Optimierung der Anzahl der Ausbildungstellen im österreichischen Gesundheitssystem

Florian Bogner
Betreut von: Claire Rippinger & Christoph Urach

Inhaltsverzeichnis

1 Abstract

We attempt to calibrate a model of the austrian healthcare system. Our goal is to make sure retiring doctors are replaced by freshmen in equal numbers. To achive this we test three algorithms, namly Particle Swarm, Great Deluge and Downhill Simplex Optimization. We identify a function to judge model outputs and optimize based on that. Experiments show that the Particle Swarm is the best of out approaches with Downhill Simplex as close second. Great Deluge instead fails to yield satisfing results.

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.

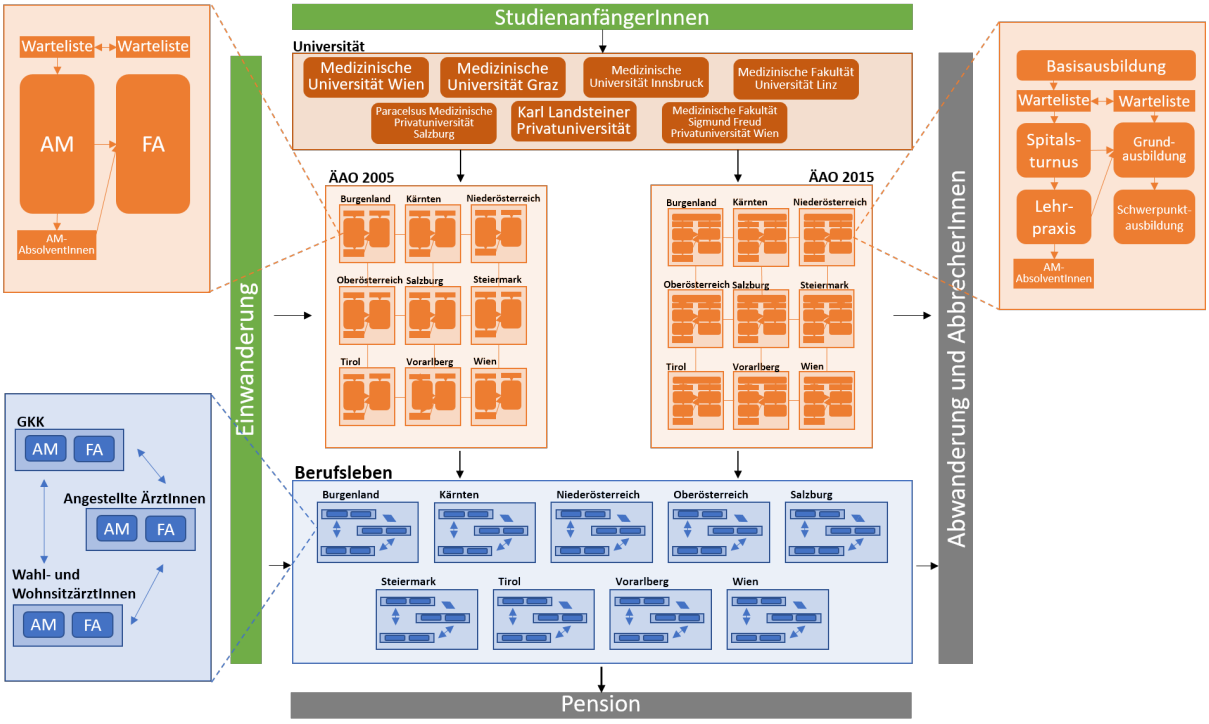


Abbildung 1: Karrierestationen der Agenten im Modell

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. Optional wird die Anzahl der Ausbildungsstellen für je drei Jahre zusammengefasst. Dies bewirkt eine Dimensionsreduktion im Parameterraum.

Sei also $J := \{2016, 2017, \ldots, 2030\}$ der Simulationszeitraum. Sei $J' := \{2019, 2020, \ldots, 2030\}$ der Outputzeitraum[1]. Formal gesehen ist das Modell dann eine Funktion $\mathcal{M}$:

$$\mathcal{M} : \mathbb{N}^J \times \mathbb{N}^J \to \mathbb{N}^{J'} \times \mathbb{N}^{J'} \times \mathbb{N}^J \times \mathbb{N}^J$$
$$AM_a, FA_a \mapsto AM_f, FA_f, AM_o, FA_o$$

mit folgender Notation:

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

$AM_p = (683, 553, 546, 519, 501, 359, 307, 291, 269, 252, 239, 211)$
$FA_p = (1082, 856, 880, 956, 978, 812, 761, 762, 752, 723, 680, 635)$

[1]Gibts da einen besseren Namen für das? Außerdem, stimmen diese Jahreszahlen eh? Ich hab sie aus meinen Plots abgelesen, aber da stehen sie im Code auch einfach nur hardcoded drinnen und ich hab keine Ahnung mehr wo ich sie eigentlich her hab.

