# Mensa Modellierung

Von

J. Hausladen, F. Hirn, T. Cil, M. Betz, M. Ströll

Projektbericht

FAU Erlangen Nürnberg

WiSe 22/23

# Inhaltsverzeichnis

# EINLEITUNG

Jeder Studierende, jeder Mitarbeiter, also auch jeder Professor der Friedrich-Alexander-Universität in Erlangen kennt die Situation. Man möchte gerne zu Mittag essen, doch man sieht bereits von weitem, dass sich wieder eine enorme Schlange an der Treppe West gebildet hat. Meist hat dieser Stau mit einer Engstelle vor der Essensausgabe zu tun, bei der Menschen nebeneinander stehen und so ein Durchkommen verhindern. Manchmal liegt es auch daran, dass sich für das Essen an der mittleren Theke ebenfalls an der Schlange für Theke 1 und 2 angestellt wird. Welche Möglichkeiten sind uns gegeben, den Andrang zu modellieren und anschließend zu optimieren?

In unserer folgenden Projektarbeit haben wir dieses Thema nun genauer untersucht und dazu Abbildungen angefertigt, die den Studentenfluss simulieren. Ziel ist es, eine Lösung für das genannte Problem zu finden und die Wartezeit zu beseitigen, oder auch größtmöglich zu minimieren. Wir möchten dazu mehrere Vorschläge präsentieren. Unser Vorgehen hierbei richtet sich nach dem Modellierungszyklus der Mathematischen Modellierung, bei dem wir uns zunächst über das Problem klar geworden sind. Dafür haben wir den Ablauf der Mittagszeit beobachtet und haben selbst relevante Daten erhoben. Die entsprechenden Skalen, sowie die Datenbeschaffung beschreiben wir genauer im ersten Kapitel Datenerhebung 1. Im Vorhinein machten wir uns bereits Gedanken über die Wahl des mathematischen Modells. Wieso wir uns für die Verkehrssimulation entschieden haben, wird in Kapitel Modellentwicklung 2 erläutert. Zusätzlich stellen wir den Umgang mit den gewonnenen Daten dar.

Für eine numerische Simulation des Modells steht uns bereits eine Implementierung für eine Verkehrssimulation zur Verfügung, auf welcher unsere Arbeit basiert. Den Umgang damit werden wir in Passage Implementierung 3 konkretisieren. Im Kapitel Simulationsauswertung 4 behandeln wir noch einmal genauer den Umgang mit den aus der Simulation gewonnenen Daten und interpretieren verschiedene Lösungen. Abschließend legen wir in Kapitel 5 Optimierung & Fazit 5 die Schlüsse aus unseren Erkenntnissen dar und ziehen ein kurzes Fazit zum Projekt.

# 1 Datenerhebung

Beginnend stellen wir vor, wie wir unsere Daten erhoben haben. Dafür haben wir an einem Tag die Mensa beobachtet. In einem Zeitintervall von 5 Minuten haben wir gezählt, wie viele Studenten

1. Die Treppen Ost und West betreten.

2. Die Theken Ost und West bedienen.

3. Welche Essen gewählt werden.

4. Wie viele Personen an welcher Kasse bezahlen.

Natürlich spiegeln die Daten nur einen Tag in der Mensa und wir können nicht ohne weitere Daten darauf aufbauen. Aus diesem Grund baten wir bei der Mensaleitung um zusätzliche Daten, welche uns anschließend in einer Excel Datei zur Verfügung gestellt wurden, in der eine Woche Mensabetrieb dargestellt wurde. Bei der Analyse der Daten fiel aber sehr schnell auf, dass uns diese unsere Daten nicht ersetzen können, sondern eher als Bestätigung unsere Messung gesehen werden sollten. Die Daten haben leider nur eine Genauigkeit von Zeitintervallen in Stundenlänge. Dafür geben sie jedoch Aufschluss über die von uns nicht erhobenen Daten.

Es wurden 10060 Essen an 5 Tagen ausgegeben, was mit den etwa 2000 an einem Tag von uns gemessenen Speisen übereinstimmt. Somit können wir zeigen, dass wir einen durchschnittlich besuchten Tag für unsere Messung in der Mensa hatten.

Zusätzlich zeigt sich, dass die Salattheke (stets  0,5% Anteil am Tag) sowie die Selbstbedienungstheke ( 5% Anteil am Tag), für unser Modell einen vernachlässigbaren Anteil haben, sodass wir uns in unserem realen Modell ebenfalls nur auf die Theken konzentrieren können. Die SB-Theke können wir im Modell ignorieren, da sie einerseits, einen sehr geringen Anteil aufweist, aber auch teilweise nur als Beilage zusätzlich gesehen werden, weshalb sich die Relevanz ebenfalls verringert. Gleichzeitig hat diese aus den genannten Gründen auch keinen bis kaum einen Einfluss auf die Problemstelle, die zu analysieren unser Auftrag war.

