Abschlusspräsentation Mensa-Optimierung

Jannik Hausladen Fabian Hirn Temucin Cil Martin Betz Matthias Ströll

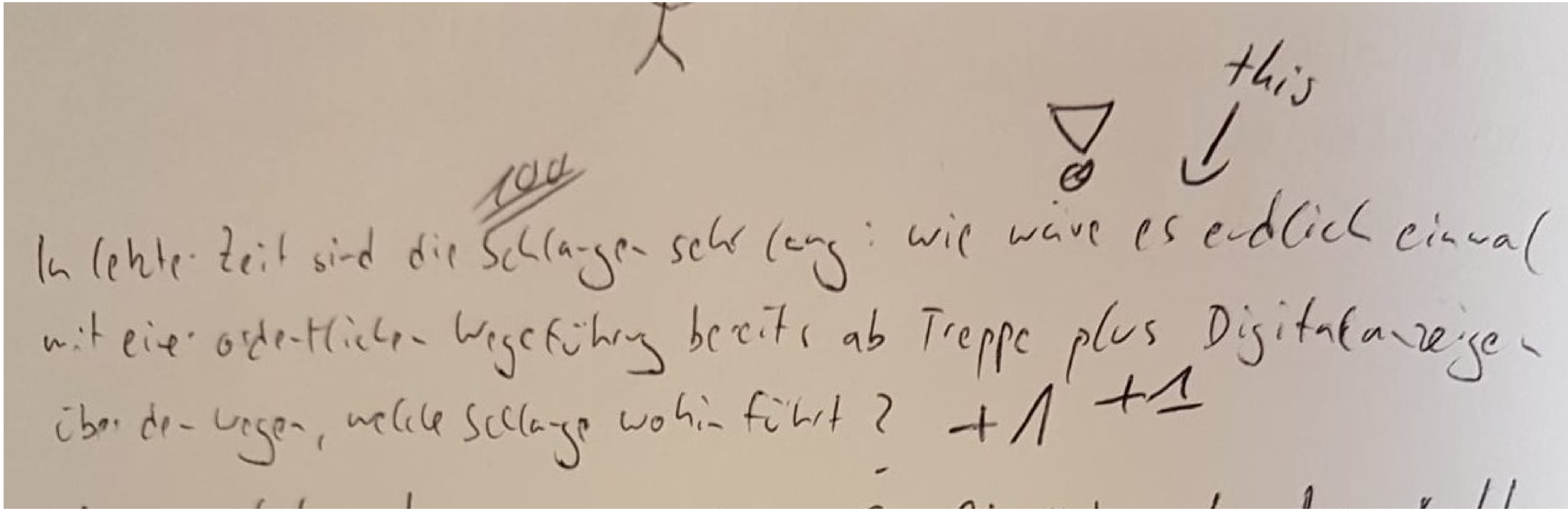
Friedrich-Alexander Universität Erlangen-Nürnberg, Department Mathematik

February 8, 2023

Problemstellung

"Wie kann man den Studentenfluss in der Südmensa optimieren um lange Wartezeiten zu vermeiden?"

Dies ist tatsächlich ein relevantes Problem:



Recap - Zwischenpräsentation

Stand bei Zwischenpräsentation

- Entscheidung für Verkehrssimulation getroffen
- Erste Simulationen
  - Autos fuhren teils unkontrolliert
  - Autos kollidierten
- Erste Aufnahme von Daten zur Analyse

Recap - Zwischenpräsentation

Stand bei Zwischenpräsentation

- Entscheidung für Verkehrssimulation getroffen
- Erste Simulationen
  - Autos fuhren teils unkontrolliert
  - Autos kollidierten
- Erste Aufnahme von Daten zur Analyse

Ziele aus der Zwischenpräsentation

- Implementierung fertigstellen
  - Kollisionen verhindern
- Parameter finden
  - Sensitivitätsanalyse
  - Machbarkeit bewerten

Recap - Zwischenpräsentation

Stand bei Zwischenpräsentation

- Entscheidung für Verkehrssimulation getroffen
- Erste Simulationen
  - Autos fuhren teils unkontrolliert
  - Autos kollidierten
- Erste Aufnahme von Daten zur Analyse

Ziele aus der Zwischenpräsentation

- Implementierung fertigstellen
  - Kollisionen verhindern
- Parameter finden
  - Sensitivitätsanalyse
  - Machbarkeit bewerten

Aktueller Stand

- Implementierung nachhaltig verbessert
  - Zusätzlich bessere Ampeln und Kreuzungen
- Simulation mit auswertbaren Rückgabewerten
  - Analyse erfolgreich
  - Verbesserungspotential entdeckt

Datenerhebung

Datenerhebung

- Erfassung der Studentenzahlen mittels Strichliste

Datenerhebung

- Erfassung der Studentenzahlen mittels Strichliste
- Messungszeitraum: 11:30 - 14:10

Datenerhebung

- Erfassung der Studentenzahlen mittels Strichliste
- Messungszeitraum: 11:30 - 14:10
- Messungsorte: Ost- und Westtreppe, Theken und Kassen

Datenerhebung

- Erfassung der Studentenzahlen mittels Strichliste
- Messungszeitraum: 11:30 - 14:10
- Messungsorte: Ost- und Westtreppe, Theken und Kassen
- Diskretisierung in 5 min Schritten

Datenerhebung

- Erfassung der Studentenzahlen mittels Strichliste
- Messungszeitraum: 11:30 - 14:10
- Messungsorte: Ost- und Westtreppe, Theken und Kassen
- Diskretisierung in 5 min Schritten

von	bis	Kasse					Theke			Essen1	Essen 2	Essen 3	Treppe		
		1	2	3	4	5	1	2	3	4	5	6	Ost	West	Gesamt
11:30	11:35	14	21	20	8	10	9	13	17	2	9	9	24	72	96
11:35	11:40	23	28	26	17	2	19	19	25	6	9	12	31	89	120
11:40	11:45	24	17	23	15	8	14	12	23	5	15	9	16	76	92
11:45	11:50	23	20	28	24		16	9	24	2	12	11	34	66	100
11:50	11:55	18	22	18	23		15	20	17	4	10	10	19	78	97
11:55	12:00	22	18	7	25		15	6	24	4	7	3	27	50	77
12:00	12:05	12	16	13	22		9	20	17	4	9	6	23	53	76
12:05	12:10	19	19	11	14		7	7	21	1	5	5	8	55	63
12:10	12:15	16	13	6	17		16	7	21	4	7	10	32	37	69
12:15	12:20	15	11	11	13		5	9	7	3	7	4	9	24	33
12:20	12:25	1	19	15	15		9	11	13	2	2	5	5	57	62
12:25	12:30	18	22	17	15		15	13	16	2	12	11	27	36	63
12:30	12:35	8	8	8	10		13	8	10	4	1	5	13	35	48
12:35	12:40	9	12	12	18		9	8	13	3	7	2	9	51	60
12:40	12:45	17	1	18	20		16	13	27	9	6	4	28	52	80
12:45	12:50	24	0	12	22		13	9	11	2	11	4	10	42	52
12:50	12:55	12	4	2	11		7	9	6	2	2	3	7	21	28
12:55	13:00	0	8	9	5		4	6	11	0	2	2	15	37	52
13:00	13:05	0	20	11	18		7	8	19	2	6	6	6	42	48
13:05	13:10	0	8	19	15		17	13	13	3	5	2	7	58	65
13:10	13:15	4	11	14	16		10	6	15	3	5	6	55	49	104
13:15	13:20	7	13	10	8		5	14	15	0	9	2	4	36	40
13:20	13:25	7	13	9	11		7	5	4	0	3	2	8	30	38
13:25	13:30	3	0	12	5		5	8	10	0	7	0	8	15	23
13:30	13:35	4	14	2	8		5	3	11	0	6	3	7	33	40
13:35	13:40	0	14	8	13		5	7	12	0	2	2	3	21	24
13:40	13:45	10	13	2	15		14	14	9	0	1	4	5	56	61
13:45	13:50	13	17	10	20		10	18	15	0	13	1	12	84	96
13:50	13:55	23	18	25	23		19	27	34	0	0	0	0	88	88
13:55	14:00	19	18	13	24		32	22	30	0	0	0		71	71
14:00	14:05	8	4	22	19		16	11	11	0	0	0	0	27	27
14:05	14:10	0	0	4	5		0	1	1	0	0	0	0	3	3
		373	422	417	494	20	363	356	502	67	190	143	452	1544	1996