3 Bewertungsfunktionen

Um Optimierungsalgorithmen anwenden zu können, braucht man eine Möglichkeit, zwei Modellresultate zu vergleichen. Wir tun dies mittels einer Bewertungsfunktion $\mathcal{F}$:

$$\mathcal{F} : \mathbb{N}^J \times \mathbb{N}^J \times \mathbb{N}^{J'} \times \mathbb{N}^{J'} \times \mathbb{N}^J \times \mathbb{N}^J \times \mathbb{N}^{J'} \times \mathbb{N}^{J'} \to \mathbb{R}$$
$$AM_a, FA_a, AM_f, FA_f, AM_o, FA_o, AM_p, FA_p \mapsto \mathcal{F}(\ldots)$$

3.1 Die erste Bewertungsfunktion

Die erste Bewertungsfunktion ist:

$$\begin{aligned}
\mathcal{F}_1(\ldots) = c_1 \cdot & \left\| \frac{(AM_f - AM_p)}{AM_p} \cdot (1, \cdots, 0.5) \right\|_2 + \\
c_2 \cdot & \left\| \frac{(FA_f - FA_p)}{FA_p} \cdot (1, \cdots, 0.5) \right\|_2 + \\
c_3 \cdot & \left\| \left(\frac{\sum_{j=2018}^{i} AM_{f,j} - AM_{p,j}}{\sum_{j=1}^{i} AM_{p,j}} \right)_{i \in J'} \right\|_2 + \\
c_4 \cdot & \left\| \left(\frac{\sum_{j=2018}^{i} FA_{f,j} - FA_{p,j}}{\sum_{j=1}^{i} FA_{p,j}} \right)_{i \in J'} \right\|_2 + \\
c_5 \cdot & \left\| \frac{AM_o}{AM_a} \right\|_2 + \\
c_6 \cdot & \left\| \frac{FA_o}{FA_a} \right\|_2
\end{aligned}$$

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.

3.2 Die zweite Bewertungsfunktion

Die zweite Bewertungsfunktion ist eine Erweiterung der ersten:

$$\mathcal{F}_2(\ldots) = \mathcal{F}_1(\ldots) + c_7 \cdot \sum_{i \in J \setminus \{2016\}} \left(\frac{AM_{a,i-1} - AM_{a,i}}{1000} \right)^4 + c_8 \cdot \sum_{i \in J \setminus \{2016\}} \left(\frac{FA_{a,i-1} - FA_{a,i}}{1000} \right)^4$$

Nach ein paar Testläufen ist aufgefallen, das 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.

3.3 Die dritte Bewertungsfunktion

Im Laufe des Testens ist aufgefallen, dass manchmal im Laufe eines Downhill-Simplex-Algorithmus die Anzahl der Ausbildungsstellen divergieren.[2] 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 ausschaut [3] und bei 1 gegen ∞ konvergiert.

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

erfüllt diese Eigenschaften und die dritte Bewertungsfunktion lautet somit:

$$\mathcal{F}_3(\ldots) = \ldots + c_5 \cdot \left\| f\left(\frac{AM_o}{AM_a} \right) \right\|_2 + c_6 \cdot \left\| f\left(\frac{FA_o}{FA_a} \right) \right\|_2 + \ldots$$

[2] Die vermeintlichen Lösungen entsprechen freiem Zugang zu Ausbildungsstellen, welcher vielleicht die beste Lösung zum Problem des Ärztemangels wäre.

[3] Diese Bedingung sorgt dafür, dass $\mathcal{F}_2 \leq \mathcal{F}_3$ immer und $\mathcal{F}_2 \approx \mathcal{F}_3$ bei kleinen Anteilen der unbesetzten Ausbildungsstellen.

4 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 $f : \mathbb{R}^d \to \mathbb{R}$, die Dimension d und eine Matrix $bounds = \binom{a_1,\cdots,a_d}{b_1,\cdots,b_d} \in \mathbb{R}^{2\times d}$, die einen Quader $\bigtimes_{i=1}^{d}[a_i, b_i] \subset \mathbb{R}^d$ 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.