# 2 Modellentwicklung

Ein erster wichtiger Schritt in der Modellentwicklung war es, sich für ein geeignetes Modell zu entscheiden. Für uns hieß das, die Wahl zwischen Verkehrssimulation und Schwarmsimulation zu treffen. Bei der Schwarmsimulation wird die Bewegung des einzelnen Studenten hauptsächlich nach den folgenden Prinzipien beeinflusst:

Kohäsion: Der Student versucht in der Nähe der restlichen Studenten, also in der Mensa zu bleiben.

Separation: Der Student versucht Kollisionen mit anderen Studenten (und Mobiliar) zu vermeiden.

Alignment: Der Student versucht seine Bewegungsrichtung und Geschwindigkeit an die seiner Nachbarn anzupassen.

Eine solche Schwarmsimulation hätte zwar den Vorteil einer realistischeren Darstellung des Problems an der Engstelle geboten, aber auch Nachteile wie eine ungünstige Modellierbarkeit der architektonischen Einschränkungen (Anziehung/Abstoßung von Theken, Wänden, ...) und den Verlust der Umsetzbarkeit der Schlangenbildung der Studenten gehabt.

Bei der Verkehrssimulation hingegen hängt die Bewegung eines einzelnen Studenten meist nur von dessen Wunschgeschwindigkeit und -abstand, sowie dem vorausfahrenden Studenten ab. Hierbei bieten sich Vorteile wie eine einfachere Implementierung mit simpleren Differentialgleichungen als bei der Schwarmsimulation, sowie der Irrelevanz der Umgebung außerhalb des Straßennetzes. Als Nachteil gibt es nur zu nennen, dass man die Mensa (inklusive der Staustelle beim Treppenaufgang) erst in ein Straßennetz übersetzen muss.

Nach diesen Überlegungen entschieden wir uns für die Umsetzung der Verkehrssimulation und stellten zusätzlich noch einige Modellannahmen auf: Wir gehen davon aus, dass die Studenten bereits vor Betreten der Mensa wissen, welches Gericht sie erwerben möchten und an welcher Theke dieses ausgegeben wird. Des Weiteren nehmen wir an, dass jeder Student maximal ein Hauptgericht nimmt und den kürzesten verfügbaren Weg zur entsprechenden Theke wählt. Außerdem verzichteten wir auf die Modellierung von insignifikanten Angeboten mit einer durchschnittlichen Nutzung durch weniger als 5 Prozent der Studenten. Gemäß den von der Mensa bereitgestellten Daten waren dies: die Salatbar sowie die Selbstbedienungstheke in der Mitte der Mensa. In der Simulation fließt der Studentenstrom somit von den Treppen über die jeweiligen Theken hin zu den entsprechenden Kassen.

Mit der Entscheidung für eine Verkehrssimulation, gilt es nun, die in der Datenerhebung gewonnenen Erkenntnisse zu abstrahieren. Zum einen müssen wir aus den Ergebnissen der Treppenzählung eine Funktion $\phi$ herleiten, die zu einem gegebenen Zeitpunkt t die Anzahl der neu in die Mensa kommenden Studenten $\phi(t)$ zurückgibt. Da die Interpolation aller Datenpunkte (vgl. Abb. 1) ein zu spezifisches Muster ergäbe, entscheiden wir uns dazu, die Funktion zu vereinfachen. Um dies zu bewerkstelligen, lassen wir nicht essenzielle Datenpunkte weg und interpolieren lediglich die Werte bei den Extremalstellen der ursprünglichen Interpolation (vgl. Abb. 2). Die erhaltene Funktion liegt immer noch nah genug an den Rohdaten, um diese angemessen zu repräsentieren, ist aber gleichzeitig allgemein genug, um für jeden Wochentag verwendet werden zu können. Für die Interpolation wird in Python die kubische Spline-Interpolation aus dem Paket Scipy verwendet. Somit ergibt sich für die Studentenflussfunktion $\phi(t)$:

$$\phi(t) = scipy.interpolate.Bspline(scipy.interpolate.splrep(X, Y, s = 0)) \qquad (1)$$

wobei $X = [0, 5, 45, 70, 80, 100, 115, 135, 155]$ und $Y = [96, 120, 33, 80, 28, 104, 23, 96, 3]$, die Zeiten und Studentenzahlen aus den Rohdaten sind und $s = 0$ eine Glattheitsparameter der Funktion ist.

