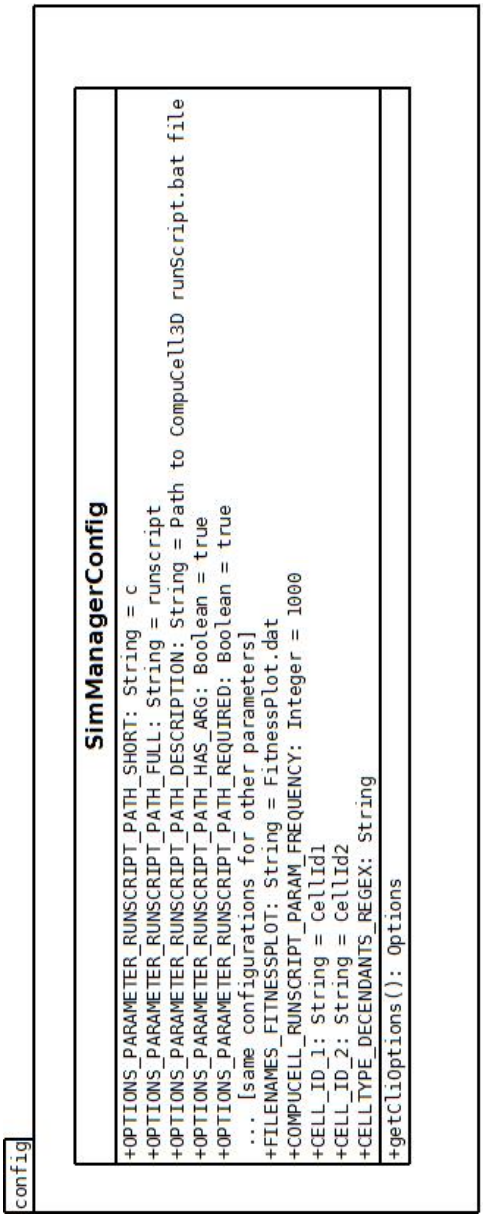


Abbildung 6.7: *config*-Paket

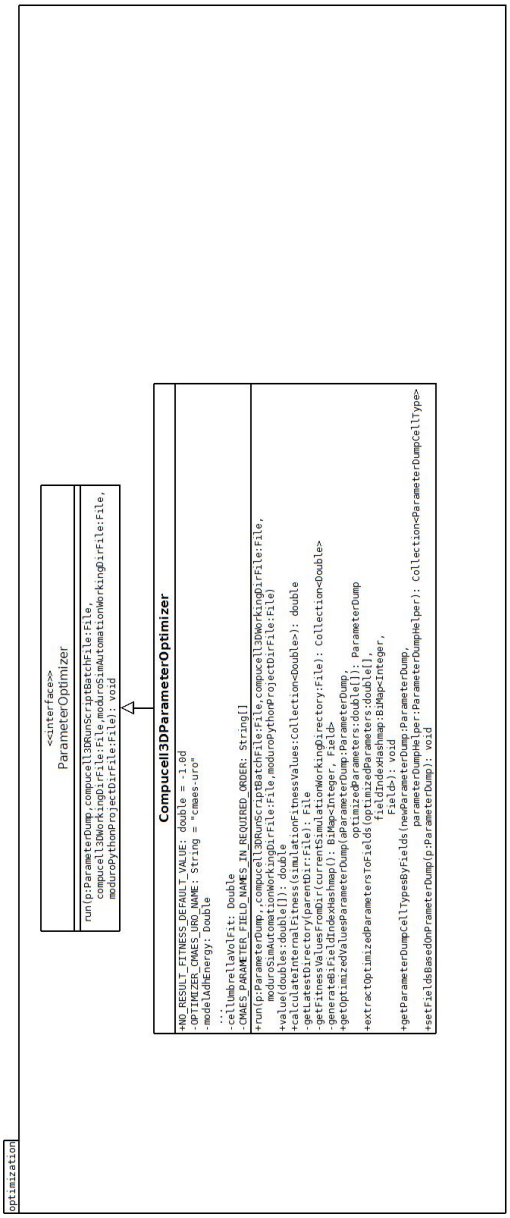


Abbildung 6.8: *optimization*-Paket

6.1.3 Aufgetretene Probleme

Folgende Probleme sind während der Implementierung der Optimierungslösung aufgetreten:

- Das Parsen der Parameter hat sich als recht aufwändig herausgestellt, wie es sich an den Klassendiagrammen erkennen lässt. Die vielen verschiedenen Typen innerhalb eines ParameterDump müssen ihren Typ, Wert und Hintergrund beibehalten. Unter anderem war das Parsen der *descendants*-Parameter schwierig zu verwirklichen, sowie das Setzen der Sterbewahrscheinlichkeiten. Fließkommazahlen werden vom Java-Compiler anders verarbeitet und gesetzt als im Python-Interpreter.
- Das Einhalten der Parameterreihenfolge im ParameterDump stellte sich an sich schon als Herausforderung dar, jedoch fallen in einigen Modellen die Parameter im Model-Block anders aus und es muss sichergestellt werden, dass möglichst alle Modelle eingelesen werden können und neue Werte auch den richtigen Parametern zugeschrieben werden.
- Die Moduro-Toolbox lief nicht immer zuverlässig, wenn nicht der genaue Workflow eingehalten wurde. Waren die Verzeichnisse der Workspaces leer oder beinhalteten Ordner, die nicht der Namenskonvention entsprachen, ließ sich das Tool nicht starten. Das Einbauen von Logs war nötig, um eine bessere Fehlererkennung zu gewährleisten.
- Das Entwickeln in CC3D gestaltet sich dahingehend als schwierig, da - anders als in anderen Python-Projekten - die Möglichkeit zum Debuggen nicht zur Verfügung steht. Die Ausgaben in der Anzeige waren allerdings recht hilfreich bei der Fehlersuche und -behebung.
- Das CMA-ES-Projekt enthielt ebenfalls Fehlerquellen, die sich äußerten, wenn man die Default-Parameter umstellen wollte. Jedoch war das Debuggen und das Setzen einiger semantischer Maßnahmen ausreichend, um auftretende Fehlerquellen zu beseitigen.

