

# Optimierung der Anzahl der Ausbildungstellen im österreichischen Gesundheitssystem

Florian Bogner

Betreut von: Claire Rippinger & Christoph Urach

# Inhaltsverzeichnis

<b>1</b>	<b>Abstract</b>	<b>2</b>
<b>2</b>	<b>Einleitung</b>	<b>2</b>
2.1	Das Modell . . . . .	2
2.2	Die Aufgabenstellung . . . . .	2
2.3	Das vereinfachte Modell . . . . .	3
<b>3</b>	<b>Kalibrierungsansätze</b>	<b>4</b>
3.1	Partikelschwarm . . . . .	4
3.2	Sintflut . . . . .	5
3.3	Downhill Simplex . . . . .	6
3.4	Vergleich . . . . .	7
<b>4</b>	<b>Bewertungsfunktionen</b>	<b>8</b>
4.1	Die erste Bewertungsfunktion . . . . .	8
4.2	Die zweite Bewertungsfunktion . . . . .	9
4.3	Die dritte Bewertungsfunktion . . . . .	9
<b>5</b>	<b>Technische Details</b>	<b>10</b>
<b>6</b>	<b>Resultate</b>	<b>10</b>
<b>7</b>	<b>Fazit</b>	<b>10</b>

# 1 Abstract

TODO: scrivere l'abstracto (inglese)

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

## 2 Einleitung

In dem von dexhelpp geschaffenen Modell für die Laufbahn österreichischer Ärzte von Studienbeginn bis Pensionierung sollen bestimmte Parameter via Optimierungsalgorithmen kalibriert werden. Da das Modell sehr detailliert und komplex ist, dauert ein Simulationsdurchlauf mehrere Minuten. Deshalb ist es notwendig Algorithmen zu finden, die mit Simulationsauswertungen sehr sparsam umgehen.

### 2.1 Das Modell

Es handelt sich um ein agentenbasiertes Modell. Die Agenten repräsentieren ÄrztInnen an verschiedenen Stellen in ihrer Ausbildung oder ihrem Berufsleben. Sie beginnen als Studenten an einer der sieben Universitäten für Medizin in Österreich. Nach dem Studienabschluss beginnen die Agenten mit ihrem Turnus, entweder als Allgemeinmediziner (AM) oder Facharzt (FA). Absolventen der AM-Ausbildung entscheiden sich dann ob sie als AM berufstätig werden oder die FA-Ausbildung beginnen.

Die Ausbildung im Turnus und das Berufsleben ist aufgeteilt auf die neun Bundesländer und weiter aufgeteilt auf den Urbanisationsgrad (Städte, Kleinstädte und Vororte, Ländliche Gebiete). Die Agenten treffen während ihrer Laufbahn viele Entscheidungen wie z.B. Studienwechsel, Emigration, Wahl der Ausbildungsstelle, Berufswechsel, Vollzeit oder Teilzeit, etc.

Das ganze Modell hat hunderte Parameter, die zum größten Teil aus Expertenschätzungen bestehen oder aus historischen Daten extrapoliert wurden. Der Anfangszustand besteht aus aktuellen echten Daten. Das Modell wird dann bis zu einem festgelegtem Jahr durchgerechnet und Informationen über alle Agenten in Form einer Datenbank gespeichert. Aus dieser kann man dann alles auslesen, was man beobachten will.

### 2.2 Die Aufgabenstellung

Diese Seminararbeit beschäftigt sich mit der optimalen Wahl der Anzahl der Ausbildungsstellen, jeweils für Allgemeinmediziner und Fachärzte. Jedes Jahr, wenn ÄrztInnen in Pension gehen, werden offene Stellen hinterlassen, die idealerweise durch Berufseinsteiger gefüllt werden. Ziel ist es, möglichst genau so viele Absolventen wie offene Stellen in den jeweiligen Bereichen zu haben.