4.1 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($f, d, bounds, numOfParticles, maxVel, acc$)
  $I := \{1, 2, \ldots, numOfParticles\}$
  for $i \in I$ **do** ▷ für jeden Partikel
    initialisiere pos_i als zufällig gewählte Position innerhalb der *bounds*
    initialisiere vel_i = als zufällig gewählte Richtung
    $pBest_i = \infty$ ▷ der "persönlich"beste Wert
    initialisiere $pBestPos_i$
  end for
  $gBest = \infty$ ▷ bester Wert "global"
  initialisiere gBestPos

  for $k \in \mathbb{N}_{<totalSteps}$ **do**
    for $i \in I$ **do**
      if $f(pos_i) < pBest_i$ **then**
        $pBest_i = f(pos_i)$
        $pBestPos_i = pos_i$
        if $pBest_i < gBest$ **then**
          $gBest = pBest_i$
          $gBestPos = pBest_i$
        end if
      end if
    end for

    for $i \in I$ **do**
      $vel_i = vel_i + acc \cdot rand() \cdot (pBestPos_i - pos)$
      $vel_i = vel_i + acc \cdot rand() \cdot (gBestPos - pos)$
       ▷ rand() generiert auf $[0, 1]$ uniform verteile unabhängige Zufallsvariablen.
      if $||vel_i|| > maxVel$ **then**
        $vel_i = maxVel \cdot \frac{vel_i}{||vel_i||}$
      end if
      $pos_i = pos_i + vel_i$
    end for
  end for
   return $gBestPos$
end function

4.2 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.[4] 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($f, d, bounds, initialLevel, deltaLevel, Schrittweite, (optional)Startparamet$
  initialisiere x innerhalb der *bounds*
  $level = initialLevel$
  for $k \in \mathbb{N}_{<totalIterations}$ **do**
   $dx \sim N(0, stepsize)$
    if **then**$f(x + dx) < level$
      $x = x + dx$
      $level = level - deltaLevel$
    end if
  end for
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[5] 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 σ als kleinen Wert der als Schrittweitenparameter betrachtet werden kann.[6] Eine andere Möglichkeit ist eine feste Schrittweite in eine zufällige Richtung zu gehen. Letztlich wurde eine Affinkombination dieser beiden gewählt.

4.3 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.

[4] Für die bildliche Vorstellung suchen wir jetzt das Maximum, nicht das Minimum wie oben angegeben
[5] TODO: Ordentliche Quellenangaben und Zitate
[6] Der Wanderer ist dann ein Wiener.

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 $f(x_0) < f(x_1) < \ldots < f(x_d)$
    $m := \frac{1}{d} \sum_{i=0}^{d-1} x_i$ ▷ der Mittelpunkt aller Ecken außer der schlechtesten
    $r := (1 + \alpha)m - \alpha x_d$ ▷ Der Reflektierte Punkt
    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**
        $x_d := c$ ▷ Der Kontrahierte Punkt
      else
        $x_i := \sigma x_0 + (1 - \sigma)x_i \forall i \neq 0$ ▷ Komprimiere den Simplex
      end if
    end if
  end for
  return x_0
end function

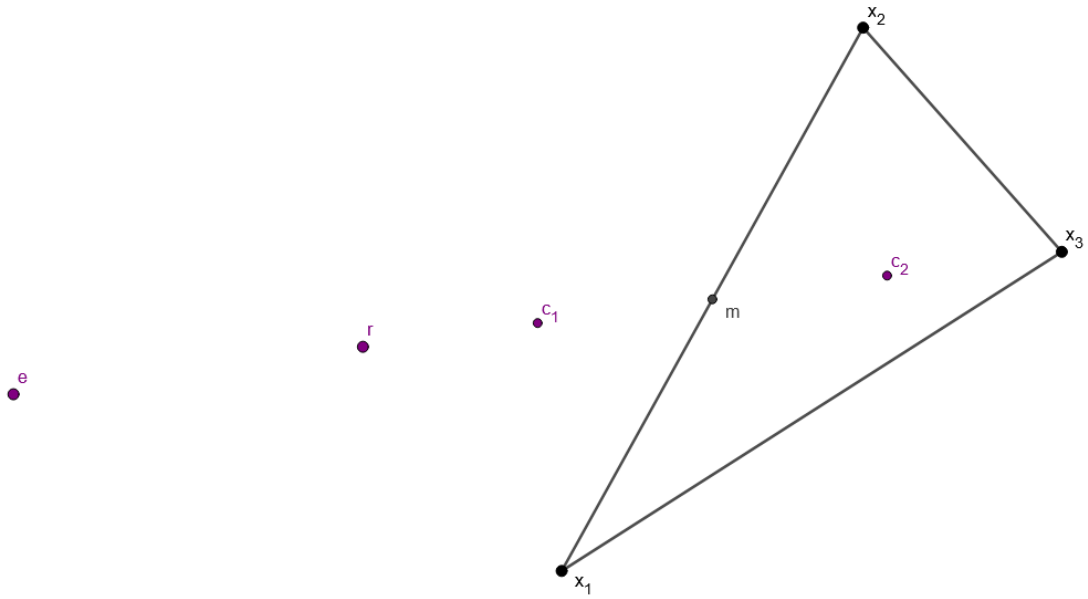


Abbildung 2: Simplex in der Ebene, sowie die vier neuen Möglichkeiten für x_3

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.