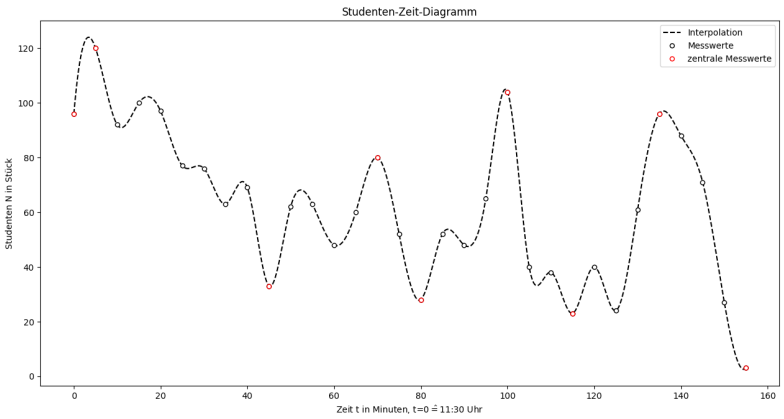


Abbildung 1: naive Interpolation

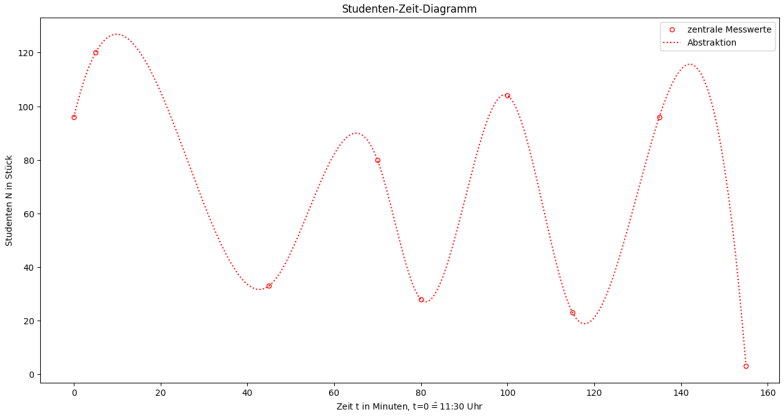


Abbildung 2: abstrahierte Interpolation

Zum Anderen gilt es, die Verteilung der Studenten auf die einzelnen Theken zu abstrahieren. Hierzu definieren wir eine Verteilungsfunktion $\Theta : [0,1] \times \mathbb{R}^+ \to [0,1]^6$, die von der Treppenverteilung $\omega \in [0,1]$ (wobei $\omega$ der Anteil an Studenten ist, die über die Westtreppe kommen. $1-\omega$ ist dann der entsprechende Anteil für die Osttreppe) und der Zeit $t$ abhängt. Sie gibt ein 6-Tupel an Werten zurück, die jeweils die Anteile der angestrebten Theken zum jeweiligen Zeitpunkt darstellen. Da die erhobenen Rohdaten (vgl. Abb. 3) auch diesbezüglich zu spezifisch sind, mitteln wir - auf 4 Zeitintervallen aufgeteilt - die Werte (vgl. Abb.4). Auch wenn die Werte in den ersten 3 Intervallen recht ähnlich sind, ist dies durchaus sinnvoll, da sich vor allem das letzte Intervall (nach Schließung der Osttreppe) stark von den anderen unterscheidet. Somit ergibt sich für die Verteilungsfunktion $\Theta$ mit $I_1 = [0, 2700)\,; I_2 = [2700, 5400)\,; I_3 = [5400, 8100)\,; I_4 = [8100, 9600]$ folgende Definition:

$$\Theta(\omega, t) = \begin{cases} (0.28\omega, 0.26\omega, 0.46\omega, 0.16(1-\omega), 0.42(1-\omega), 0.39(1-\omega))^{\mathrm{T}} & t \in I_1 \\ (0.31\omega, 0.31\omega, 0.39\omega, 0.21(1-\omega), 0.43(1-\omega), 0.36(1-\omega))^{\mathrm{T}} & t \in I_2 \\ (0.29\omega, 0.30\omega, 0.42\omega, 0.09(1-\omega), 0.61(1-\omega), 0.35(1-\omega))^{\mathrm{T}} & t \in I_3 \\ (0.24\omega, 0.39\omega, 0.37\omega, 0, 0, 0)^{\mathrm{T}} & t \in I_4 \\ 0 & \text{sonst} \end{cases} \quad (2)$$