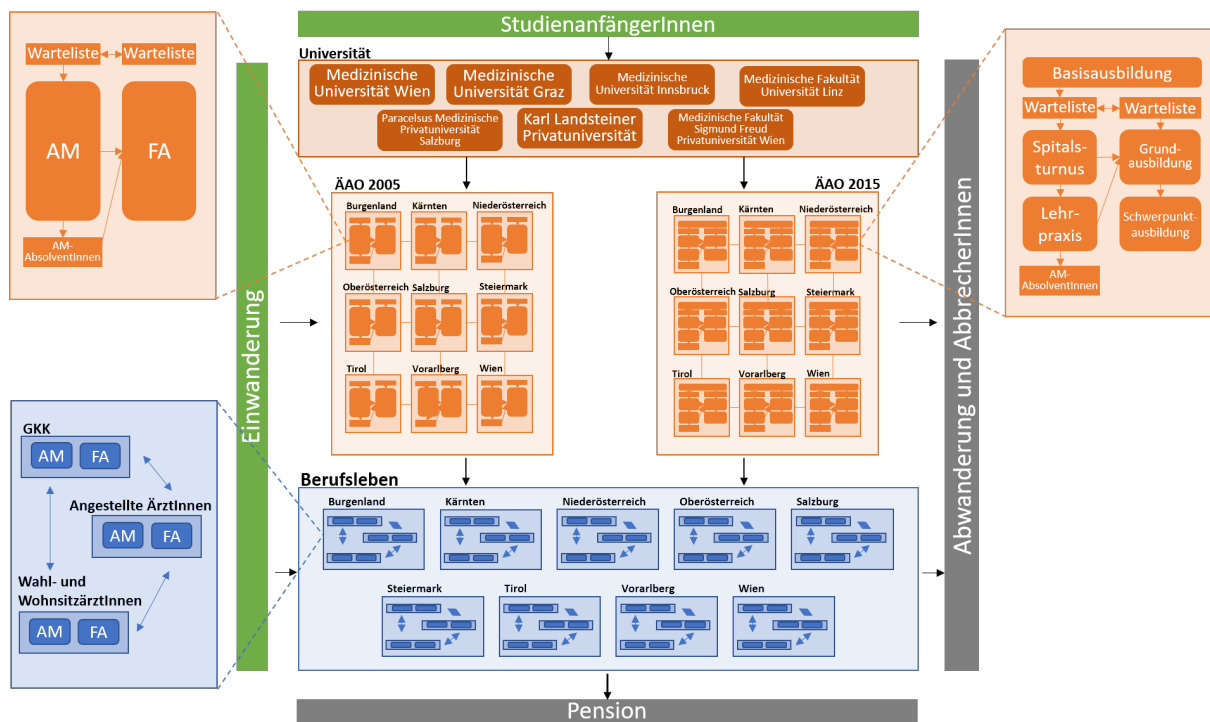


Abbildung 1: Karrierestationen der Agenten im Modell

## 2.3 Das vereinfachte Modell

Es wird keine Rücksicht auf das Bundesland, in dem ein angehender Arzt seine Ausbildung absolviert, sowie die Fachrichtung eines angehenden Facharztes genommen. Es werden nur die Anzahl der freien Stellen und Absolventen aller Fachrichtungen in ganz Österreich verglichen. Die Anzahl der Ausbildungsstellen werden für je drei Jahre konstant angenommen. Dies bewirkt eine Dimensionsreduktion im Parameterraum.

### 3 Kalibrierungsansätze

Das Problem verlangt nach Algorithmen, die nur mit Funktionsauswertungen auskommen und keine Gradienten oder Richtungsableitungen brauchen, da die Parameter des Modells ja natürliche Zahlen sind und numerische Gradientenbestimmung sowieso zu teuer wäre.

Die drei verwendeten Algorithmen haben einiges gemeinsam. Als gemeinsame Argumente haben sie eine Funktion  $f : \mathbb{R}^d \rightarrow \mathbb{R}$ , die Dimension  $d$  und eine Matrix  $bounds = \begin{pmatrix} a_1, \dots, a_d \\ b_1, \dots, b_d \end{pmatrix} \in \mathbb{R}^{2 \times d}$ , die einen Quader  $\times_{i=1}^d [a_i, b_i] \subset \mathbb{R}^d$  beschreibt. Aus diesem Quader werden die Startwerte gewählt. Das gesuchte Minimum sollte in diesem Quader sein.