4.4 Vergleich

Die drei Ansätze profitieren unterschiedlich von der Verfügbarkeit von vielen Prozessorkernen.

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. Bei vielen Kernen kann man die Anzahl der Partikel einfach der Anzahl der Kerne gleichsetzen. So verwendet man nahezu 100% der verfügbaren Rechenleistung und verwendet außerdem alle ausgewerteten Stellen sinnvoll im weiteren Verlauf.

Der Sintflut-Algorithmus profitiert hingegen sehr wenig von Parallelisierung. Man kann spekulativ mehrere mögliche Schritte zugleich auswerten und falls der erste ins Wasser steigt, den zweiten überprüfen und so weiter. Man erkauft 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*.

Für den Downhill Simplex-Algorithmus gibt es in jedem Schritt zwei Möglichkeiten, entweder der schlechteste Punkt x_d 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. Diese vier Möglichkeiten können gleichzeitig ausgewertet werden und man erkauft sich eine bis zu dreifache Beschleunigung.

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

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 können die Algorithmen komplett modellagnostisch implementiert werden.

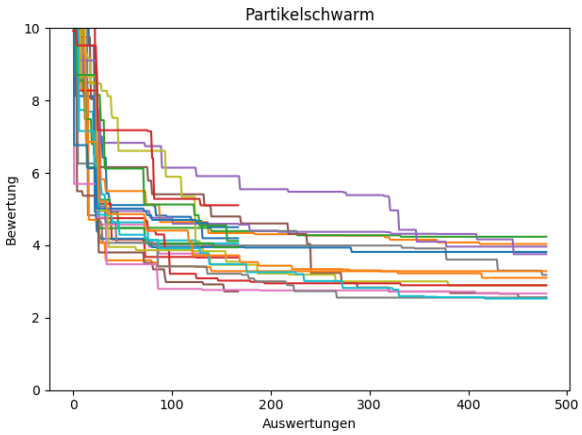
Zusätzlich schreibt der Wrapper alle Funktionsaufrufe einer Ausführung in einer *.csv* Datei mit. Es werden die Parameter, die Modellergebnisse, der Wert der Bewertungsfunktion sowie die einzelnen Komponenten der Bewertungsfunktion gespeichert. Diese Logs ermöglicht detaillierte Analyse des Konvergenzverhaltens.

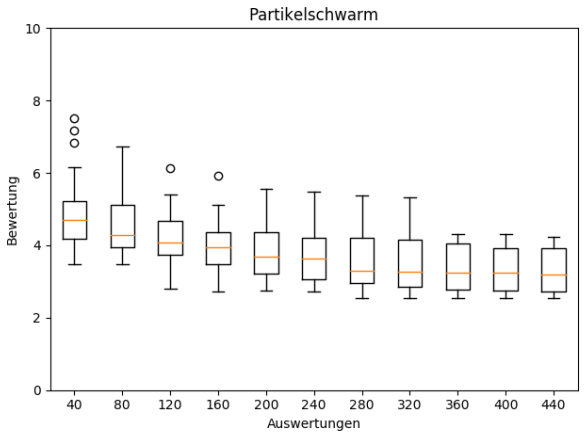
6 Resultate

6.1 Experiment 1

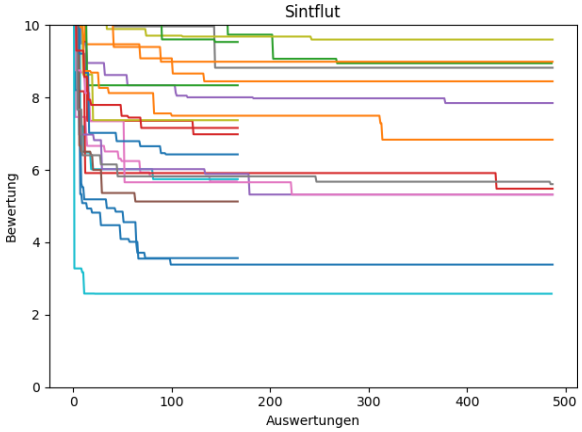
Wir führen die drei Algorithmen mehrmals durch und beobachten Konvergenzverhalten und erzielte Werte. Die Parameter werden für je 3 Jahre konstant gehalten. Damit ist die Dimension $d = 10$. Die Startwerte werden aus $bounds = [0, 1500]^d$ gewählt. Wir erlauben 160 bzw. 480 Auswertungen. Dies entspricht etwa 2 bzw. 6 Stunden Rechenzeit auf meinem 8-kernigen Prozessor.