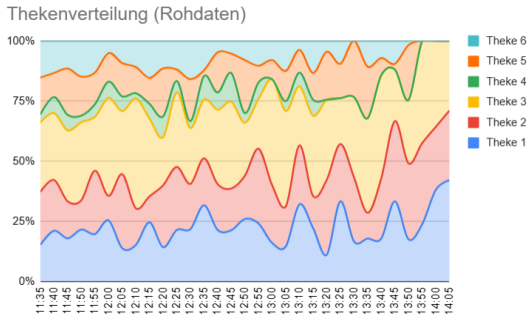


Abbildung 3: Rohdaten der Thekenverteilung

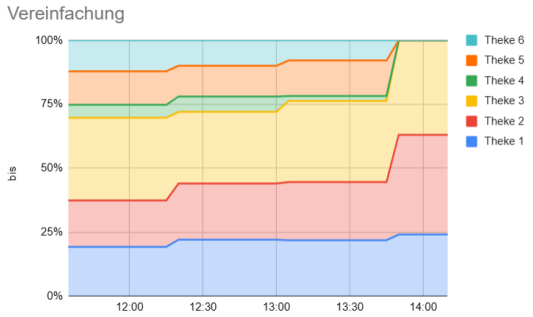


Abbildung 4: abstrahierte Version der Thekenverteilung

# 3 Implementierung

Wie bereits in der Einleitung beschrieben, stand uns eine Implementierung einer Verkehrssimulation zur Verfügung, mit deren Hilfe wir den Ablauf in der Mensa modellieren wollten. Im Sinne der objektorientierten Programmierung wird hierbei jedem Objekt einer realen Straßenkarte äquivalent eine Klasse bzw. Python-Datei zugeordnet. Eine *Straße* besteht dabei trivialerweise aus jeweils einem zweidimensionalen Start- und Endpunkt, die verbunden werden. Der Klasse *Ampel* werden eine oder mehrere Straßen übergeben, an deren Enden sie sich befinden. Werden mehrere Straßen derselben Ampel-Klasse übergeben, schalten die einzelnen Ampeln im selben Zyklus. Ein *Fahrzeug* wird mit einem Gewicht und einem Pfad, bestehend aus den einzelnen zu befahrenden Straßen erstellt und an den *Fahrzeuggenerator* übergeben. Das Gewicht bestimmt hierbei, wie häufig ein einzelnes Fahrzeug eines bestimmten Pfades erstellt wird und entspricht dem Produkt der Studentenflussfunktion mit der Verteilungsfunktion auf die einzelnen Theken, die in Kapitel 2 vorgestellt wurden. Zur zeiteffizienten Berechnung dieser Gewichte haben wir die Klasse *spawning* eingeführt.

Ein Fahrzeug entspricht in unserer Simulation einem Studenten, wobei sein Pfad durch die Auswahl eines Essens bestimmt wird bzw. welche Wege er bis zur Kasse wählt. Mithilfe der Ampeln lässt sich die Wartezeit an den Essensausgaben und den Kassen repräsentieren. Und der Fahrzeuggenerator wird an den Treppen platziert, um den Zustrom an Studenten über die einzelnen Zeitabschnitte darzustellen. Hierzu wurde innerhalb der Klasse mit der Studentenflussfunktion aus Kapitel 2 eine Variable vehicle_rate berechnet. Die Aufgabe der Implementierung bestand hauptsächlich darin, die Anpassung des Codes an die spezifischen Gegebenheiten der Mensa durchzuführen. Zunächst sollte aber das Layout der Mensa erstellt werden. In Abbildung 5 ist die Mensa ohne Kassen und in Abbildung 6 mit diesen zu sehen. Die einzelnen Theken wurden als Blöcke auf der graphischen Oberfläche erstellt, sind also keine eigenen Klassen.