6.2 Definitionen und Erläuterungen zum Verständnis der Durchführung

Im Verlauf dieser Arbeit wird beschrieben, dass:

$$UroFunktion \iff Fitness - Funktion \iff ObjectiveFunction$$

Doch warum werden alle drei Begriffe in unterschiedlichem Kontext genutzt, wenn sie doch jeweils als Synonym des anderen Begriffs dienen sollten?

Was auch zu Anfang der Bearbeitung noch nicht präsent war, wird hier näher erläutert:

UroFunktion

Die UroFunktion beschreibt den Wertebereich, der als Input der Optimierung dient. Sie beschreibt die Berechnung der 22 zu optimierenden Parametern und wird in diesem Szenario durch ein (gefiltertes) ParameterDump dargestellt.

Durch interne Kommunikation wurde die UroFunktion definiert als $f(x) = (\varphi_1, ..., \varphi_n, t)$. Wie in den Sampling-Beispielen schon zu sehen ist, entspricht jeder Parameter φ einer Kurve. Alle φ Kurven ergeben zusammen eine mehrdimensionale Funktion.

Bei der Betrachtung aller Optimierungsparameter, beginnend von der Adhäsionsenergie bis zur Volumen-Fitness der Umbrellazellen, ergibt sich also:

$$f(adhEnergy, ..., umbrellaVolFit) \triangleq f(\varphi_1, ..., \varphi_n) \tag{6.1}$$

Wie allerdings dieses $f(x)$ wirklich aussieht, bleibt aber immer noch unbekannt, nimmt jedoch folgende Struktur an:

$$f(\varphi_1, ..., \varphi_n) = (x \cdot \varphi_1 + x \cdot \varphi_2 + ... + x \cdot \varphi_n)$$

Jedoch braucht man die Form oder Gestalt von $f(x)$ auch nicht genau zu kennen, um die Funktion zu optimieren. Was allerdings unbedingt vonnöten ist, ist eine Metrik, die den Ausgang dieser UroFunktion abbildet. Das findet sich in der unten beschriebenen Fitness-Funktion, deren Resultat eine verschachtelte Darstellung der UroFunktion ist. Denn jeder Wert, der zu einem Zeitpunkt t berechnet wird, ist ein Ergebnis aus $f(\varphi_1, ..., \varphi_n)$ mit einem Wert aus $\bar{f} = (\frac{f_v + f_a}{2})$.

Vereinfacht formuliert ergibt die Optimierung einen Skalar mit jeweils einem ParameterDump und einem Wert aus [0,1].

Fitness-Funktion

Die Fitness-Funktion beschreibt in diesem Kontext eine von [26] definierte Funktion, die die von Moduro-CC3D produzierten Fitnesswerte nutzt, um die allgemeine Güte einer Simulation möglichst gut zu beschreiben. Die Funktion fungiert als Metrik und bewertet den Output der Simulationen. Die Kurzform beschreibt einfach gesagt die Anordnung und das Volumen des Urothelgewebes für jeden Monte-Carlo-Step mit gleicher Gewichtung durch die Formel: $\bar{f} = (\frac{f_v + f_a}{2})$.

Ausgeschrieben heißt die Formel:

$$\overline{f} = \left(\frac{\left(\frac{1}{(1-L_B)+(L+L_I)+(1-L_U)+1}\right) + \left(\left(\frac{1}{\frac{(S_B-I_B)^2}{a-V_{max}}+1}\right) + \left(\frac{1}{\frac{(S_I-I_I)^2}{a-V_{max}}+1}\right) + \left(\frac{1}{\frac{(S_U-I_U)^2}{a-V_{max}}+1}\right)\right)}{2} \right) \tag{6.2}$$

Wichtig ist es zu unterscheiden, was den Werte- und was den Definitionsbereich einer Funktion ausmacht. Die Fitness-Funktion beschreibt den Wertebereich im Intervall zwischen [0,1] für den Output der Simulation. Die Ergebnisse der Fitness-Funktion finden sich in den FitnessPlot-Dateien, die während eines Simulationslaufs generiert und dokumentiert, sowie für die Optimierung genutzt werden.

ObjectiveFunction

Die *ObjectiveFunction* ist die Bezeichnung, die in den Frameworks genutzt wird, um eine Fitness-Funktion zu beschreiben. Die hier genutzte *ObjectiveFunction* ist die gewrappte Fitness-Funktion. Bei der Referenzanalyse bezeichnet die *ObjectiveFunction* die Rosenbrock-Funktion. Es ist also immer die Funktion, wogegen im aktuellen Kontext optimiert wird.