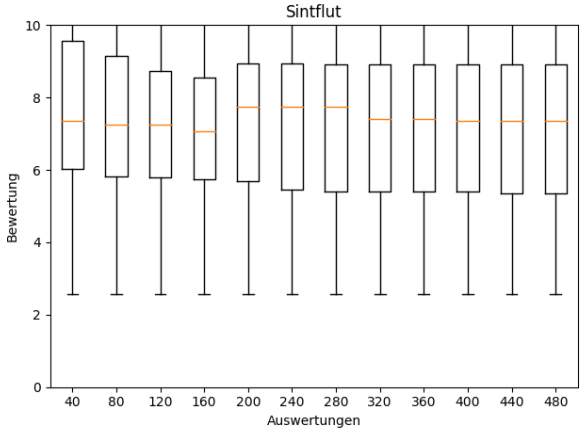
Um das Konvergenzverhalten zu beobachten betrachten wir die Spalte für die Bewertungsfunktion in den Logs. Sei e_i die Bewertung der i-ten Modellauswertung, also der i-ten Zeile im Log. Wir betrachten $m_i := min\{e_k : k \leq i\}$. Dies ist die Bewertung, wenn man den Algorithmus nach i Modellauswertungen abbrechen würde. Die Folge der m_i ist durch Definition monoton fallend und flacht ab wenn ein (zumindest lokales) Minimum gefunden wird.



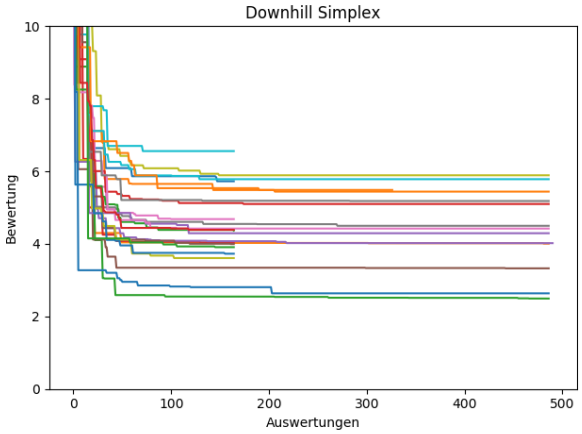


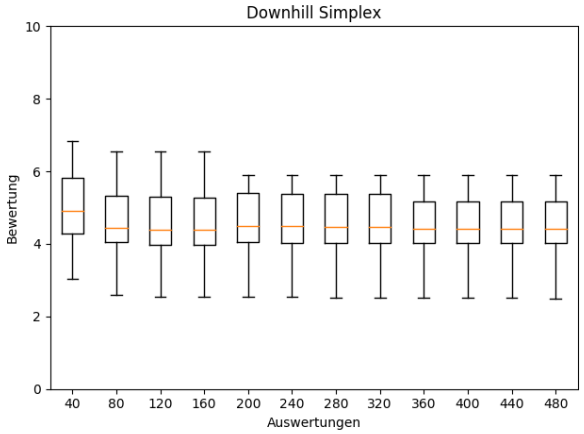
Beim Partikelschwarm sieht man, dass stetige Verbesserungen auch noch spät gefunden werden. Man kann vermuten, dass mehr Auswertungen durchaus zu mehr besseren Ergebnissen führen würden. Nur 160 Auswertungen sind definitiv zu wenig für den Partikelschwarm.





Die Bewertungen beim Sintflut-Algorithmus sind kreuz und quer. Manche passen gar nicht auf die Skala während ein anderer Durchgang nach nur 11 Auswertungen einen der besten Werte insgesamt findet, oder besser gesagt, errät. Diese große Varianz macht den Algorithmus unbrauchbar.