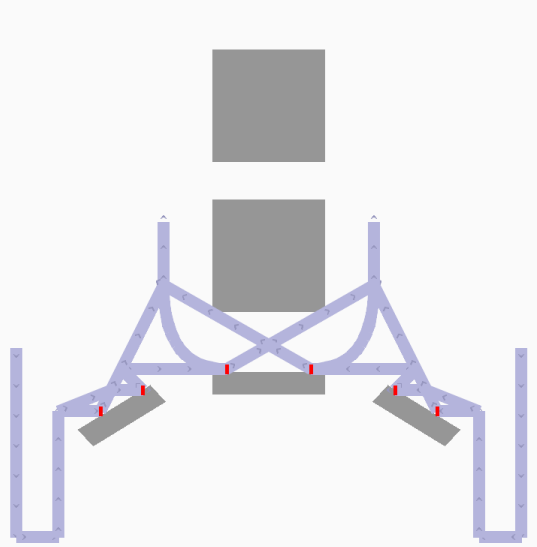

Abbildung 5: Karte der Mensa ohne Kassen

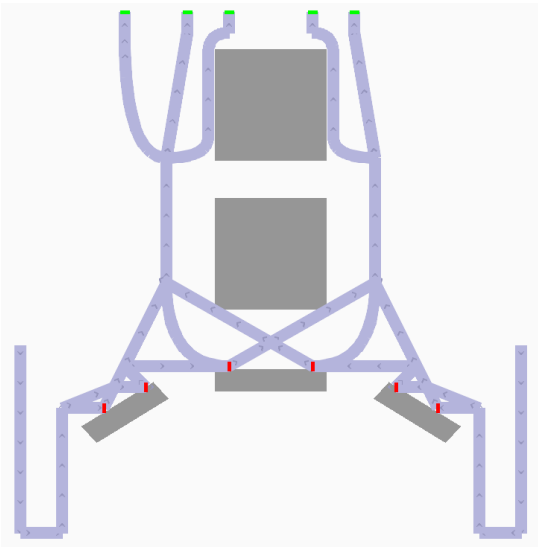

Abbildung 6: Karte der Mensa mit Kassen

In der gegebenen Verkehrssimulation wurden Kreuzungen mithilfe von Ampeln modelliert, da wir diese aber bereits für die Theken und Kassen vorgesehen hatten, musste das Verhalten der Studenten bei sich schneidenden Pfaden neu implementiert werden. Wir lösten dies mithilfe einer Warteschlange innerhalb der Update-Funktion der Klasse Straße. Dies ist im ersten Codebeispiel 1 dargestellt. Als Erstes wird in Zeile 1 geprüft, ob das erste Fahrzeug auf der Straße eine weitere in seinem Pfad enthält. Ist dies der Fall, wird diese in der zweiten Zeile als next_road definiert. Handelt es sich bei dieser um eine Straße, die einer Kreuzung folgt, und ist das Fahrzeug nahe genug dran (Zeile 3), folgen 2 weitere Abfragen. Ansonsten wird das Fahrzeug nicht beeinflusst. Die erste Abfrage in Zeile 5 prüft, ob das Fahrzeug in der Warteschlange enthalten ist, wobei es hinzugefügt wird, wenn dies nicht Fall ist. Anschließend wird in Zeile 8 geprüft, ob das Fahrzeug das Erste in dieser Warteschlange ist. Trifft dies zu, wird seine Geschwindigkeit um den Faktor in Zeile 9 verringert, um es auszubremsen. Unterschreitet das Fahrzeug darüber hinaus den vordefinierten Abstand der sogenannten „Stopping-Zone“, wird es komplett gestoppt (Zeile 13). Treten die beiden Fälle nicht ein, wird ein bestehender Stop des Fahrzeugs wieder aufgehoben (Zeile 15) und weitere folgende Fahrzeuge wieder beschleunigt (Zeile 17). Insgesamt bewirkt dieser Code, dass der erste Student, der eine Schnittstelle erreicht, diese immer überqueren kann, und alle weiteren entweder ausgebremst werden oder komplett zum Stehen kommen.

```
1 if first.current_road_index < len(first.path) - 1:
2     next_road = self.sim.roads[first.path[first.current_road_index +
  1]]
3     if next_road.is_merging and (first.x >= self.length - self.
  intersection_slow_distance):
4         # add to waiting queue
5         if next_road.merging_queue.count(first) == 0:
6             next_road.merging_queue.append(first)
7         # slow down if not first
8         if next_road.merging_queue.index(first) != 0:
9             factor = (self.length - first.x) / (self.
```

```
    intersection_slow_distance) * self.intersection_slow_factor
10              first.slow(factor * first._v_max)
11              if first.x >= self.length - self.intersection_stop_distance
     and first.x <= self.length - self.intersection_stop_distance / 2:
12                  # Stop vehicles in the stop zone
13                  first.stop()
14          else:
15              first.unstop()
16              for vehicle in self.vehicles:
17                  vehicle.unslow()
```

Listing 1: Kreuzungen

Bei gewöhnlichen Ampeln wird in gleichbleibenden Intervallen zwischen der Rot- und Grünphase gewechselt. Unsere Idee war es aber, dass jeder Student bei der Abholung seines Essens bzw. der Bezahlung für die Zeit, die dieser Vorgang benötigt, stehen bleibt und alle weiteren Studenten solange warten müssen. In Codebeispiel 2 ist zu sehen, wie das in der Ampel-Klasse umgesetzt wurde. Zeile 9 entspricht dabei der Rot- und Zeile 11 der Grünphase. Die Funktion *increment* wird immer dann aufgerufen, wenn ein Student die Ampel erreicht. Der Zähler der passierten Fahrzeuge (Zeile 14), sowie die Verzögerung werden inkrementiert (Zeile 15). Unsere Erhebung, wie in Kapitel 1 beschrieben, hat ergeben, dass der Vorgang des Tellernehmens ca. 3 Sekunden dauert. Dies entspricht 150 Zeiteinheiten in unserer Simulation, was in der Variable *cycle_delay* gespeichert wurde. Zudem sollen alle 6 Studenten die Teller wieder aufgefüllt werden. Zu diesem Zweck wird in Zeile 17 die Verzögerung um 300 Zeiteinheiten bzw. 6 Sekunden erhöht.

```
1  def update(self, sim):
2      if self.fixed_cycle:
3          cycle_length = 30
4          k = (sim.t // cycle_length) % 2
5          self.current_cycle_index = int(k)
6      else:
7          self.delay = max(0, self.delay - 1)
8          if self.delay > 0:
9              self.current_cycle_index = 0
10         else:
11             self.current_cycle_index = 1
12
13 def increment(self):
14     self.passed_cars += 1
15     self.delay += self.cycle_delay
16     if self.passed_cars % 6 == 0:
17         self.delay += 300
```