Abb.1: Messungen

Studentenfluss

- Visualisierug der Messergebnisse

Studentenfluss

- Visualisierug der Messergebnisse

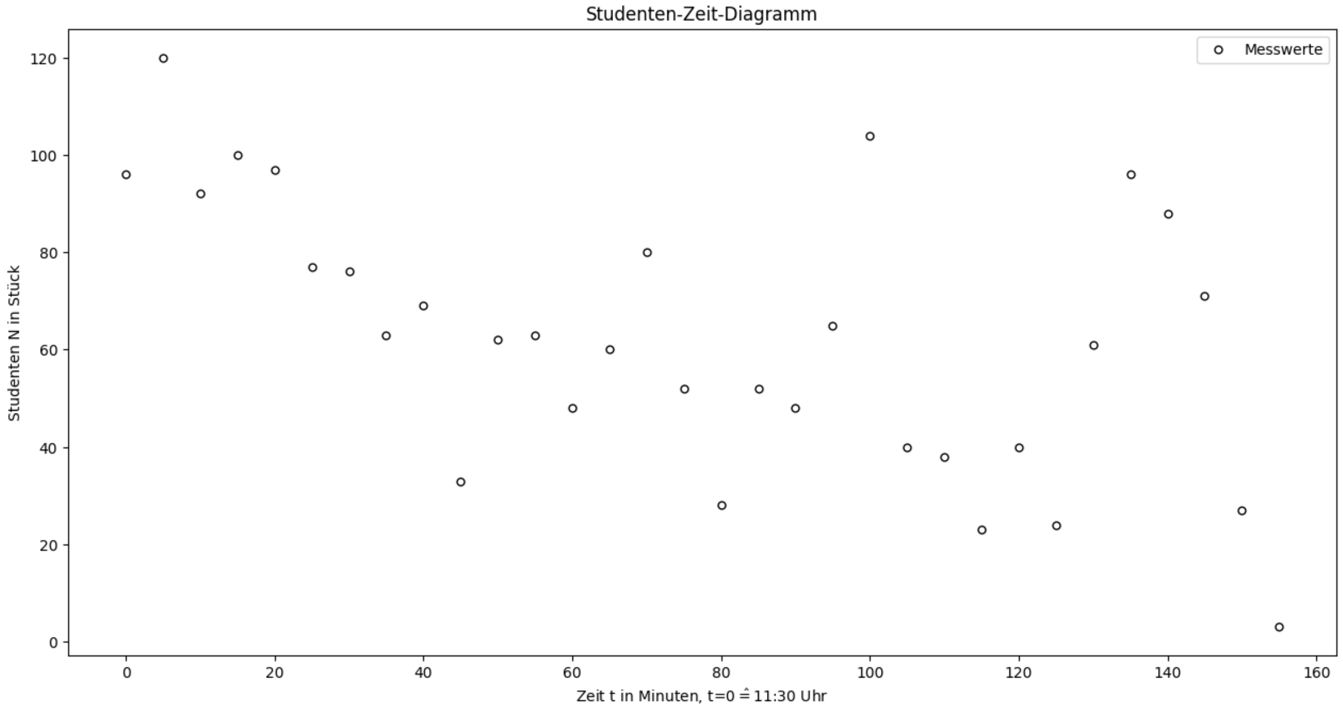


Abb.2: Menge der Stundenten zu den einzelnen Zeitschritten

Studentenfluss

- Approximation einer Funktion durch Polynom 31.Grades

Studentenfluss

- Approximation einer Funktion durch Polynom 31.Grades

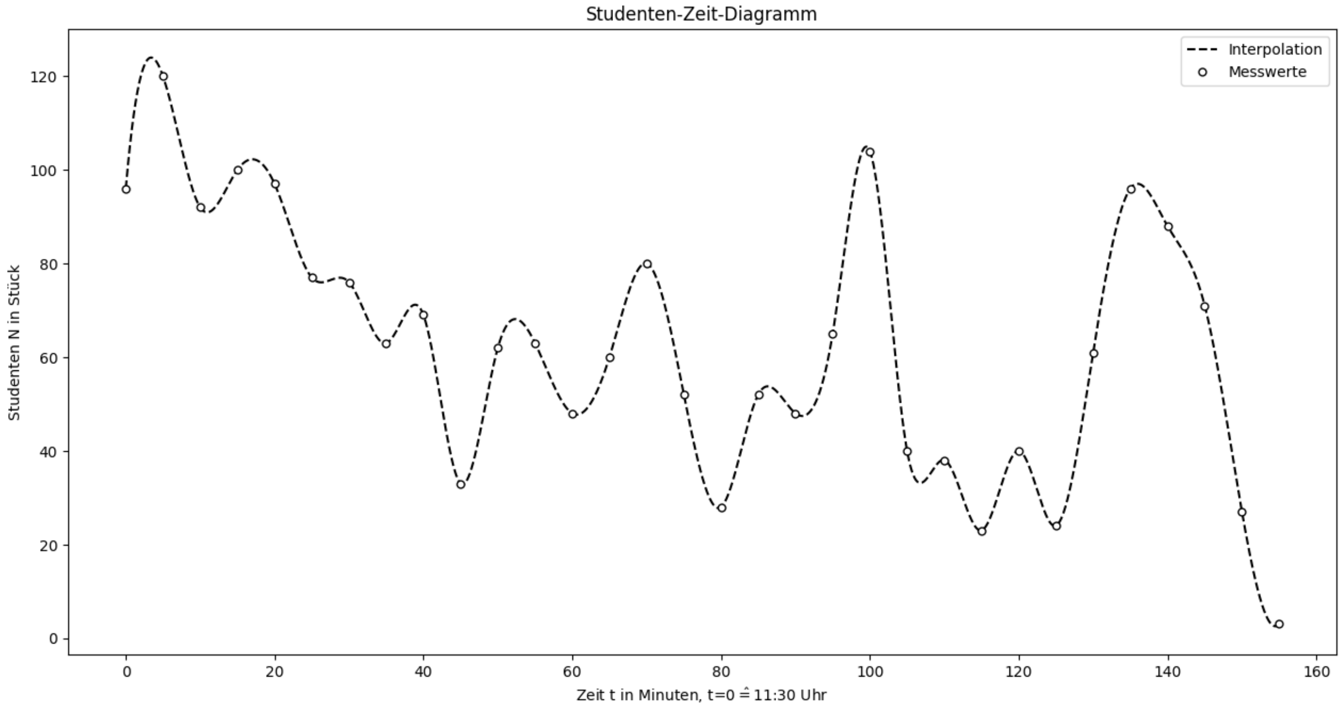


Abb.3: Approximation einer Funktion zur Darstellung der Studenten in der Mensa

Studentenfluss - Abstraktion

Studentenfluss - Abstraktion

- Vereinfachung des Modells anhand der Extrempunkte

Studentenfluss - Abstraktion

- Vereinfachung des Modells anhand der Extrempunkte
- Dadurch realitätsgetreuere Werte