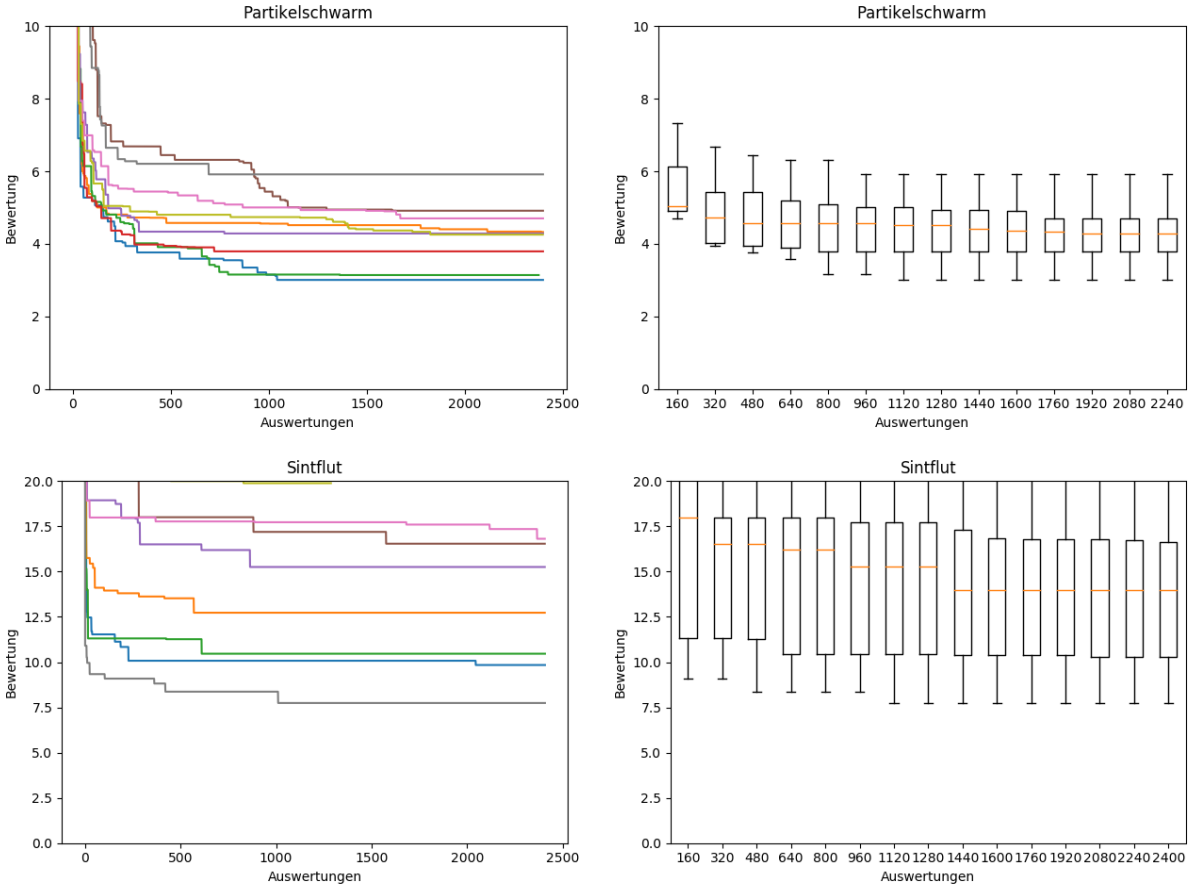
Der Downhill Simplex-Algorithmus konvergiert sehr schnell innerhalb der ersten 160 Auswertungen und bleibt dann in einem lokalen Minimum liegen. Er profitiert nicht

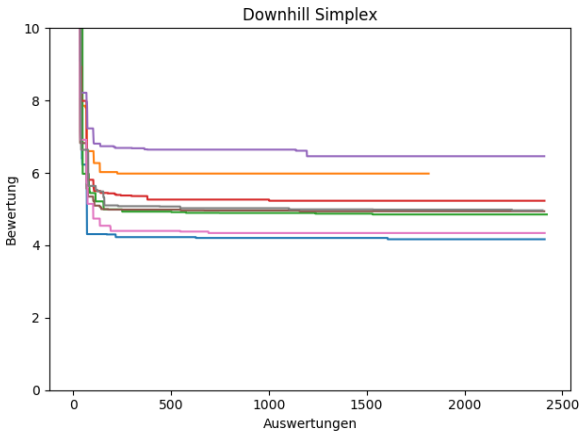
wirklich von mehr Auswertungen. Die schnelle und sehr konsistene Konvergenzrate ist ein Vorteil.

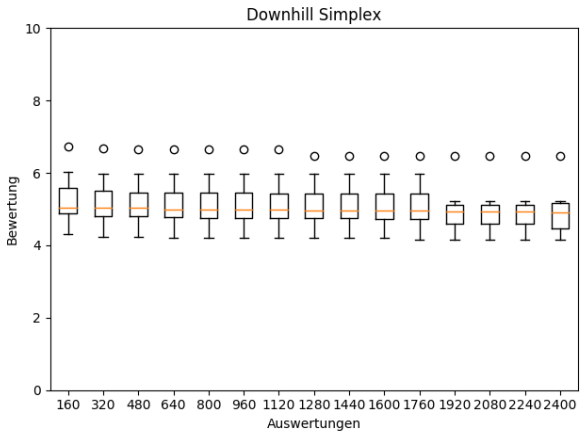
	160 Auswertungen			480 Auswertungen		
	Partikel schwarm	Sintflut	Downhill Simplex	Partikel schwarm	Sintflut	Downhill Simplex
Anzahl der Durchführungen	24	24	24	14	14	13
Mittelwert	3.94	6.99	4.50	3.25	6.95	4.39
Median	3.95	7.07	4.39	3.14	7.34	4.42
Varianz	0.53	4.59	0.96	0.34	5.16	1.12
Bester Wert	2.73	2.58	2.55	2.53	2.58	2.49
Schlechtester Wert	5.91	11.39	6.56	4.24	10.04	5.89
Durchschnitt der TOP3	2.84	3.18	2.90	2.55	3.76	2.82

6.2 Experiment 2

Für das zweite Experiment geben wir keine Einschränkungen an die Parameter vor. Jedes Jahr ist unabhängig von den anderen. Damit ist $d = 30$. Wie zuvor seinen die $bounds = [0, 1500]^d$. Die Anzahl der erlaubten Auswertungen wird auf 2400 erhöht.