Listing 2: Ampeln

# 4 Simulationsauswertung

Aus der Modellentwicklung in Kapitel 2 ist bereits offensichtlich, dass einige wenige Parameter deutlich besser untersucht werden können als andere. Vor allem Faktoren wie der Grundriss der Mensa oder die Ausgabegeschwindigkeit der Essen an den Theken

bleiben daher konstant in der Auswertung. Obwohl auch die Anzahl an Studenten sowie die zeitliche Verteilung dieser nicht in unserer Hand liegen und damit nicht optimierbar sind, betrachten wir trotzdem andere Studentenflüsse, insbesondere solche, die konstant sind, um theoretische Limits für einen maximalen Durchfluss an Studenten zu bestimmen. Wir betrachten dabei nur ein Modell über die Treppenaufgänge und Theken, die Kassen werden in der Auswertung nicht berücksichtigt. Primär folgt daraus, dass die berechneten Zeiten auch nur für die Strecke durch die Mensa ohne Kassen gültig sind, hier ist aber schon genug zu optimieren. Wir benötigen allerdings zuerst eine geeignete Norm für die Wartezeit der Studenten in der Mensa. Naheliegend wäre es, einfach den Durchschnitt über die Wartezeit aller Studenten $\mathrm{W}_1 = \frac{1}{N}\sum_{i=1}^{N} T_i$ zu betrachten. Für unsere Optimierung erscheint dieser Durchschnitt nicht optimal, denn eine Minimierung dieser Norm sorgt nicht für Elimination von starken Peaks und Ausreißern zu Stoßzeiten. Das andere Extrem wäre die Unendlich-Norm $\mathrm{W}_\infty = \max_{i=1\ldots N} T_i$, die nur die längste Wartezeit aller Studenten betrachtet. Auch das ist nicht ideal, da im schlimmsten Fall bei Minimierung von $\mathrm{W}_\infty$ der Durchschnitt $\mathrm{W}_1$ sogar steigt. Wir betrachten deshalb eine euklidische Norm

$$\mathrm{W}(v,w) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} T_i\,(v,w)^2} \tag{3}$$

Dabei ist die Division durch $N$ relevant für eine Vergleichbarkeit bei Simulationen mit verschiedener Anzahl an Studenten. In 3 wird bereits die Abhängigkeit von der maximalen Geschwindigkeit $v$ in Pixel pro Sekunde, und $w \in [0,1]$, dem Parameter, der angibt, zu welcher Wahrscheinlichkeit ein neuer Student die West-Treppe benutzt, hervorgehoben. Abhängigkeiten der Funktion vom Studentenfluss und der einzelnen Gewichte für Essen 1 bis 6 sind nicht explizit angegeben, da diese in den meisten Fällen anhand unserer Daten gewählt werden und nur in einzelnen Auswertungen angepasst werden. Es ist dementsprechend unser Ziel, Werte $v_*$ und $w_*$ zu finden, die 3 minimieren. $v_*$ sollte dabei in einem angemessenen Bereich liegen.

$$(v_*, \omega_*) = \arg\min_{v\in[8,20],\omega\in[0,1]} \mathrm{W}\,(v,\omega) \tag{4}$$

Bei dieser Minimierung stützen wir uns vor allem auf eine Vielzahl an Simulationsauswertungen und versuchen so Erkenntnisse zu erlangen, wo optimale Werte für $v$ und $w$ liegen könnten. In Abbildung 7 werden die Geschwindigkeit und Verteilung gegen die Norm aus 3 aufgetragen. Bei Wahl eines konstanten Flusses, erkennt man hauptsächlich, dass erwartungsgemäß eine schnellere Maximalgeschwindigkeit zu kürzeren Wartezeiten führt. Die Norm hat tatsächlich die Einheit Sekunden und darf als Wartezeit interpretiert werden, rote Werte (höher) sind dabei natürlich schlechter als grüne Werte.

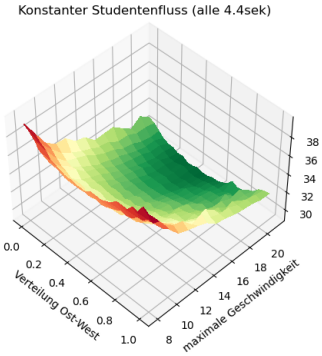


Abbildung 7: Norm der Wartezeit bei konstantem Studentenfluss mit einem Student pro ca. 4, 4 Sekunden

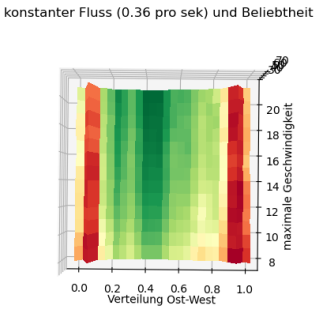


Abbildung 8: Ansicht von oben, konstanter Studentenfluss mit ca. 0.36 Studenten pro Sekunde, also größer als in Abbildung 7 mit ca. 0.23