6.3 Workflow für Simulationsläufe

6.3.1 Parametereinstellung

Zur akkuraten Ausführung einer Optimierung sollte nicht nur eine gut definierte Fitness-Funktion, sondern auch eine adäquat eingestellte Parameterumgebung vorhanden sein.

Prinzipiell eignet sich die Default-Einstellung der CMA-ES-Parameter nicht besonders gut. Um bessere Ergebnisse zu erlangen, ist es äußerst wichtig, den Algorithmus und die Optimierungsparameter zu kennen und zu verstehen. Anbei ein kleiner Auszug der wichtigsten Parameter zur Optimierung einer Funktion des Apache Commons Frameworks für die CMA-ES:

Die Optimierung basiert auf der abstrakten Basis-Klasse der **MultivariateOptimizer**, welche eine **ObjectiveFunction** berechnet und ein **PointValuePair** zurückgibt.

Parameter des CMA-ES-Quellcodes:

maxIterations - Maximale Anzahl an Iterationen, die der Algorithmus durchlaufen soll.

stopFitness - Ist der Funktionswert der *ObjectiveFunction* kleiner als der *stopFitness*-Wert, wird der Optimierungsdurchlauf abgebrochen.

isActiveCMA - Die Kovarianz-Update-Methode wird gewählt.

diagonalOnly - Anzahl der initialen Iterationen, an denen die Kovarianzmatrix diagonal bleibt.

checkFeasableCount - Bestimmt die Anzahl der zufällig gewählten Zufallszahlen und ob diese *OutOfBounds* sind.

random - Der Zufallsgenerator zum Generieren der Zufallszahlen.

generateStatistics - Funktion zum Sammeln von Statistiken.

maxEval - Anzahl der erlaubten Evaluationen für die *ObjectiveFunction*.

f - ObjectiveFunction auf Basis einer *MultivariateFunction*. Diese Funktion entspricht der zu optimierenden Funktion und gibt einen Vektor als Eingabeparameter mit. Der Output einer *ObjectiveFunction* ist das *PointValuePair* dargestellt durch einen Skalar.

goalType - Bestimmt den Optimierungstyp. Der *goalType* gibt an, ob eine Funktion maximiert oder minimiert wird.

CMAESOptimizer.Sigma - Sigma-Eingabewerte zum Bestimmen der Abweichung für neue Suchpunkte um den *InitialGuess*-Punkt. Empfohlen sind Distanzen, die bei der Suche des Optimums helfen: globale Optima lassen sich mit entsprechend großem Sigma besser finden, da diese in die gesamte Suchlandschaft streuen. Zu kleine oder kleine Werte können zum frühen Beenden einer Suche führen.

CMAESOptimizer.PopulationSize - Die Anzahl der zu Beginn strategisch möglichst günstig gesetzten Populationsgröße. Eine gute Populationsgröße befindet sich bei etwa 4 + 3 $ln(n)$, wobei n die Anzahl der zu optimierenden Parameter darstellt. Das Anpassen der Populationsgröße verbessert ebenso die globale Suche bezüglich der Geschwindigkeit.

PointValuePair - Konvergenzprüfer, in diesem Falle ein *PointValuePair*. Diese Klasse gibt einen Funktionswert und einen dazugehörigen Parameterwert zurück. Diese Klasse selbst bringt wiederum eigene Eigenschaften mit sich:

point - Punktkoordinaten der gefundenen Optimierungspunkte.

value - Wert der gefundenen Optimierungspunkte.

upperBounds - Ist ein Vektor, der für jeden Eingangswert eine obere Grenze definiert.

lowerBounds - Ist ein Vektor, der für jeden Eingangswert eine untere Grenze definiert.

6.3.2 Nebenbedingungen

Da jedes Optimierungsproblem sehr individuell ist, ist es auch vonnöten, dieses so gut wie möglich zu beschreiben. Das Setzen einer Fitness-Funktion muss gelegentlich durch Nebenbedingungen ergänzt werden. Beispielsweise darf eine Funktion aus einem von außen bestimmten Grund keine Werte $<= 2$ annehmen. So verhält es sich auch in diesem speziellen Fall: Die Fitnesswerte der *ObjectiveFunction* dürfen keine Werte außerhalb des Intervalls [0,1] annehmen. Ebenfalls ist durch den Ablauf optimierter Simulationen festzustellen, dass einige der übergebenen Parameter keine negativen Funktionswerte annehmen dürfen. Beispielsweise kann ein Wachstumsfaktor nicht negativ sein, da die Zelle ja nicht schrumpfen soll oder auch kann. Um diese Option ganz auszuschließen, werden alle Parameter dieser Bedingung unterzogen.

Nebenbedingungen werden implementiert, indem ein Wrapper um die *ObjectiveFunction* geschachtelt wird oder das Implementieren von *Upper-* und *LowerBounds*. Durch das individuelle Handling kann jeder optimierte Wert in einen Wertebereich gesetzt werden.