Studentenfluss - Abstraktion

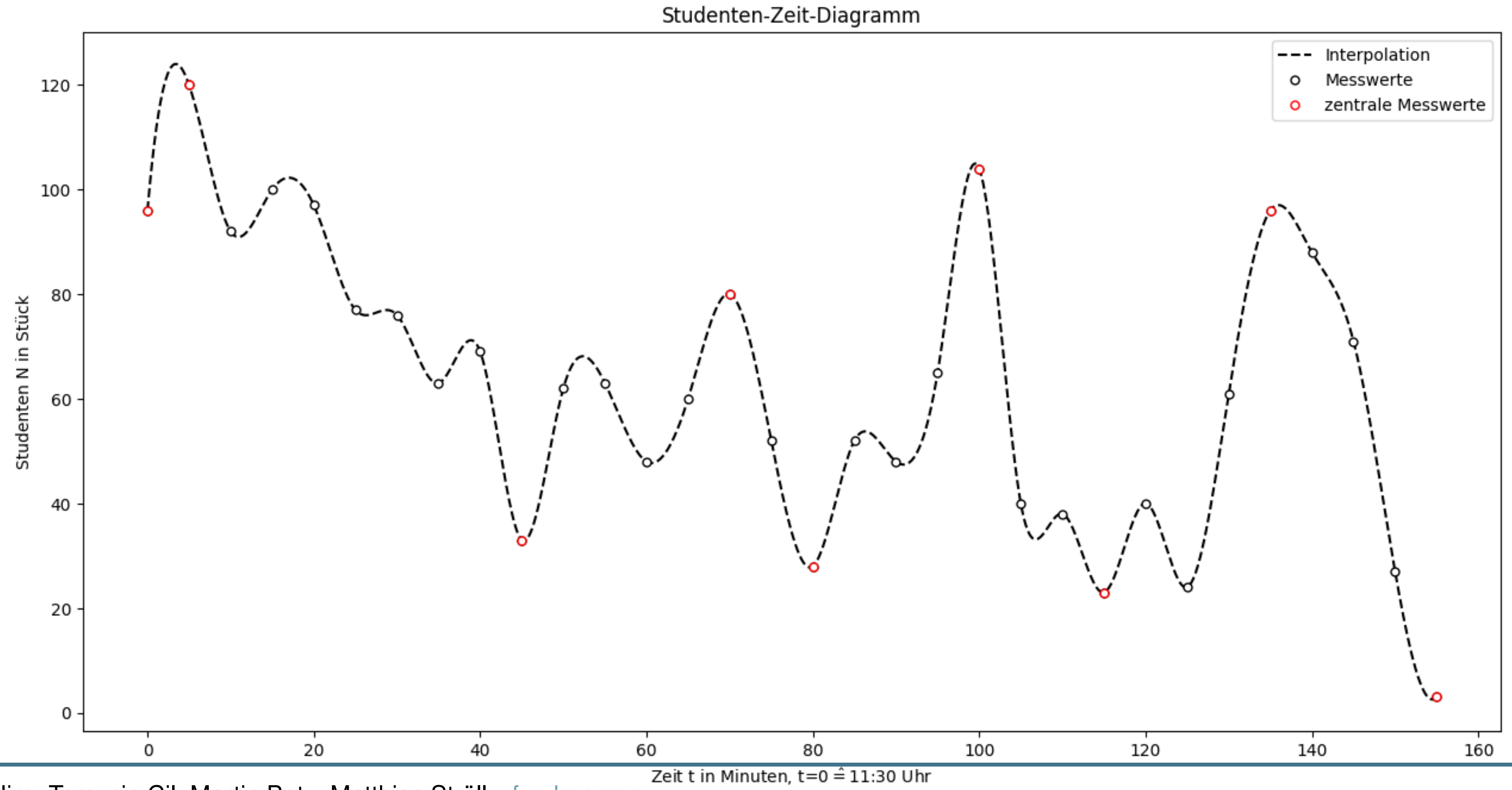
- Vereinfachung des Modells anhand der Extrempunkte
- Dadurch realitätsgetreuere Werte
- Einfacher zu rechnen

Studentenfluss - Abstraktion

- Vereinfachung des Modells anhand der Extrempunkte
- Dadurch realitätsgetreuere Werte
- Einfacher zu rechnen
- Sinnvolle Vereinfachung, da nicht jeder Tag exakt einen solchen Verlauf hat

Studentenfluss - Abstraktion

- Vereinfachung des Modells anhand der Extrempunkte
- Dadurch realitätsgetreuere Werte
- Einfacher zu rechnen
- Sinnvolle Vereinfachung, da nicht jeder Tag exakt einen solchen Verlauf hat



Studentenfluss - Abstraktion

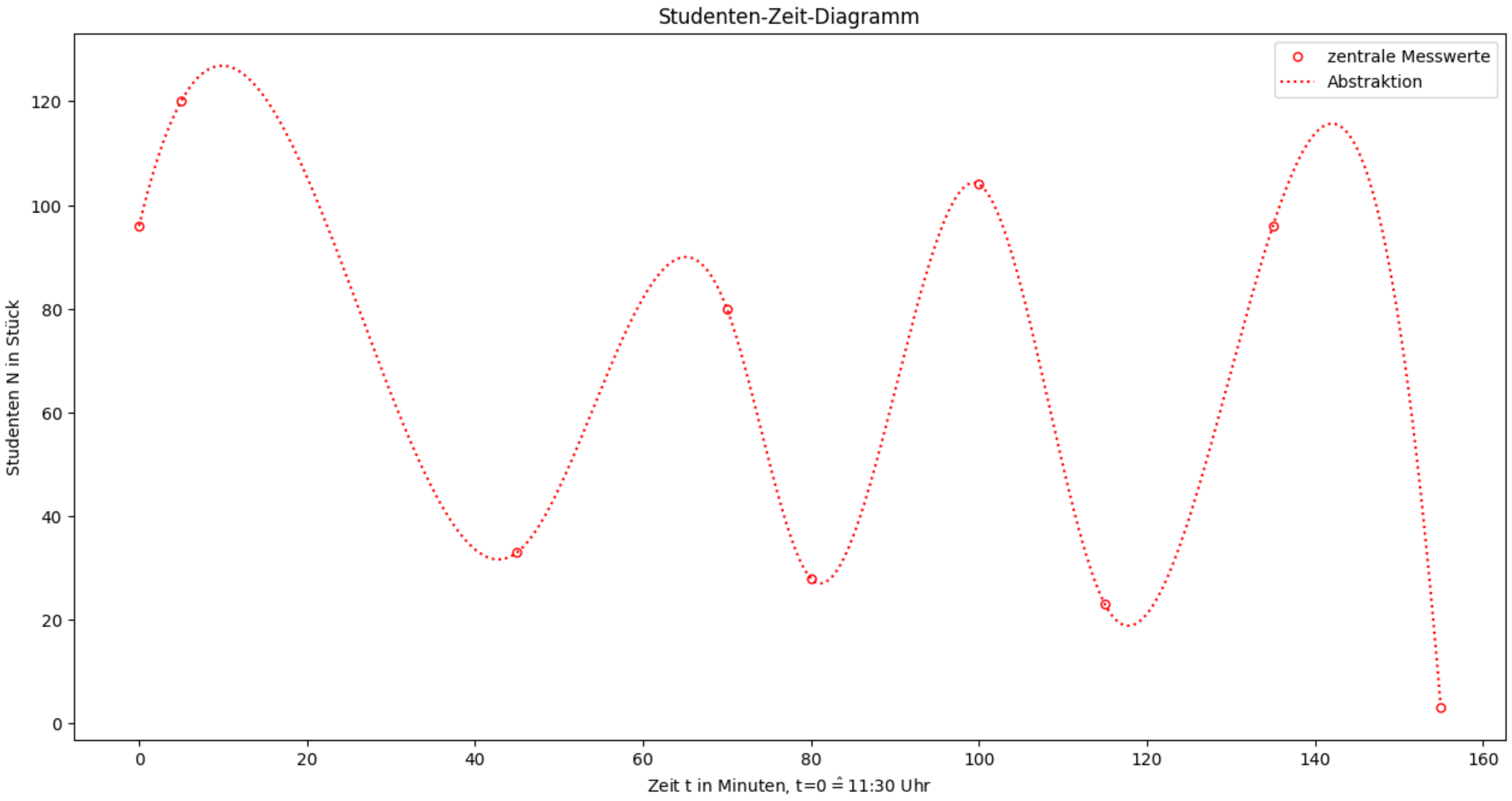


Abb.5: Finale Funktion zur Darstellug von Studenten

Studentenfluss - Abstraktion

- Interpolation mittels scipy:
- $scipy.interpolate := scitp, \quad X := [0, 5, 45, 70, 80, 100, 115, 135, 155], \quad Y := [96, 120, 33, 80, 28, 104, 23, 96, 3]$
- Funktion $\phi(t) := scitp.Bspline\,(scitp.splrep\,(X, Y, s = 0))$