Durch die größere Freiheit in den Parametern erhofft man sich besser bewertete Lösungen, da ja der alte Parameterraum Teilmenge des neuen ist. Alle drei Algorithmen können aber leider mit der zusätzlichen Komplexität nicht umgehen und erbringen trotz verfünffachter Zeit schlechtere Bewertungen. Der Sintflut-Algorithmus leidet ganz besonders, man beachte die veränderte y-Achse.

	1200 Auswertungen			2400 Auswertungen		
	Partikel schwarm	Sintflut	Downhill Simplex	Partikel schwarm	Sintflut	Downhill Simplex
Anzahl der Durchführungen	9	9	8	7	8	7
Mittelwert	4.37	14.76	5.13	4.42	13.89	4.99
Median	4.52	15.26	4.97	4.33	13.99	4.94
Varianz	0.77	20.41	0.51	0.71	18.29	0.48
Bester Wert	3.01	7.74	4.20	3.01	7.74	4.17
Schlechtester Wert	5.92	21.74	6.46	5.92	21.74	6.46
Durchschnitt der TOP3	3.32	9.43	4.48	3.70	9.35	4.45

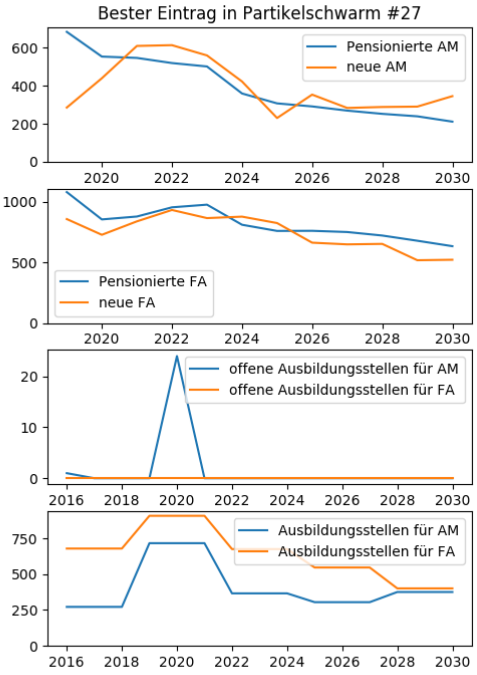
6.3 Fazit

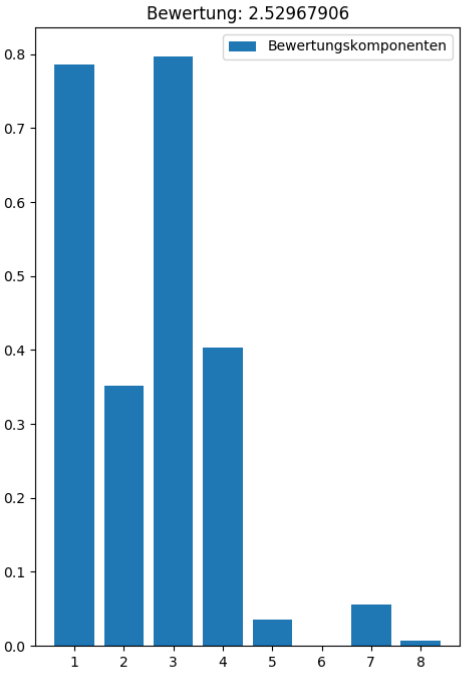
Was ist nun der beste Kalibrierungsansatz? Die Antwort hängt vom Kontext der Kalibrierung ab. Wenn es schnell gehen soll, also in der Größenordnung von ein paar Stunden, besticht der Downhill Simplex mit seiner schnellen Konvergenzrate. Hat man mehr Zeit und wünscht sich eine bessere Güte des Ergebnisses, wird der Partikelschwarm attraktiver. Ab einer gewissen Grenze lohnt es sich aber, mehrere kürzere *Runs* zu starten und den besten dieser zu wählen.