6.3.3 Workflow zur Ausführung einer Optimierung

Anforderungen

Das Tool wurde unter folgenden Bedingungen entwickelt und getestet. Andere Versionen sind, sofern nicht weiter angegeben, unter Vorbehalt nutzbar:

- Das Tool ist mit Compucell3D 3.7.4
- Windows 10
- Java 8 und Maven 3.0.5

Abhängigkeiten

- ModuroOptimizationAutomation unter *https://github.com/informatik-mannheim/Moduro-Optimizer*
- Moduro-CC3D unter *https://github.com/informatik-mannheim/Moduro-CC3D*
- Moduro-Toolbox bei Bedarf unter *https://github.com/informatik-mannheim/Moduro-Toolbox*

Diese Abhängigkeiten beinhalten nicht die im Sourcecode verwendeten Frameworks.

Konfiguration der ModuroOptimizationAutomation

Ein Optimierungslauf braucht ein Working-Directory, um die gespeicherten NEWParameterDumps und die runJSON.cc3d-Dateien abzulegen. Dort wird für jede

Simulation ein eigenständiges Verzeichnis mit dem geparsten Modellnamen angelegt. Um der Applikation Parameter zu übergeben, wird die runScript.bat zum Starten von CC3D benötigt, die auch als Argument mitgegeben wird. Leider gibt es in diesem Modus keine Möglichkeit ein Verzeichnis zu lokalisieren, um den Ablageort der cc3d-Dateien aufzurufen. Das Workspace-Verzeichnis, das von CC3D standardmäßig angelegt wird, kann weiterhin zur Auswertung und zum Einsehen der Fitness genutzt werden.

Um das Setzen von festen Dateipfaden zu umgehen, werden die Verzeichnisse über die Programmargumente übergeben. Lediglich die UnitTests werden durch manuell gesetzte Pfade aufgestellt.

Folgende Parameter müssen vor dem Lauf übergeben werden:

- c - Pfad zur *runscript.bat*-Datei. Diese Datei startet den CC3D-Prozess.
- p - Pfad zum initialen ParameterDump im CC3D-Workspace, welches es zu optimieren gilt.
- w - Pfad zum CC3D-Workspace. Das Verzeichnis, das von CC3D zum Ablegen der generierten Dateien wie ParameterDumps und Fitness-Dateien nutzt.
- t - Pfad zum SimulationManager-WorkingDirectory. Dieses Verzeichnis muss angelegt werden, um die optimierten NEWParameterDumps im JSON-Dateiformat abzulegen. Aus diesem Verzeichnis zieht CC3D die zum Simulieren benötigten JSON-Dateien. Für jeden Optimierungslauf wird ein Ordner mit dem Namen des Modells angelegt. Innerhalb dieses Ordners wird für jede Iteration ein Ordner errichtet, der den Zeitstempel trägt, der den Zeitpunkt des Iterationsbeginns dokumentiert.
- s - Pfad im CC3D-Verzeichnis, welches ein Python-Modell referenziert. Dieser Pfad soll auf die *RunJson.py*-Datei zeigen, die von CC3D aufgerufen wird.

Beispielanwendungen finden sich in der *readme.md*.

6.4 Beispiel einer Optimierung

Ist ModuroOptimizationAutomation ausreichend konfiguriert, steht der Benutzung nichts im Wege. Bei der Implementierung der Kommandozeilenapplikation wurde viel Wert auf das Logging gelegt, da das, gerade bei der Entwicklung mit CC3D, meist die einzige Möglichkeit war, die Prozesse zu überblicken. Die Ausgaben im Terminal geleiten den Benutzer so durch den Optimierungsverlauf.

Anbei werden die wichtigsten Ausgaben des Tools abgebildet und beschrieben, um den Optimierungsverlauf nachvollziehen zu können:

Abbildung 6.9 zeigt das Verzeichnis des initialen ParameterDumps, das herangezogen wird. Im Anschluss wird dieses ParameterDump zerlegt. Die Datei wird durch-

laufen und nach Schlüsselelementen durchsucht, die für die Kapselung der einzelnen Blöcke verantwortlich sind. Die *ParentElements* zeigen die Zeilen innerhalb des ParameterDumps, in denen sich die Schlüsselelemente befinden.

Abbildung 6.10 beschreibt wie die Elemente den Collection-Blöcken zugeteilt werden. Abgebildet ist der Block *ExecConfig*. Die weiteren Blöcke werden analog verarbeitet. Ist die Datei wieder zusammengesetzt, werden die restlichen Abläufe abgearbeitet.

Sofern alles korrekt abläuft, beginnt der CC3D-Prozess mit der Simulation (Abbildung 6.11).