Als Output haben die Algorithmen (natürlicherweise) den Input für die Funktion  $f$  wo das Minimum vermutet ist.

Durch die unmittelbare Anwendungsnähe ist Echtzeit-Laufzeit das wichtigste Bewertungskriterium. Durch die moderne Prozessorarchitektur mit mehreren Kernen betrachten wir auch die Parallelisierbarkeit der Algorithmen. Da die Verwendung von mehreren Kernen in einem Simulationsdurchlauf nicht einfach möglich ist, bzw. nicht im Scope dieser Seminararbeit liegt, müssen wir uns mit der parallelen Ausführung von mehreren Simulationsdurchläufen begnügen.

#### 3.1 Partikelschwarm

Dieser Algorithmus basiert auf einem Schwarm von Partikeln die sich durch den Parameterraum bewegen. Zu Beginn des Algorithmus werden die Positionen von einer Anzahl an Partikeln innerhalb der *bounds* sowie deren Geschwindigkeit zufällig gewählt. In jedem Schritt wird die Geschwindigkeitsänderung, also eine Beschleunigung ausgerechnet. Diese zieht die Partikel zum Teil zu dem bisher besten Punkt, den der jeweilige Partikel erreicht hat und zum Teil zum bisher besten Punkt überhaupt. Zur Geschwindigkeit wird dann jeweils die Beschleunigung addiert und zur Position die Geschwindigkeit.

---

**Algorithm 1** Partikelschwarm

---

```
function FINDMINIMUM( $f, d, bounds, numOfParticles, maxV, acc$ )  
   $I := \{1, 2, \dots, numOfParticles\}$   
   $(pos_i)_{i \in I}$  = Vektor von zufällig gewählten Positionen innerhalb der bounds  
   $(vel_i)_{i \in I}$  = Vektor von zufällig gewählten Richtungen  
   $pBest_i = \infty$   
   $gBest = 1$   
  
  while do  
    end while  
end function
```

---

Der Partikelschwarm-Algorithmus eignet sich hervorragend zu Parallelisierung. Die Partikel bewegen sich alle gemeinsam in einem Zeitschritt und deren neue Position muss danach gleichzeitig ausgewertet werden. Idealerweise wählt man dann auch noch die Anzahl der Partikel als ein Vielfaches der Anzahl der verfügbaren Prozessorkerne um alle Kerne immer zu nutzen.

## 3.2 Sintflut

Der Sintflut-Algorithmus ist eine Verbesserung des naiven Hill-Climbing-Algorithmus, welcher an einem zufälligen Punkt im Parameterraum startet und dann iterativ zufällige Schritte macht. Bei einer Verbesserung wird der neue Punkt angenommen, sonst bleibt man beim alten Punkt. Eine große Schwäche des Hill-Climbing-Algorithmus ist das leichte Verfangen in lokalen Optima. Der Sintflut-Algorithmus verspricht diese Schwäche zu korrigieren.

Bildlich gesprochen befindet sich ein herumirrender Wanderer in einer Landschaft mit Hügeln und Tälern.<sup>1</sup> Es regnet konstant und so steigt der Wasserspiegel immer weiter an. Der Wanderer, anders als im Hill-Climbing-Algorithmus kann jetzt auch bergab gehen, aber er kann nicht schwimmen. Wenn der Wanderer schließlich keinen Schritt mehr machen kann, ohne nasse Füße zu bekommen, muss er wohl einen Gipfel erreicht haben. Durch die Möglichkeit bergab zugehen erhofft man sich die gröbere Toleranz gegenüber lokalen Optima.

---

**Algorithm 2** Sintflut

---

**function** FINDMINIMUM( $f, d, bounds, Sealevel, Rain, Schrittweite, (optional)Startparameter$ )

**end function**

---

Die Wahl des Startwasserstands und der Regenmenge ist von großer Bedeutung. Bei zu tiefen Wasserstand wandert man unbeschränkt herum und verschwendet effektiv Rechenzeit, oder schlimmer sogar wandert weit weg vom (z.B. als gute Schätzung gewählten) Startwert. Bei zu hohem Wasserstand startet der Wanderer vielleicht schon in mitten eines Ozeans, weit weg von Land. Der Algorithmus bricht dann schnell ohne Ergebnis ab.