Verteilung auf Theken

Verteilungsfunktion Θ

Verteilung auf Theken

Verteilungsfunktion Θ

- Funktion zur Bestimmung zu welcher Theke ein Student gehen wird, gibt ein 6-Tupel an Wahrscheinlichkeiten für die einzelnen Theken zurück

Verteilung auf Theken

Verteilungsfunktion Θ

- Funktion zur Bestimmung zu welcher Theke ein Student gehen wird, gibt ein 6-Tupel an Wahrscheinlichkeiten für die einzelnen Theken zurück
- $\Theta : [0,1] \times \mathbb{R}^+ \to [0,1]^6$

Verteilung auf Theken

Verteilungsfunktion Θ

- Funktion zur Bestimmung zu welcher Theke ein Student gehen wird, gibt ein 6-Tupel an Wahrscheinlichkeiten für die einzelnen Theken zurück
- $\Theta : [0,1] \times \mathbb{R}^+ \to [0,1]^6$
- Abhängigkeit von Treppengewicht ω, da die Thekenverteilung mit dem Nutzungsverhältnis der Treppen kompatibel sein muss (Theke 1,2,3 $\longleftrightarrow$ Treppe West und Theke 4,5,6 $\longleftrightarrow$ Treppe Ost)

Verteilung auf Theken

Verteilungsfunktion Θ

- Funktion zur Bestimmung zu welcher Theke ein Student gehen wird, gibt ein 6-Tupel an Wahrscheinlichkeiten für die einzelnen Theken zurück
- $\Theta : [0, 1] \times \mathbb{R}^+ \rightarrow [0, 1]^6$
- Abhängigkeit von Treppengewicht ω, da die Thekenverteilung mit dem Nutzungsverhältnis der Treppen kompatibel sein muss $($Theke 1,2,3 $\longleftrightarrow$ Treppe West und Theke 4,5,6 $\longleftrightarrow$ Treppe Ost$)$
- Zeitabhängigkeit, da bei der Auswertung der erhobenen Daten die Thekenverteilung sehr variiert

Verteilung auf Theken

Verteilungsfunktion Θ

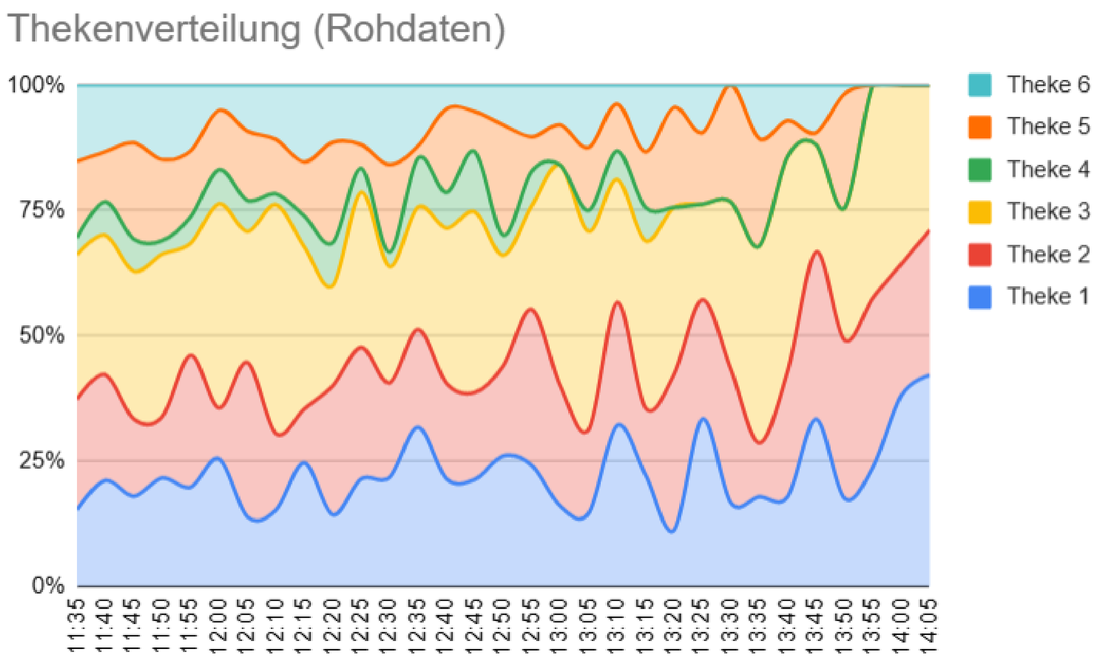
- Funktion zur Bestimmung zu welcher Theke ein Student gehen wird, gibt ein 6-Tupel an Wahrscheinlichkeiten für die einzelnen Theken zurück
- $\Theta : [0,1] \times \mathbb{R}^+ \to [0,1]^6$
- Abhängigkeit von Treppengewicht ω, da die Thekenverteilung mit dem Nutzungsverhältnis der Treppen kompatibel sein muss $($Theke 1,2,3 $\longleftrightarrow$ Treppe West und Theke 4,5,6 $\longleftrightarrow$ Treppe Ost$)$
- Zeitabhängigkeit, da bei der Auswertung der erhobenen Daten die Thekenverteilung sehr variiert
- Zur Modellierung wurden Vereinfachungen getroffen: Unterteilung in 4 Teilintervalle $I_i, i = 1, ..., 4$ mit jeweils konstanten Verteilungen, die aus dem Mittelwert der Rohdaten innerhalb der Intervalle hervorgehen

Verteilung auf Theken

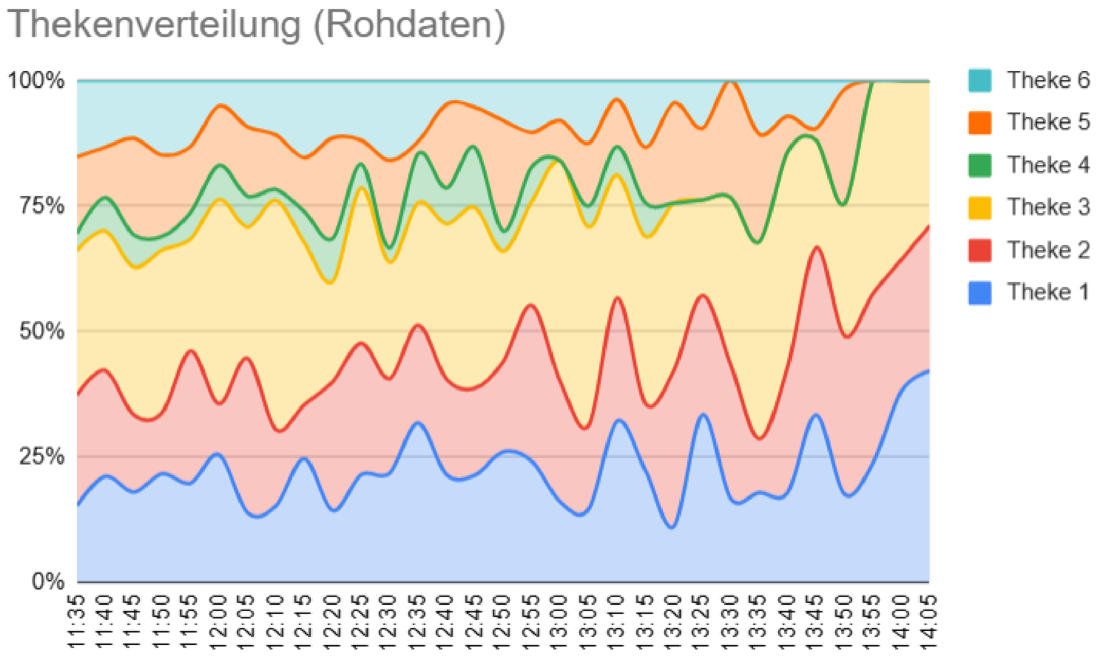
$$\Theta\left(\omega, t\right) = \begin{cases} \left(0.28\omega, 0.26\omega, 0.46\omega, 0.16\left(1-\omega\right), 0.42\left(1-\omega\right), 0.39\left(1-\omega\right)\right)^{\mathsf{T}} & 0 \leq t < 2700 \\ \left(0.31\omega, 0.31\omega, 0.39\omega, 0.21\left(1-\omega\right), 0.43\left(1-\omega\right), 0.36\left(1-\omega\right)\right)^{\mathsf{T}} & 2700 \leq t < 5400 \\ \left(0.29\omega, 0.30\omega, 0.42\omega, 0.09\left(1-\omega\right), 0.61\left(1-\omega\right), 0.35\left(1-\omega\right)\right)^{\mathsf{T}} & 5400 \leq t < 8100 \\ \left(0.24\omega, 0.39\omega, 0.37\omega, 0, 0, 0\right)^{\mathsf{T}} & 8100 \leq t \leq 9600 \\ 0 & \text{sonst} \end{cases} \tag{1}$$