Interessanter werden die Plots und Auswertungen, wenn der Studentenfluss erhöht wird und so realistischere Werte für die Menge an Studenten erreicht werden. Insbesondere in Abbildung 8 wird ersichtlich, dass neben der Geschwindigkeit bei größeren Flüssen auch an anderer Stelle Optimierungspotential vorliegt. Die Norm sinkt für schneller Geschwindigkeit zwar noch immer, im Vergleich zu einer Veränderung des Parameters w, sind diese Verbesserungen jedoch nur minimal. Stattdessen erscheint es nun relevanter, den Parameter $w$ entsprechend anzupassen, sodass eine möglichst kleine Wartezeit erreicht wird.

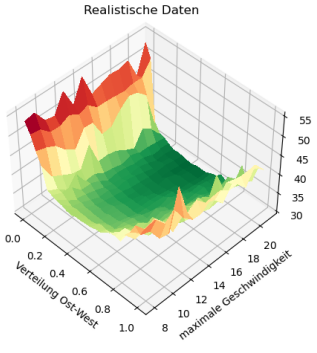


Abbildung 9: Norm der Wartezeit mit realistischen Werten für Beliebtheit der Essen und Studentenfluss

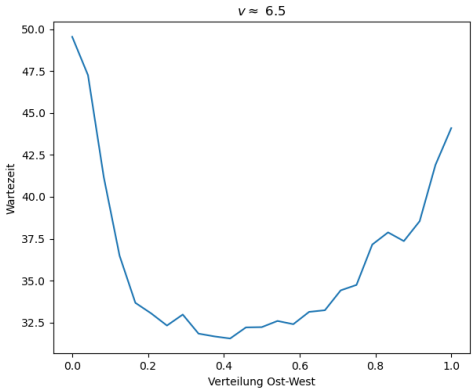


Abbildung 10: Ausschnitt aus Abbildung 9 für festes $v$

Auch bei Betrachtung der realistischen Daten, in Stoßzeiten betreten mehr als 0.36 Studenten die Mensa pro Sekunde, ist die Auswertung ähnlich. Gerade Abbildung 10 verdeutlicht das Optimierungspotential, aus unserer Datenerhebung wissen wir, dass aktuell eine Ost-West-Verteilung von ungefähr 0.75 vorliegt, bei Verbesserung dieser also eine Verbesserung der Norm mit sich bringen würde.

Dank der großen Flexibilität unseres Modells, können leicht auch weitere Parameter untersucht werden, die zwar in der Mensa nicht so leicht umgesetzt werden können,

aber dennoch zu interessanten, jedoch nicht mehr wirklich besseren, Resultaten führen. Beispielsweise sorgt eine strikte Trennung von Essen 1 und 2 auf die Theken Ost und West, also dass an jeder Seite nur ein Essen zur Wahl steht, schon für minimal bessere Wartezeiten.

# 5 Optimierung und Fazit

Nach der Auswertung der Modellierung wollen wir die Ergebnisse noch analysieren und nach Optimierungsmöglichkeiten suchen: Wir entscheiden uns in der Auswertung für eine naive Sensitivitätsanalyse (also der Betrachtung der relativen Änderungen in den Graphen bei Varianz der Parameter), anstelle einer formalen partiellen Ableitung in die Parameterrichtung, da Ersteres hier deutlich praktikabler scheint. In Abbildung 8 lässt sich gut erkennen, dass die Abweichungen in der Wartezeit bei Änderung der Treppenverteilung $\omega$ deutlich drastischer ausfallen als bei Änderung der Studentengeschwindigkeit $v$. Die Wartezeit hängt also deutlich stärker von $\omega$ ab als von $v$. Demnach ist die Wartezeit also gegenüber der Treppenverteilung sensitiver, als gegenüber der Studentengeschwindigkeit.

Nun gilt es noch ein Fazit aus der Modellierung zu ziehen. Erwartungsgemäß hat unsere Auswertung bestätigt, dass eine gleichmäßige Aufteilung der Studenten auf die beiden Treppen (also $\omega = 1 - \omega = 0,5$) die Wartezeit minimiert (vgl. Abb. 9 und 10). Bei hohen Geschwindigkeiten der Studenten ist dahingegen die Ausgabegeschwindigkeit an den Essenstheken der limitierende Faktor. Ein weiteres, leicht nachvollziehbares Resultat, das wir erhalten haben ist, dass die Wartezeit minimiert wird, wenn sich der Andrang auf die einzelnen Gerichte gleichmäßig verteilt, also jedes Gericht gleich beliebt ist. Allerdings liegt es nicht in unserer Macht, eine solche Verteilung zu erwirken, da jeder Student persönliche Vorlieben für sein Mittagessen hat.