Die Regenmenge kontrolliert quasi die Konvergenzrate. Bei zu viel Regen kann der Wanderer vielleicht nicht schnell genug auf einen Berg flüchten. Je weniger Regen, desto länger dauert die Ausführung des Algorithmus, doch die Genauigkeit und die Konfidenz in das Ergebnis steigt.

In der originalen Quelle<sup>2</sup> ist nur von einer kleinen stochastischen Änderung die Rede. Welche Verteilung diese haben soll wird nicht näher spezifiziert. Eine naheliegende Möglichkeit ist die Normalverteilung mit  $\mu = 0$  und  $\sigma$  als kleinen Wert der als Schrittweitenparameter betrachtet werden kann.<sup>3</sup> Eine andere Möglichkeit ist eine feste Schrittweite in eine zufällige Richtung zu gehen. Letztlich wurde eine Affinkombination dieser beiden gewählt.

Durch die einzelnen Schritte eignet sich dieser Algorithmus nicht besonders gut zur Beschleunigung durch Parallelisierung. Man kann spekulativ mehrere mögliche Schritte zugleich auswerten und, falls der erste ins Wasser steigt, den zweiten überprüfen und so weiter. Man erkaufte sich dadurch ein bisschen Beschleunigung, aber viele der Auswertungen fließen gar nicht in den Algorithmus ein. Mit steigender Prozessorkernanzahl trifft man schnell auf diminishing returns.

---

<sup>1</sup>Für die bildliche Vorstellung suchen wir jetzt das Maximum, nicht das Minimum wie oben angegeben

<sup>2</sup>TODO: Ordentliche Quellenangaben und Zitate

<sup>3</sup>Der Wanderer ist dann ein Wiener.

### 3.3 Downhill Simplex

Dieser Algorithmus basiert auf einem Simplex im Parameterraum, also der konvexen Hülle von  $d + 1$  vielen Punkten. Die Eckpunkte werden ausgewertet und in jedem Schritt wird der schlechteste Eckpunkt durch einen besseren ersetzt. Der Simplex bewegt sich dadurch im Laufe der Zeit dem figurativen Hügel hinab einem lokalen Minimum entgegen und zieht sich letztendlich über dem Minimum zusammen.

---

**Algorithm 3** Downhill Simplex

---

```

function FINDMINIMUM( $f, d, bounds, \alpha = 1, \gamma = 2, \beta = \frac{1}{2}, \sigma = \frac{1}{2}$ )
   $x_i$  für  $i \in \mathbb{N}_{\leq d}$  werden zufällig innerhalb der  $bounds$  gewählt
  for  $k \in \mathbb{N}_{<totalSteps}$  do
    sortiere  $x_i$  sodass  $f(x_0) < f(x_1) < \dots < f(x_d)$ 
     $m := \frac{1}{d} \sum_{i=0}^{d-1} x_i$  ▷ der Mittelpunkt aller Ecken außer der schlechtesten
     $r := (1 + \alpha)m - \alpha x_d$  ▷ Reflexion von  $x_d$  an  $m$ 
    if  $f(r) < f(x_0)$  then
       $e := (1 + \gamma)m - \gamma x_d$  ▷ Der Expandierte Punkt
      if  $f(e) < f(r)$  then
         $x_d := e$ 
      else
         $x_d := r$ 
      end if
    else if  $f(r) < f(x_{d-1})$  then
       $x_d := r$ 
    else
      if  $f(r) < f(x_N)$  then
         $h := r$ 
      else
         $h := x_d$ 
      end if
       $c := \beta m + (1 - \beta)h$ 
      if  $f(c) < f(x_d)$  then ▷ Der Kontrahierte Punkt
         $x_d := c$ 
      else
         $x_i := \sigma x_0 + (1 - \sigma)x_i \forall i \neq 0$  ▷ Komprimiere den Simplex
      end if
    end if
  end for
end function

```

---

Im Pseudocode wurde  $f$  hier oftmals zur Klarheit doppelt und dreifach aufgerufen. In der tatsächlichen Implementation wird  $f$  für jeden Punkt natürlich nur einmal aufgerufen und das Ergebnis gespeichert.