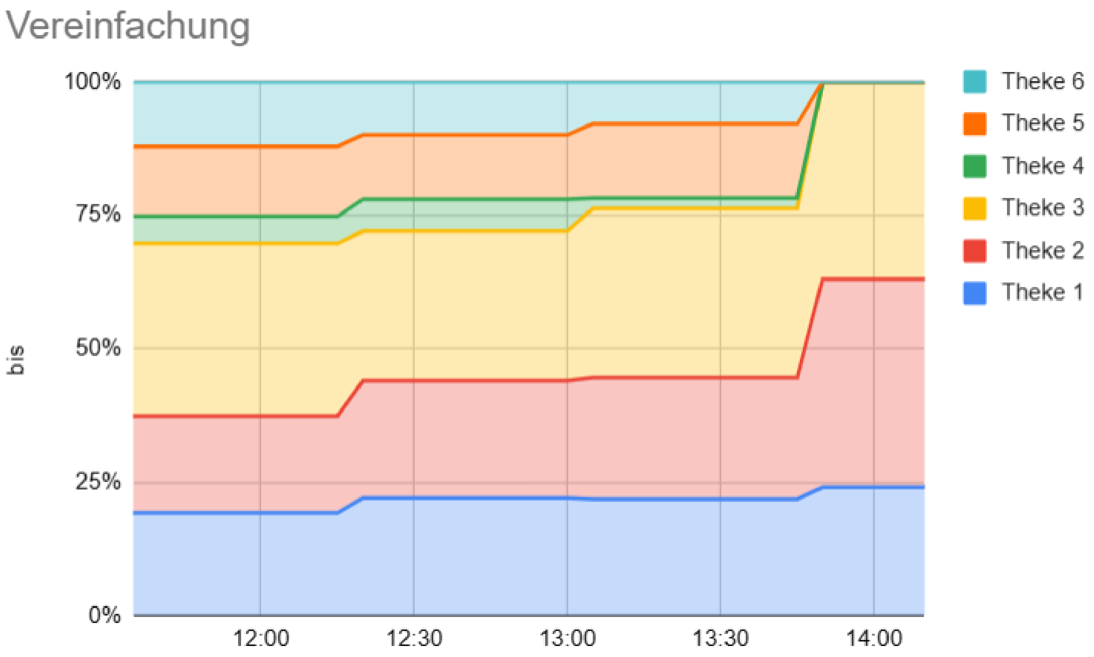
Verteilung auf Theken

Verteilung auf Theken



Verteilung auf Theken



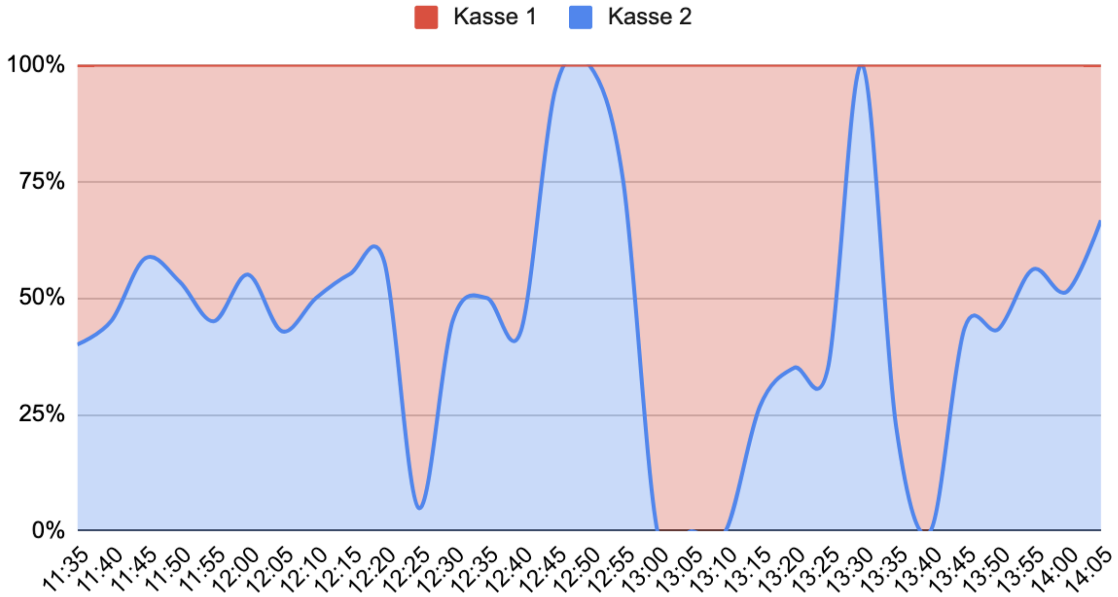


Verteilung auf Kassen

- Annahme: kürzeste Wege $\rightarrow$ "Kassenblöcke"