Ist der Simulationsvorgang beendet, werden die Fitnesswerte der Simulation aus dem FitnessPlot des Workspace bezogen und berechnet, siehe Abbildung 6.12. Die mittlere Fitness bewertet die Güte des letzten Simulationsdurchlaufs. Jetzt wird das ParameterDump erstmals optimiert und gestartet. Abbildung 6.13 zeigt wie die CMA-ES die Werte manipuliert und neu setzt. Dieser Vorgang erstreckt sich nun über alle Iterationen.

```
"C:\Program Files (x86)\Java\jdk1.8.0_101\bin\java" ...
Starting ModUro automation system at date: Sun Feb 19 09:15:58 CET 2017
entering parseParamDump
target file is: C:\Users\Station\CC3DWorkspace\SdCdbCdiInDa.cc3d-2017-01-31-12-39-07-821000\ParameterDump.dat
splitting ParameterDump file.
ParameterDump: {}
getting indices of mainKey which introduce datablocks
Indices of ParentElements: [2, 25, 37, 55, 73, 91, 109, 127]
```

Abbildung 6.9: Initialer ParameterDump

```
getting all parameters which belong to entry: ExecConfig:
Adding value ExecConfig:to collection.
Adding value MCSperDay: 500to collection.
Adding value SEED: 7517to collection.
Adding value boundary_x: Periodicto collection.
Adding value debugOutputFrequency: 50000to collection.
Adding value dimensions: 2to collection.
Adding value flip2DimRatio: 0.5to collection.
Adding value fluctuationAmplitude: 10.0to collection.
Adding value initNutrientDiffusion: Falseto collection.
Adding value latticeSizeInVoxel: 48000to collection.
Adding value maxSteps: 360000to collection.
Adding value neighborOrder: 1to collection.
Adding value sampleIntervalInDays: 0.5to collection.
Adding value sampleIntervalInMCS: 250to collection.
Adding value simDurationDays: 720to collection.
Adding value voxelDensity: 0.8to collection.
Adding value xDimension: 400to collection.
Adding value xLength: 500to collection.
Adding value yDimension: 120to collection.
Adding value yLength: 150to collection.
Adding value zDimension: 1to collection.
Adding value zLength: 0to collection.
index 24 has an empty row. Block is complete
parameter dump block contains 22 elements.
Masterkey found
hashmap contains 22 elements
```

Abbildung 6.10: Zerlegung der ParameterDump

```
Generating CellTypes based on current field values
Found 1 items.
Found 1 items.
Found 1 items.
Found 1 items.
Found 1 items.
Found 1 items.
Running CC3D
Generating directory to store data that is used to run the current simulation.
Creating folder for current simulation
TargetDir for this simulation run will be C:\Users\Station\Documents\moduro-automation-working-dir\SdCdbCdiInDa\1487492159789
Directory C:\Users\Station\Documents\moduro-automation-working-dir\SdCdbCdiInDa\1487492159789created: true
New target dir for this simulation: C:\Users\Station\Documents\moduro-automation-working-dir\SdCdbCdiInDa\1487492159789
Converting Model Setup to JSON.
Enter exportParameterDump()
EXporting ParameterDump File as Json to: C:\Users\Station\Documents\moduro-automation-working-dir\SdCdbCdiInDa\1487492159789
IMPORTANT. WE WILL USE THE DEFAULT CHARSET TO SAVE THE JSON STRING: UTF-8
ParameterDump has been exported.
Exit exportParameterDump()
Saving target configuration file to final output.
Generating CC3D File at: C:\Users\Station\Documents\moduro-automation-working-dir\SdCdbCdiInDa\1487492159789\runJson.cc3d
cc3d File saved: C:\Users\Station\Documents\moduro-automation-working-dir\SdCdbCdiInDa\1487492159789\runJson.cc3d
Generated cc3d file
Preperations to run simulation complete. Running Compucell
Enter getCompuCellProcess()
exit getCompuCellProcess()
Starting Compucell simulation, waiting until finished.
steppablePath=C:\Program Files (x86)\CompuCell3D\lib\CompuCell3DSteppables
```

Abbildung 6.11: Setup vor dem Starten des Simulationsprozesses

```
Compucell Simulation is done.
Exit runSimulation()
CC3D is done, will check working directory and parse FitnessPlot.dat
Found latest CC3D Workdir Simulation dir: C:\Users\Station\CC3DWorkspace\runJson_cc3d_02_19_2017_09_16_01
Loaded fitnessValues. Calculating values.
Calculating Fitness for this run...
Average fitness for this simulation is 0.8643517973163738
Final internal fitness for this Run is: 0.8643517973163738
Iterating internalFitnessFunction
Building new ParameterDump based on optimized values
Generating optimized ParameterDump, based on optimized Parameters
```

Abbildung 6.12: Berechnung der Fitness

```
Setting value of Field modelAdhEnergy from 2,000000 to 2,435409
Setting value of Field modelAdhFactor from 0,250000 to 0,368517
Setting value of Field cellStemGrothVolumePerDay from 78,539816 to 77,278841
Setting value of Field cellStemNecrosisProb from 0,000002 to 0,000000
Setting value of Field cellStemnutrientRequirement from 1,000000 to 1,107942
Setting value of Field cellStemSurFit from 0,500000 to 0,607182
Setting value of Field cellStemVolFit from 0,900000 to 0,693803
Setting value of Field cellBasalGrothVolumePerDay from 68,067841 to 65,500890
Setting value of Field cellBasalNecrosisProb from 0,000000 to 0,000000
Setting value of Field cellBasalnutrientRequirement from 1,000000 to 1,083930
Setting value of Field cellBasalSurFit from 0,500000 to 0,451060
Setting value of Field cellBasalVolFit from 0,900000 to 1,087775
Setting value of Field cellIntermediateGrothVolumePerDay from 194,386045 to 195,628620
Setting value of Field cellIntermediateNecrosisProb from 0,000000 to 0,000000
Setting value of Field cellIntermediatenutrientRequirement from 1,000000 to 0,909455
Setting value of Field cellIntermediateSurFit from 0,100000 to 0,154708
Setting value of Field cellIntermediateVolFit from 0,900000 to 0,895635
Setting value of Field cellUmbrellaGrothVolumePerDay from 359,136400 to 358,662935
Setting value of Field cellUmbrellaNecrosisProb from 0,000000 to 0,000000
Setting value of Field cellUmbrellaNutrientRequirement from 1,000000 to 1,149871
Setting value of Field cellUmbrellaSurFit from 0,100000 to 0,127174
Setting value of Field cellUmbrellaVolFit from 0,900000 to 1,108870
Generating CellTypes based on current field values
```