In jedem Schritt gibt es zwei Möglichkeiten, entweder der schlechteste Punkt wird ersetzt oder der Simplex wird komprimiert. Im ersten Fall gibt es vier Kandidaten für die neue Stelle des Eckpunkts, nämlich  $r$ ,  $e$  und zwei mal  $c$ , je nachdem ob  $h$  als  $r$  oder  $x_d$

gewählt wird. Im Sinne der Parallelisierung können nun die 4 Möglichkeiten gleichzeitig ausgewertet werden und man erkaufte sich eine bis zu dreifache Beschleunigung.

Beim ersten Auswerten der Eckpunkte zu Beginn und beim Kontrahieren des Simplex kann natürlich auch parallelisiert werden.

### **3.4 Vergleich**



## 4 Bewertungsfunktionen

Die folgenden Variablen sind allesamt Vektoren, die die jeweilige Größe für Allgemeinmediziner ( $AM$ ) und Fachärzte ( $FA$ ) über die Simulationsperiode  $J$  darstellen.

$AM_a$	$FA_a$	gesamte Ausbildungsstellen	Modellparameter
$AM_o$	$FA_o$	offen gebliebene Ausbildungsstellen	Modellresultat
$AM_f$	$FA_f$	Berufseinsteiger (f wie fertig)	Modellresultat
$AM_p$	$FA_p$	pensionierte Ärzte	vorgegebener Zielwert

### 4.1 Die erste Bewertungsfunktion

Die erste Bewertungsfunktion ist:

$$\begin{aligned}
 F_1(\dots) = & c_1 \cdot \left\| \frac{(AM_f - AM_p)}{AM_p} \cdot (1, \dots, 0.5) \right\|_2 + \\
 & c_2 \cdot \left\| \frac{(FA_f - FA_p)}{FA_p} \cdot (1, \dots, 0.5) \right\|_2 + \\
 & c_3 \cdot \left\| \left( \frac{\sum_{j=1}^i AM_{f,j} - AM_{p,j}}{\sum_{j=1}^i AM_{p,j}} \right)_{i \in J} \right\|_2 + \\
 & c_4 \cdot \left\| \left( \frac{\sum_{j=1}^i FA_{f,j} - FA_{p,j}}{\sum_{j=1}^i FA_{p,j}} \right)_{i \in J} \right\|_2 + \\
 & c_5 \cdot \left\| \frac{AM_o}{AM_a} \right\|_2 + \\
 & c_6 \cdot \left\| \frac{FA_o}{FA_a} \right\|_2
 \end{aligned}$$

wobei die Divisionen und die Multiplikationen als komponentenweise zu verstehen sind. Die ersten beiden Terme bestrafen Abweichungen zum Zielwert, also die Diskrepanz zwischen Berufseinsteigern und Pensionierten. Dies ist mit einer Diskontierung versehen, um Abweichungen in naher Zukunft stärker zu gewichten wie Abweichungen gegen Ende der untersuchten Periode.

Die zweiten zwei Terme bestrafen kumulierten Fehler. Betrachten wir folgende Beispiele: 1) In einem Jahr gibt es 200 Anfänger zu wenig, im darauffolgenden Jahr 200 zu viel. 2) In beiden Jahren sind 200 Anfänger zu wenig. Beide Beispiele werden von den ersten beiden Termen gleich bewertet, doch in der Realität ist das erste Beispiel wünschenswerter, da die Lücken vom ersten Jahr ja im Nächsten gefüllt werden.

Die dritten zwei Komponenten bestrafen offene Ausbildungsstellen. Dies ist offensichtlich davon motiviert, keine Ressourcen zu verschwenden.

Die Koeffizienten  $\vec{c}$  wurden anfänglich als  $(1, 1, 1, 1, 2, 2)$  gewählt und dies scheint keine schlechte Wahl gewesen zu sein.