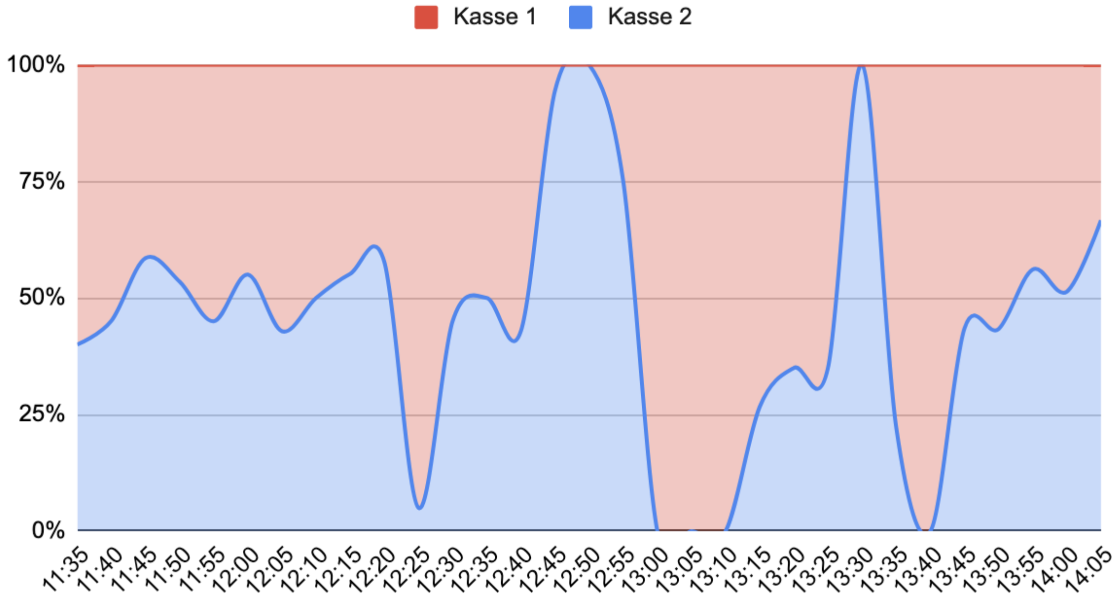
Verteilung auf Kassen

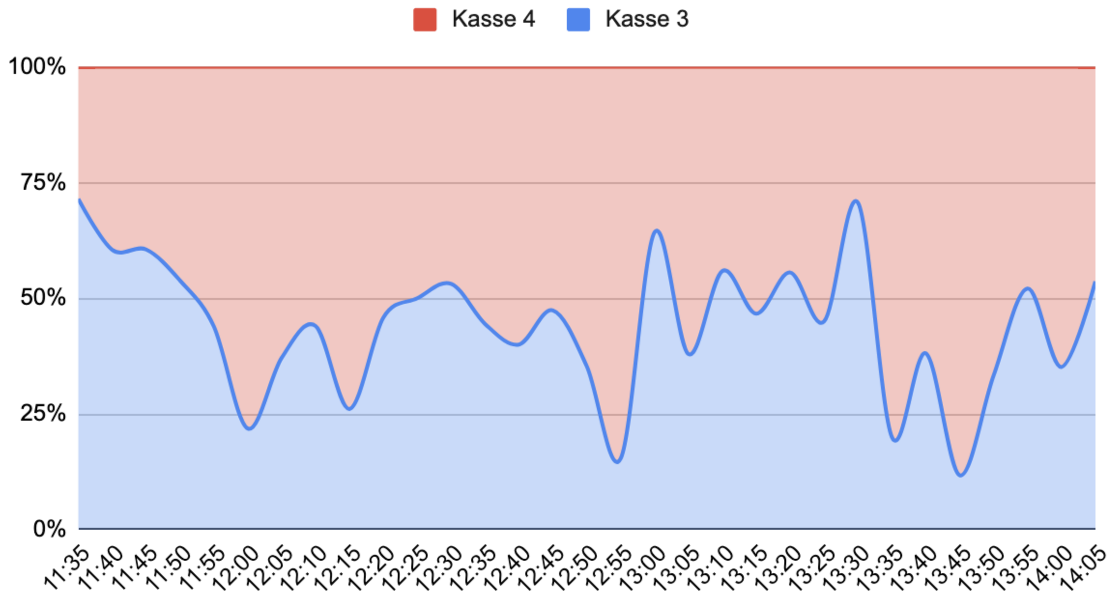
- Annahme: kürzeste Wege → "Kassenblöcke"



Verteilung auf Kassen

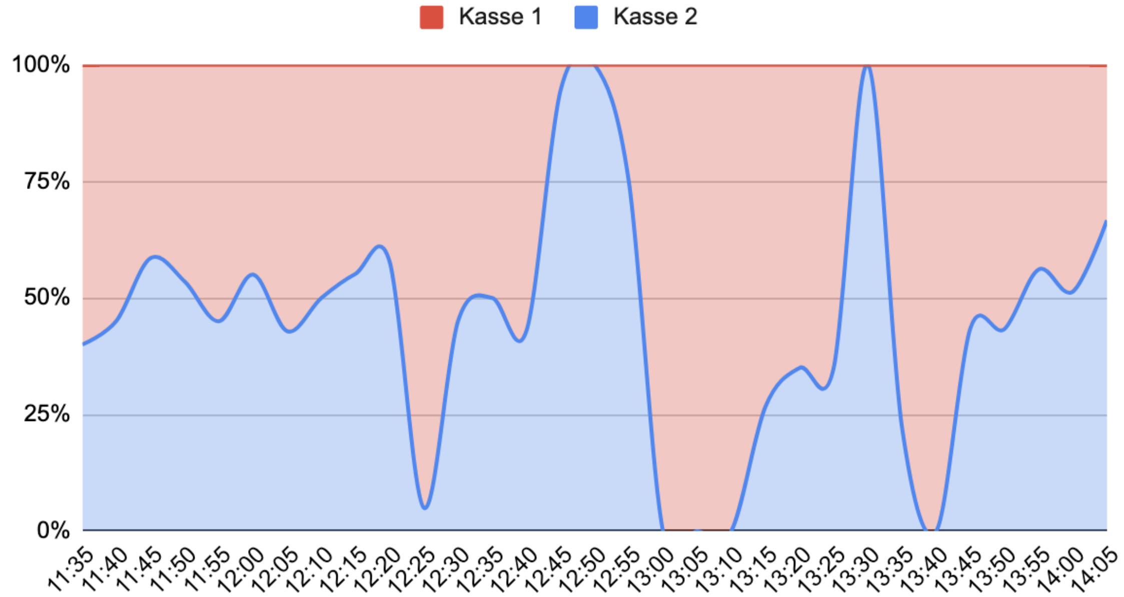
- Annahme: kürzeste Wege → "Kassenblöcke"

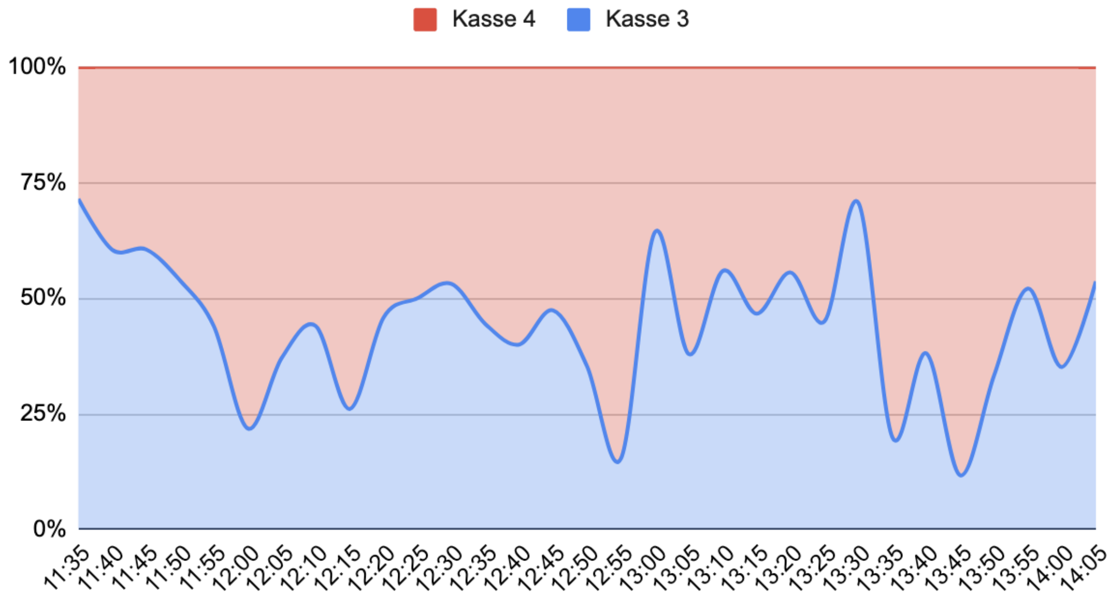




Verteilung auf Kassen

- Annahme: kürzeste Wege $\rightarrow$ "Kassenblöcke"