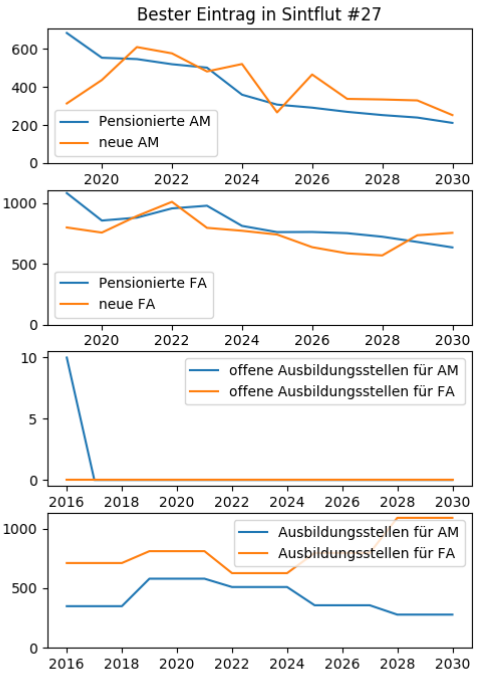
Kalibriert man nicht auf einem Desktop oder Laptop, sondern einem Cluster mit sehr vielen Prozessorkernen ist der Partikelschwarm definitiv der beste Ansatz aufgrund seiner überragenden Skalierbarkeit.

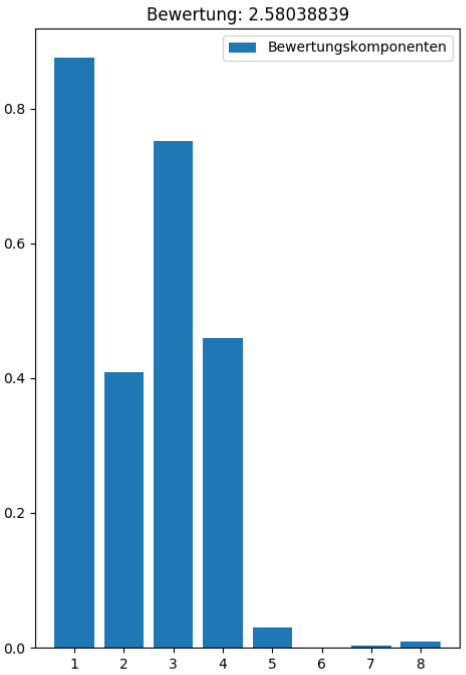
Die Zusammenfassung von je drei Jahren zu einem Parameter ist eine sehr sinnvolle Einschränkung. Der vermeintliche Mehrgewinn durch die Freiheit der Parameter ist die zusätzliche Komplexität nicht wert.

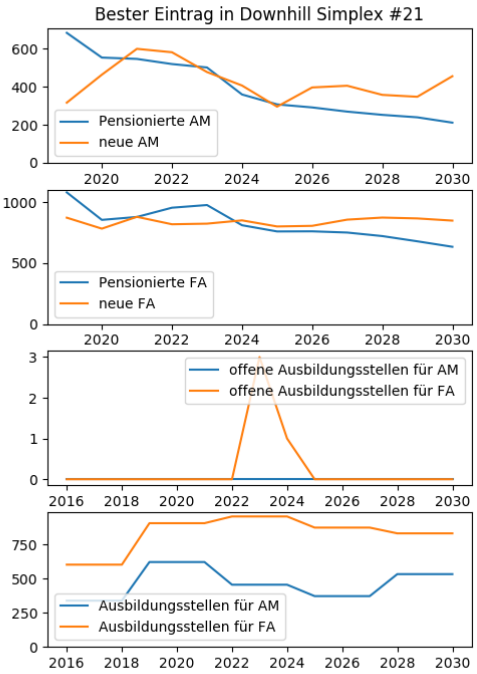
7 Die besten Lösungen

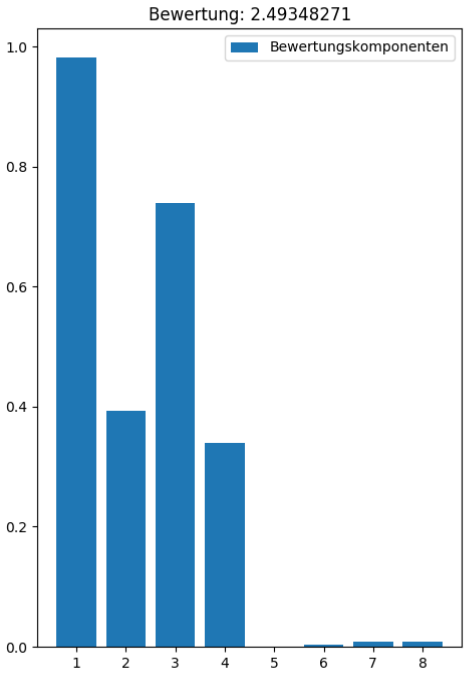









Bester Eintrag in Downhill Simplex #21
Pensionierte AM
neue AM
Pensionierte FA
neue FA
offene Ausbildungsstellen für AM
offene Ausbildungsstellen für FA
Ausbildungsstellen für AM
Ausbildungsstellen für FA


Bewertung: 2.49348271
Bewertungskomponenten