## 4.2 Die zweite Bewertungsfunktion

Die zweite Bewertungsfunktion ist eine Erweiterung der ersten:

$$F_2(\dots) = F_1(\dots) + c_7 \cdot \sum_{i \in J \setminus \{1\}} \left( \frac{AM_{a,i-1} - AM_{a,i}}{1000} \right)^4 + c_8 \cdot \sum_{i \in J \setminus \{1\}} \left( \frac{FA_{a,i-1} - FA_{a,i}}{1000} \right)^4$$

Nach ein paar Testläufen ist aufgefallen, dass immer wieder Lösungen gefunden werden, in denen die Anzahl der Ausbildungsstellen stark schwingt. Dies ist natürlich unrealistisch, da zum Beispiel viele Ausbildungsplätze schaffen, nur um den Großteil davon in 3 Jahren wieder abzuschaffen ist eine Verschwendung von Ressourcen. Deshalb wurde die erste Fehlerfunktion um weitere zwei Terme erweitert, die große Sprünge bestraft.

Der Koeffizienten  $\vec{c}$  wurden auf  $(1, 1, 1, 1, 2, 2, 2, 2)$  erweitert.

## 4.3 Die dritte Bewertungsfunktion

Im Laufe des Testens ist aufgefallen, dass manchmal im Laufe eines Downhill-Simplex-Algorithmus die Anzahl der Ausbildungsstellen divergieren.<sup>4</sup> Nachdem lange erfolglos ein Fehler im Algorithmus gesucht wurde, wurde klar, dass der Fehler in der Bewertungsfunktion liegt. In der fünften und sechsten Komponente steht der Anteil an unbesetzten Ausbildungsstellen. Ob dieser nun 99% oder 99.9% ist, macht nur ein winzigen Unterschied für die Bewertungsfunktion, aber entspricht (bei gleich vielen Auszubildenden) einer zehnfachen Erhöhung der Ausbildungsstellen. Zur Korrektur wird auf die betreffenden Terme eine Funktion angewandt, die in der Nähe der 0 ähnlich wie die Identität aussieht<sup>5</sup> und bei 1 gegen  $\infty$  konvergiert.

$$f : [0, 1) \rightarrow [0, \infty) : x \mapsto \frac{x}{1 - x}$$

erfüllt diese Eigenschaften und die dritte Bewertungsfunktion lautet somit:

$$F_3(\dots) = \dots + c_5 \cdot \left\| f \left( \frac{AM_o}{AM_a} \right) \right\|_2 + c_6 \cdot \left\| f \left( \frac{FA_o}{FA_a} \right) \right\|_2 + \dots$$

---

<sup>4</sup>Die vermeintlichen Lösungen entsprechen freiem Zugang zu Ausbildungsstellen, welcher vielleicht die beste Lösung zum Problem des Ärztemangels wäre.

<sup>5</sup>Diese Bedingung sorgt dafür, dass  $F_2 \leq F_3$  immer und  $F_2 \approx F_3$  bei kleinen Anteilen der unbesetzten Ausbildungsstellen.

## 5 Technische Details

Die gesamte Codebase ist in Python 3 geschrieben. Das Modell ist als importierte Funktion aufrufbar. Da die Signatur dieser Funktion ein *dict* ist, welches Einträge für je Allgemeinmediziner und Fachärzte hat, die Algorithmen aber ein einzelnes *numpy*-Array erwarten wurde ein Wrapper für das Modell implementiert, der Argumente übersetzt. Dieser wendet auch gleich die Bewertungsfunktion auf das Modellresultat an. Auf diese Weise konnten die Algorithmen komplett modellagnostisch implementiert werden.

Zusätzlich schreibt der Wrapper alle Funktionsaufrufe in einer *.csv* Datei mit. Es werden die Parameter, die Modellergebnisse, der Wert der Bewertungsfunktion sowie die einzelnen Komponenten der Bewertungsfunktion gespeichert. Dies ermöglicht detaillierte Analyse des Konvergenzverhaltens.

## 6 Resultate

## 7 Fazit