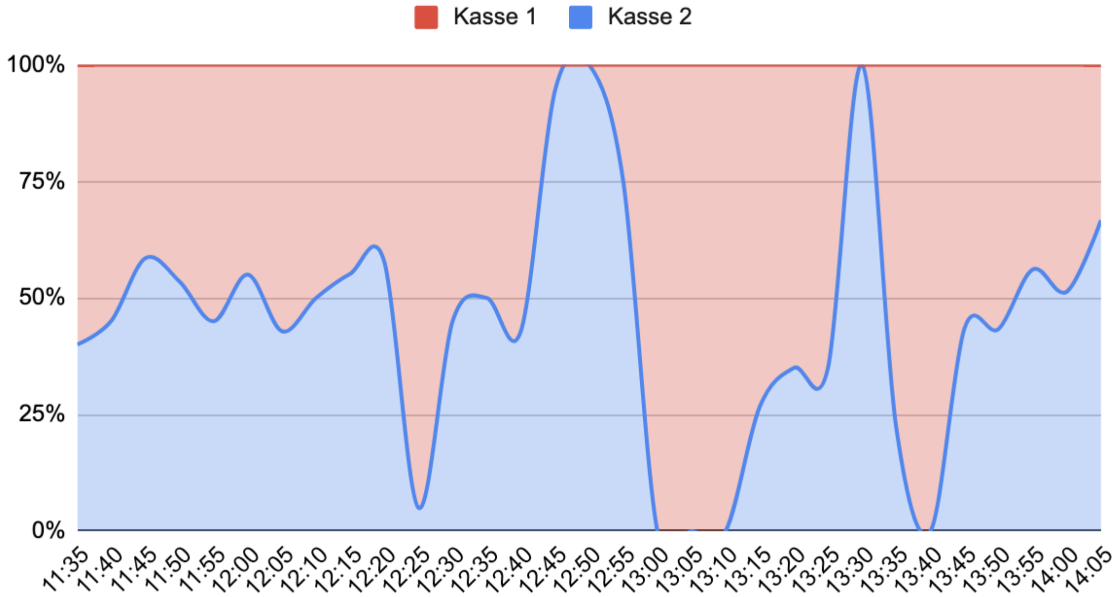


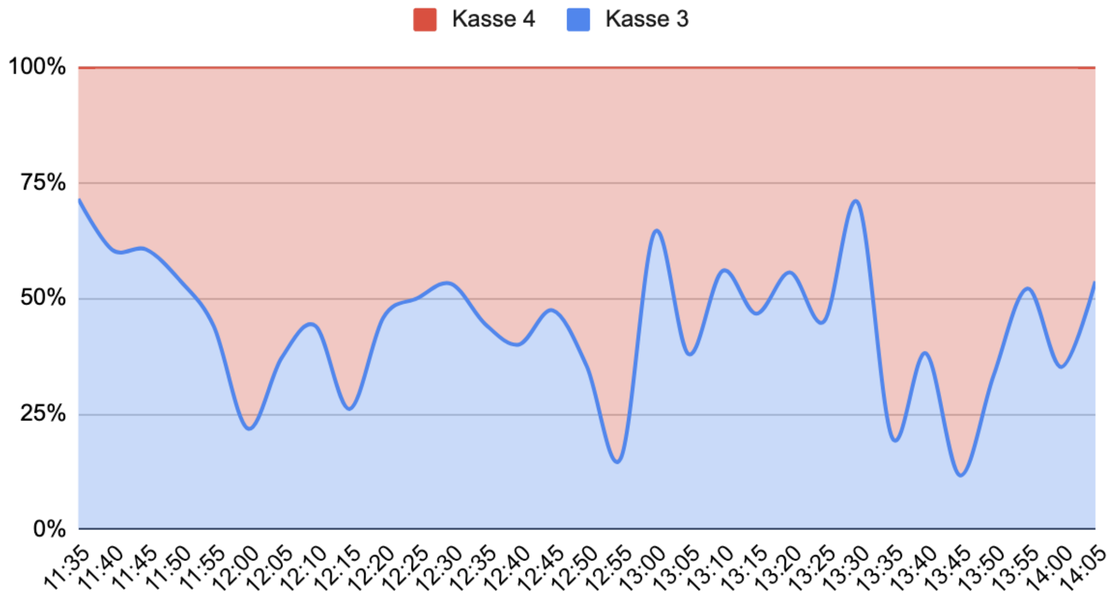


- Starke, zufällige Schwankungen

Verteilung auf Kassen

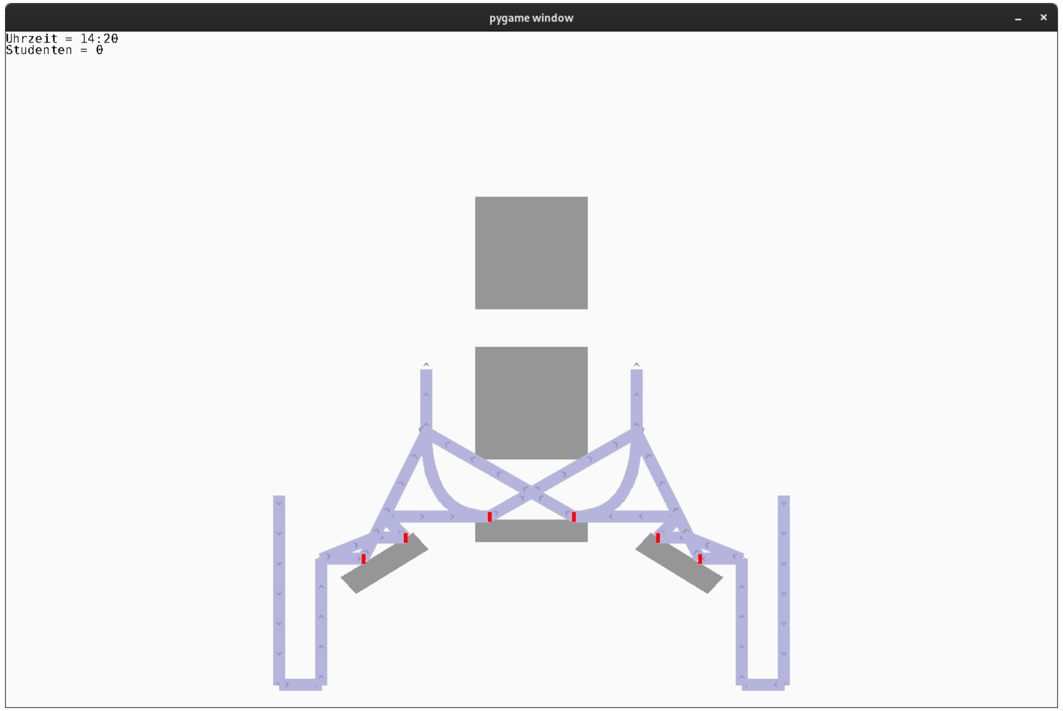
- Annahme: kürzeste Wege → "Kassenblöcke"