Abbildung 6.13: CMA-ES-Manipulation des ParameterDumps

6.5 Ergebnisse der Optimierung und Gestalt einer repräsentativen Fitnesskurve

Das Optimieren der Urothelsimulationen kann unterschiedliche Ausgänge haben. Viel hängt von den Einstellungen der Nebenbedingungen und der Definition der Fitness ab.

Die Fitness, die bei der Gestaltung einer Fitnesskurve genutzt wird, kann wie in Abbildung 6.14 verstanden werden.

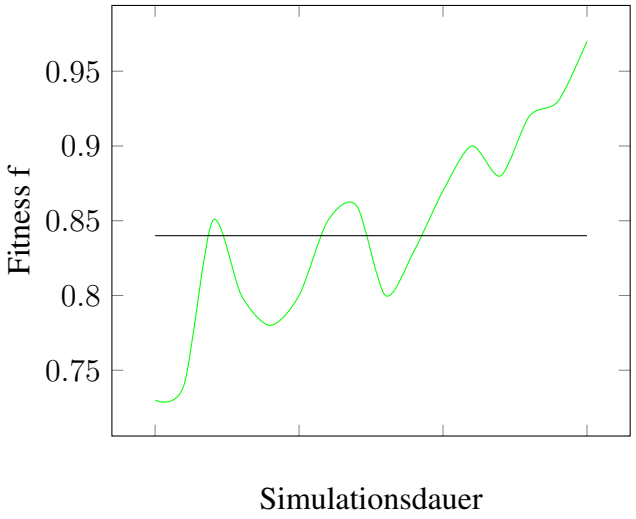


Abbildung 6.14: Schema der gemittelten Fitness

Eine Simulation läuft 720 simulierte Tage. An jedem Tag werden zwei Einträge dokumentiert, was heißt, dass ein FitnessPlot, der als Output referenziert wird, 1440 Einträge hat. Alle Einträge werden gemittelt, sodass der Mittelwert die mittlere Fitness einer Simulation darstellt.

Nach Ablauf jeder Simulation, also einer Iteration, wird der Mittelwert bewertet und vom Algorithmus für gut oder schlecht befunden. Entsprechend richtet die CMA-ES seine Parameter neu aus.

In der Methode zur Berechnung des Mittelwerts wird ebenfalls die Anzahl der Einträge im FitnessPlot beachtet. Der Mittelwert gibt lediglich Auskunft über die gemittelte Fitness und nicht über das Beenden einer Simulation. Dazu wird die Anzahl der Einträge überprüft und mit dem Mittelwert verrechnet. Sollte eine Simulation abbrechen, aber dennoch gute Fitnesswerte erlangen, kann somit ein „Strafabzug“ eingebunden werden. Je schneller eine Simulation abbricht, desto mehr fällt dieser

Strafabzug ins Gewicht. Die CMA-ES soll dadurch merken, dass diese Parametereinstellung trotz guter mittlerer Fitness zum Abbruch führt und diese mit einer schlechteren Fitness bewerten.

Ein Optimierungsdurchlauf endet immer mit dem manuellen Abbruch einer Simulation, der erreichten Evaluationszahl oder einem optimalen Fitnesswert.

Die Fitnesskurven in Abbildung 6.15 und 6.16 zeigen die bisher maximal erreichte Anzahl an Iterationen.