Nun wollen wir unsere Ergebnisse noch in Zahlen konkretisieren: Unter realen Bedingungen ergibt unser Modell eine durchschnittliche Wartezeit von $t^{\emptyset}_{real} = 33,6$ Sekunden. Unter den optimierten Bedingungen ergibt sich $t^{\emptyset}_{optimiert} = 31,3$ Sekunden. Somit kann durch die genannten Optimierungen die durchschnittliche Wartezeit um etwa 7 % verringert werden. Dies scheint leider keine allzu starke Verbesserung zu sein. Betrachtet man stattdessen jedoch die durchschnittliche Wartezeit während der Stoßzeiten, so erhält man $t^{\emptyset}_{real;Stoß} = 39,4$ Sekunden und $t^{\emptyset}_{optimiert;Stoß} = 34$ Sekunden. Bei dieser Größe erzielen die Optimierungsmaßnahmen immerhin schon eine Verbesserung um etwa 14 % . Betrachten wir anschließend noch einige Größen, die insbesondere für den einzelnen Studenten bei der Planung der Mittagspause eine Rolle spielen. Zum einen die Zeit, die Studierende länger im Schnitt mehr warten müssen, wenn sie innerhalb statt außerhalb der Stoßzeiten in die Mensa gehen. Bei dieser Größe erreichen unsere Optimierungen eine Verbesserung von knapp 54 %.

$$\frac{t^{\emptyset}_{real;Stoß} - t^{\emptyset}_{real}}{t^{\emptyset}_{optimiert;Stoß} - t^{\emptyset}_{optimiert}} = 0.534 \tag{5}$$

Eine andere für den Studierenden relevante Größe ist noch die Worst-Case-Wartezeit. Hierbei lassen sich aus unserem Modell die Werte $t^{max}_{real} = 94$ Sekunden und $t^{max}_{optimiert} = 72$ Sekunden ermitteln, was einer Verbesserung von knapp 24 % entspricht (vgl. Abb. 11).

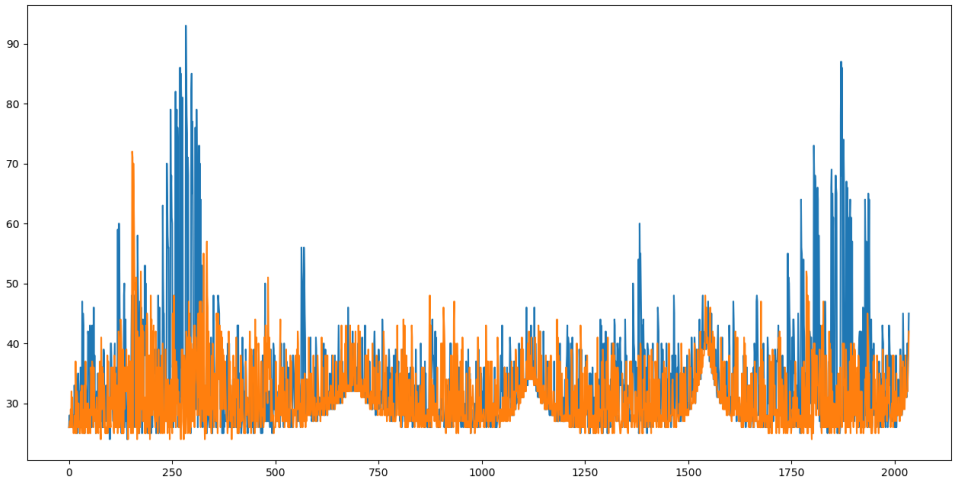


Abbildung 11: Wartezeiten real und optimiert

Somit kann geschlossen werden, dass die mit Hilfe des Modells ermittelten Optimierungsmaßnahmen eine deutlich bemerkbaren Effekt hätten. Fraglich ist trotzdem noch die Umsetzung, denn ein einfaches Schild mit dem Hinweis die Treppe Ost zu benutzen wird erfahrungsgemäß den „Faulen Studierenden “nicht dazu bewegen, einen längeren Weg für eine kürzere Zeit in der Mensa in Kauf zu nehmen. Eine digitale Anzeige mit aktuellen Echtzeiten und klar erkennbarer Zeitersparnis würden vermutlich effektiver bei pragmatischen Studierenden sein, jedoch nicht beliebig simpel umzusetzen. Es bleibt also, selbst das Wissen über die Zeitersparnis zu haben und selbst als Wissender seinen Teil zu einer besseren Verteilung auf die Treppen beizutragen!

# Abbildungsverzeichnis

# Literatur

[Hau+23] J. Hausladen, F. Hirn, T. Cil, M. Betz und M. Ströll. *Mensa-Optimierung*. März 2023. URL: https://github.com/9VZZH7/Mensa-Optimierung.

[Him21] B. Himite. *Simulating Traffic Flow in Python*. 2021. URL: https://towardsdatascience.com/simulating-traffic-flow-in-python-ee1eab4dd20f (besucht am 12. 12. 2022).