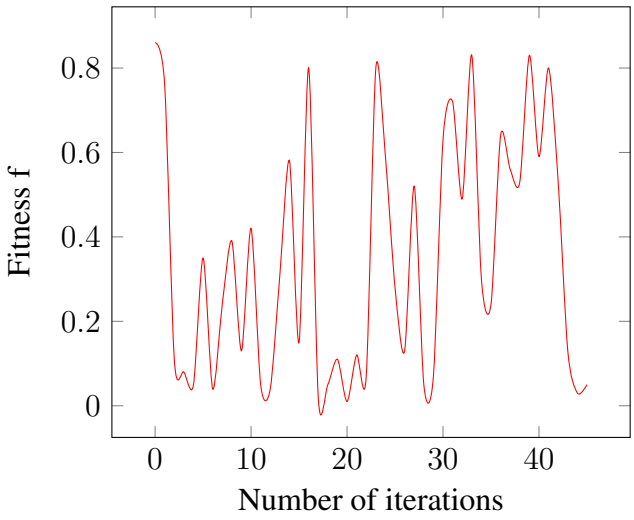


Abbildung 6.15: SdCdbCdiInDa: Fitness-Kurve

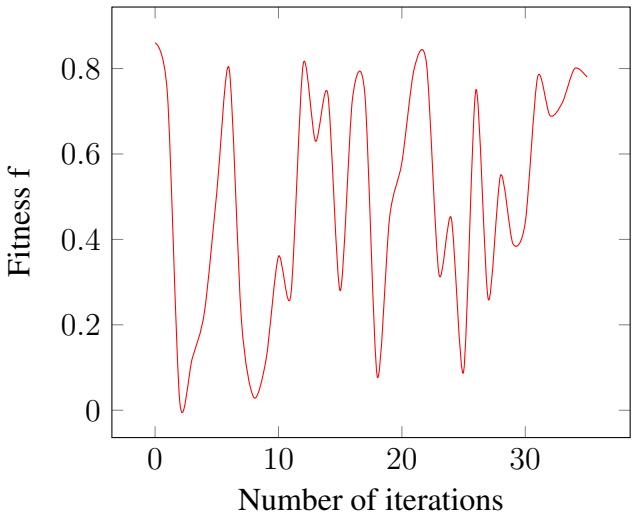


Abbildung 6.16: SpaSdbCdiInDa: Fitness-Kurve

Mit einer geeigneten Wahl der Nebenbedingungen und Parameter, kann auch die Anzahl der Iterationen erhöht werden.

Generell brechen die Simulationen sehr schnell ab, wenn keine Nebenbedingungen gesetzt werden. Die CMA-ES weiß nicht, dass das Setzen negativer Parameter zum Abbruch einer Simulation führt. Um dem vorzubeugen, müssen die *lowerBounds* auf 0 gesetzt werden. Vorteilhaft wäre es, einen biologisch sinnvollen Range für jeden einzelnen Parameter auszuloten. Das beugt nicht nur einem übereiltem Abbruch vor, sondern verkürzt möglicherweise auch die Anzahl der benötigten Iterationen pro Optimierungslauf. Eine weitere Beobachtung war die Bedeutung des sensiblen Parameters σ. Die Schrittweite σ definiert inwieweit ein Parameter verändert werden darf.

Ebenso wie bei den *Upper-* und *LowerBounds* gibt es eine Möglichkeit, über einen Vektor die Schrittweiten individuell zu vergeben. Betrachtet man die Werte der zu optimierenden Parameter, wird schnell klar, dass eine einheitliche Schrittweite keiner guten Lösung gleichkommt. Einige Parameter bewegen sich im Intervall zwischen 0 und 1 und benötigen eine lokale Suche, andere sind relativ. Eine Schrittweite von 1 kann eine Simulation in diesem Fall gänzlich umschlagen. Es reicht eine Iteration in diese Richtung und die Simulation bricht ab. Andere Parameter wie die Wachstumsrate bewegen sich im Hunderterbereich. Hier sind sehr viele Iterationen nötig, um ein Optimum zu finden. Das Runterschrauben der Schrittweite auf beispielsweise 0,1 würde zwar das Umschlagen der Simulationsgüte auf Anhieb verhindern, verlängert den Optimierungslauf aber wiederum um viele Iterationen. Deshalb sollte über das gegebene Sigma für jeden Parameter eine individuelle Schrittweite vergeben werden.

Der Verlauf der Fitnesskurve zeigt, dass die Güte der Simulation nach dem initialen Lauf abstürzt. Das liegt daran, dass die ursprünglich angenommenen Parameter schon recht gut sind. Viele Modelle erreichen bereits eine durchschnittliche Fitness von 0.8 oder 0.9. Die CMA-ES probiert sich aus, weshalb die Fitness in den nachfolgenden Iterationen einbricht. Allerdings ist zu erkennen, dass es eine leicht steigende Tendenz nach oben im Laufe der Iterationen gibt.

Wie bereits erwähnt, braucht schon allein die Rosenbrock-Funktion, die in Kapitel 4 referenziert wird, bis zu 6.000 Funktionsaufrufe, um ein Optimum zu finden. Es ist zu erwarten, dass Moduro ebenso viele für die gleiche Anzahl an Dimensionen braucht. Zudem dauern hier die Funktionsaufrufe nicht wenige Sekundenbruchteile, sondern drei bis fünf Stunden. Um ein perfektes Ergebnis zu erwarten, müsste man demnach bis zu 25.000 Stunden rechnen, was 1041 Tagen entspräche.

6.6 Medizinische Diskussion und Ausblick

Leider können nach der bisher erreichten Anzahl an Iterationen keine Schlussfolgerungen über die ausgefallenen Parameter getroffen werden. Wie oben beschrieben, braucht man mehr Rechenzeit, um vernünftige Auswertungen machen zu können.

Nichtsdestotrotz ist der Ausblick vielversprechend: Sollten die Modelle eine angemessene Rechenzeit zur Verfügung gestellt bekommen, steht dem Finden eines Optimums für jedes Modell nichts mehr im Wege.

Für das weitere Verwenden von ModuroOptimizationAutomation können allerdings noch weitere Verbesserungsansätze ausgearbeitet werden. Das Evaluieren der Optimierungsparameter ist möglicherweise noch nicht abgeschlossen. Es ist denkbar, dass das Drehen an den Parameter-Schrauben einen verbesserten Einfluss auf die Optimierungsläufe hat. Ein weiterer Punkt ist das Setzen des Intervalls für jeden einzelnen Parameter. Eine untere Grenze von 0 wurde initial für jeden Parameter gesetzt, was auch unabdingbar war. Zusätzlich ist die Obergrenze bewusst hoch gewählt worden, um den Parametern den Spielraum zu gewähren, den sie vielleicht brauchen. Die Ober- und Untergrenzen sollten für jeden Parameter nochmals individuell geprüft werden.

Als zusätzliche Funktionalität zur Auswertung einer Fitnesskurve, möglicherweise in der Moduro-Toolbox, wäre es vorteilhaft eine Anzeige über die gemittelte Fitness über die Anzahl an Iterationen zu implementieren. Bei der Auswertung der Daten hätte es sich als sehr praktisch erwiesen.

Ein weiterer wichtiger Punkt ist das Anpassen der Fitness. Die gemittelte Fitness entspricht der Fitness über die ganze Simulation hinweg. Womöglich wäre eine gemittelte Fitness über den Steady-State ab dem 150. Tag sinnvoller zum Auswerten. Die Beobachtungen aus den Samplings haben zusätzlich Vermutungen über den Range eines Parameters angeregt. Die Frage, ob einige Parameter ihr Verhalten nach Erreichen des Steady-State ändern, ist durchaus berechtigt und in der Biologie nicht ungewöhnlich.

Literaturverzeichnis

[1] ApacheCommons, 2016.

[2] Anne Auger and Nikolaus Hansen. *Tutorial: CMA-ES — Evolution Strategies and Covariance Matrix Adaptation*. INRIA Research Centre Saclay – Île-de-France Project Team TAO, July 2011.

[3] Immanuel Bomze and Wilfried Grossmann. *Optimierung - Theorie und Algorithmen: eine Einführung in Operation Research für Wirtschaftsinformatiker*. Biobliographisches Institut, F.A. Brockhaus AG, 1993.

[4] Elke Brechner, Barbara Dinkelaker, and Daniel Dreesmann. Ökologische potenz. *Kompaktlexikon der Biologie, Spektrum Akademischer Verlag, Heidelberg*, 2001.

[5] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Car setup optimization. competition software manual. Technical report, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, 2010.

[6] Julian Debatin. Studienarbeit-models. Technical report, Hochschule Mannheim, 2016.

[7] J Ferlay, F Bray, P Pisani, and DM Parkin. Globocan 2002: Cancer incidence. mortality and prevalence worldwide. *IARC Press*, IARC CancerBase No.5(2), 2004.

[8] Thomas Gasser. *Basiswissen Urologie*, volume 5. Auflage. Springer Verlag, 2011.

[9] Nikolaus Hansen. *The CMA Evolution Strategy: A Tutorial*. Research centre Saclay–Île-de-France Université Paris-Saclay, LRI, 2016.

[10] https://de.wikipedia.org/wiki/Computersimulation.

[11] https://www.lri.fr/ hansen/cmaes.

[12] Florian Jarre and Josef Stoer. *Optmierung*. Springer Verlag, 2004.

[13] Markus Kemmerling. Optimierung der fahrzeugabstimmung auf basis verrauschter daten einer autorennsimulation. Master's thesis, Technische Universität Dortmund, June 2010.

[14] Tamara Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some clasical and modern methods. *SIAM review*, 45(3):385–482, 2003.

[15] L. Jochen Krause. Parameteroptimierung bei der evolutionären entwicklung neuronaler topologien. Master's thesis, Christian-Albrechts-Universität Kiel, November 2007.

[16] Massimo Lazzeri. The physiological function of the urothelium–more than a simple barrier. *Urologia Internationalis*, 76(4):289–295, 2006.

[17] Thomas M. Lehmann and Erdmuthe Meyer zu Bexten. *Handbuch der medizinischen Informatik*. Carl Hanser Verlag München Wien, 2002.

[18] Diego G. Loyola, Mattia Pedergnana, and Sebastián Gimeno García. Smart sampling and incremental function learning for very large high dimensional data. *Neural Networks*, 78:75–87, 2016.

[19] Dirk Manski. *Urologielehrbuch.de*. Dr. Dirk Manski, 2015.

[20] Zbigniew Michalewicz and David Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, 2004.

[21] Mathuraa Pathmanathan. Entwurf und implementierung der moduro-toolbox zur analyse von urothelsimulationen. Technical report, Hochschule Mannheim, 2016.

[22] Howard H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3:175–184, 1960.

[23] Daniela Schultz-Lempel, Axel Haferkamp, and M. Goepel. *Urodynamik: Akademie der Deutschen Urologen*, volume 3. Auflage. Springer Verlag, 2012.

[24] Hildegard Speckmann and Cornelia Wittkowski. *Handbuch der Anatomie. Bau und Funktion des menschlichen Körpers*. Elsevier Verlag, München, 2012.

[25] Maciej H. Swat, Julio Belmonte, Randy W. Heiland, Benjamin L. Zaitlen, James A. Glazier, and Abbas Shirinifard. *Introduction to CompuCell3D*. Biocomplexity Institute and Department of Physics, Indiana University, 727 East 3rd Street, Bloomington IN, 47405-7105, USA.

[26] Angelo Torelli, Fabian Siegel, Philipp Erben, and Markus Gumbel. Modeling of the urothelium with an agent based approach. *Lecture Notes in Computer Science*, 9044:375–385, 2015.

Abbildungsverzeichnis