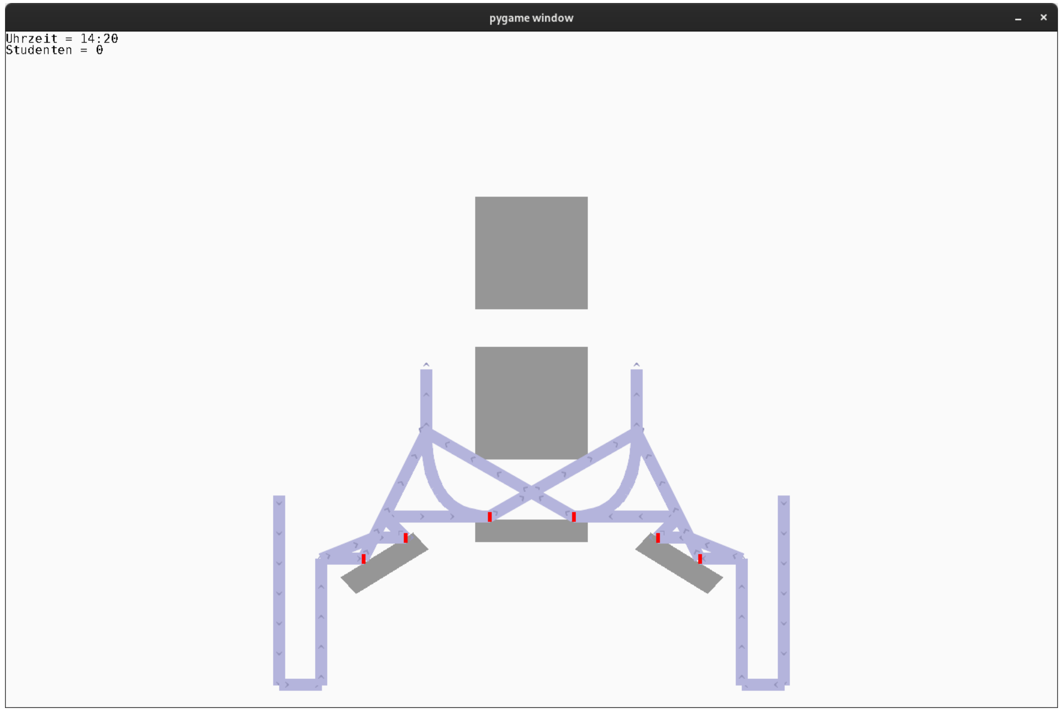


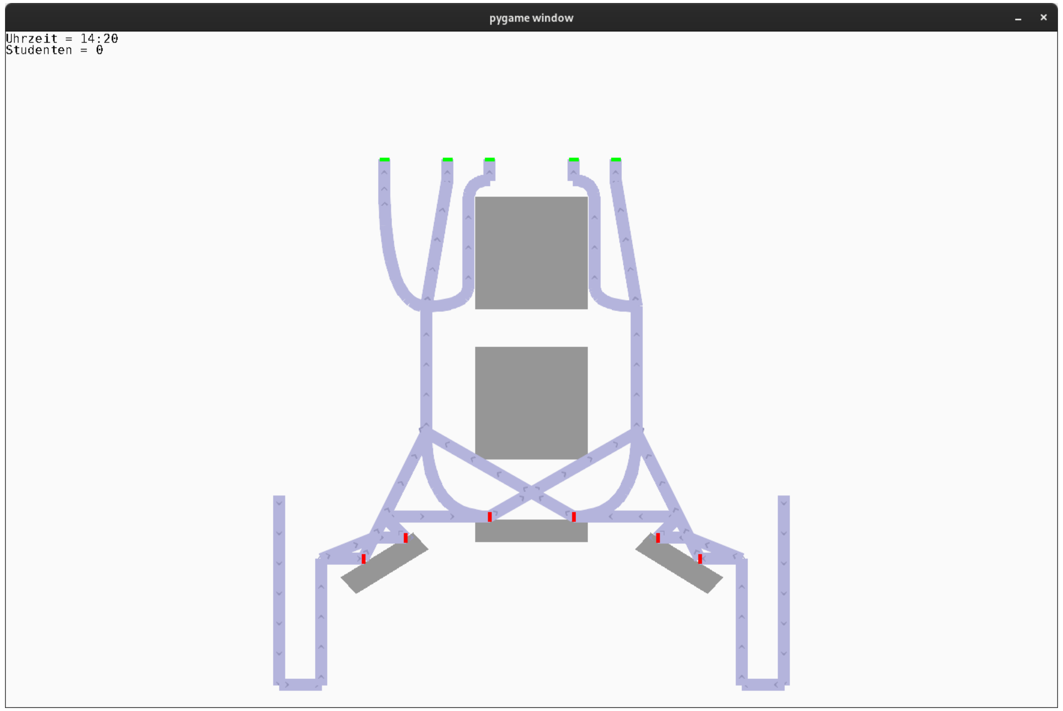
- Starke, zufällige Schwankungen
- → Gleichverteilung innerhalb eines Blocks

Mensa als Straßennetz

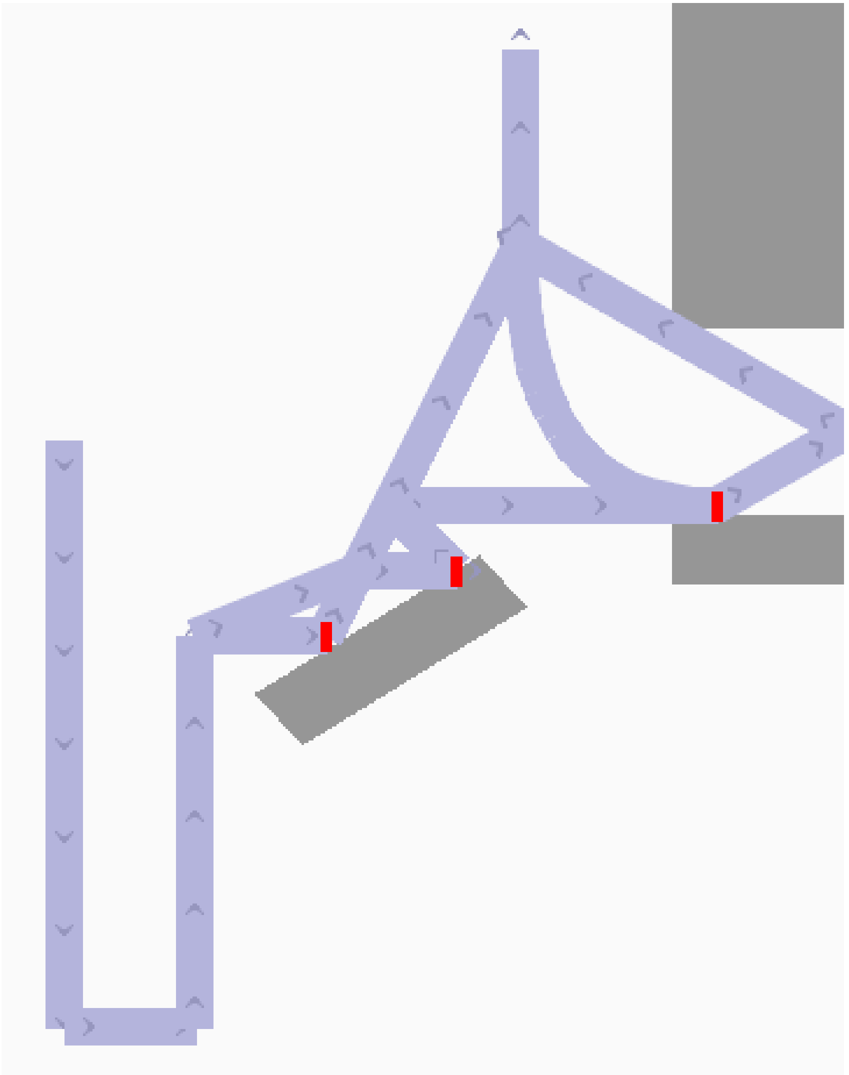


Mensa als Straßennetz



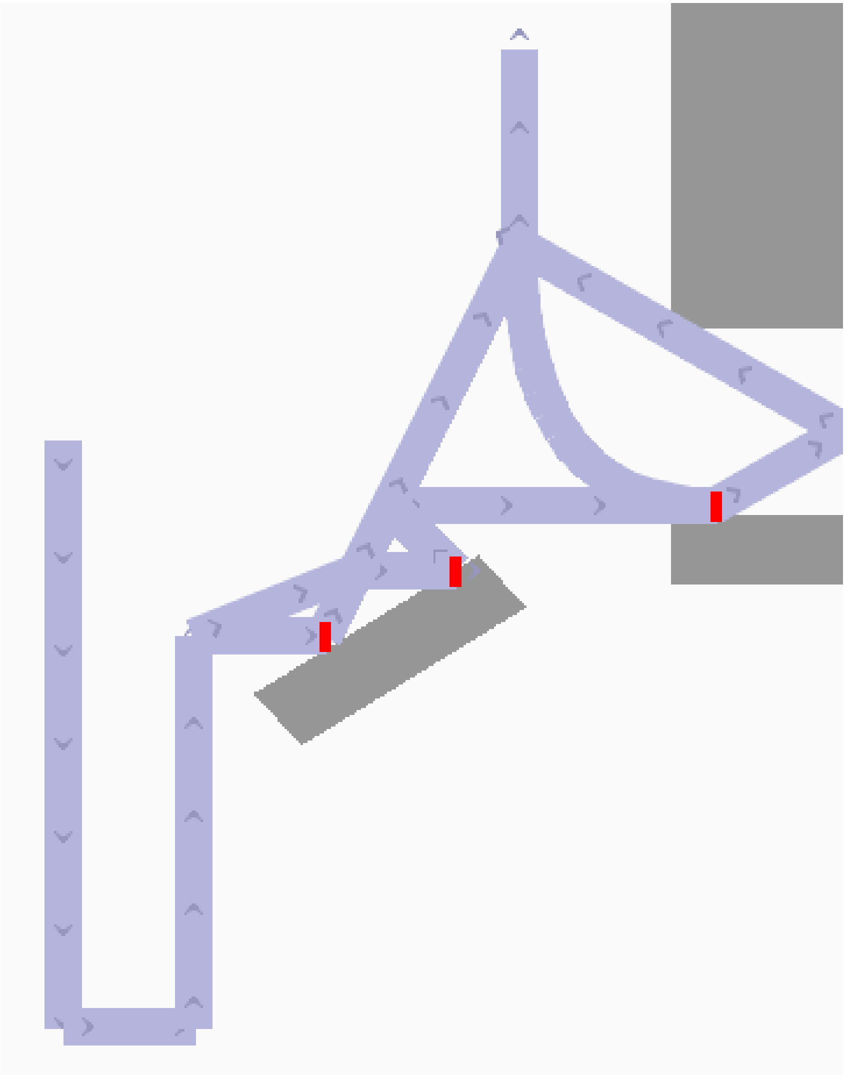


Studentenerzeuger



[weights.essen_3_west, {'path': [0, 1, 2, 4, 7, 8, *road(28), 11]}]

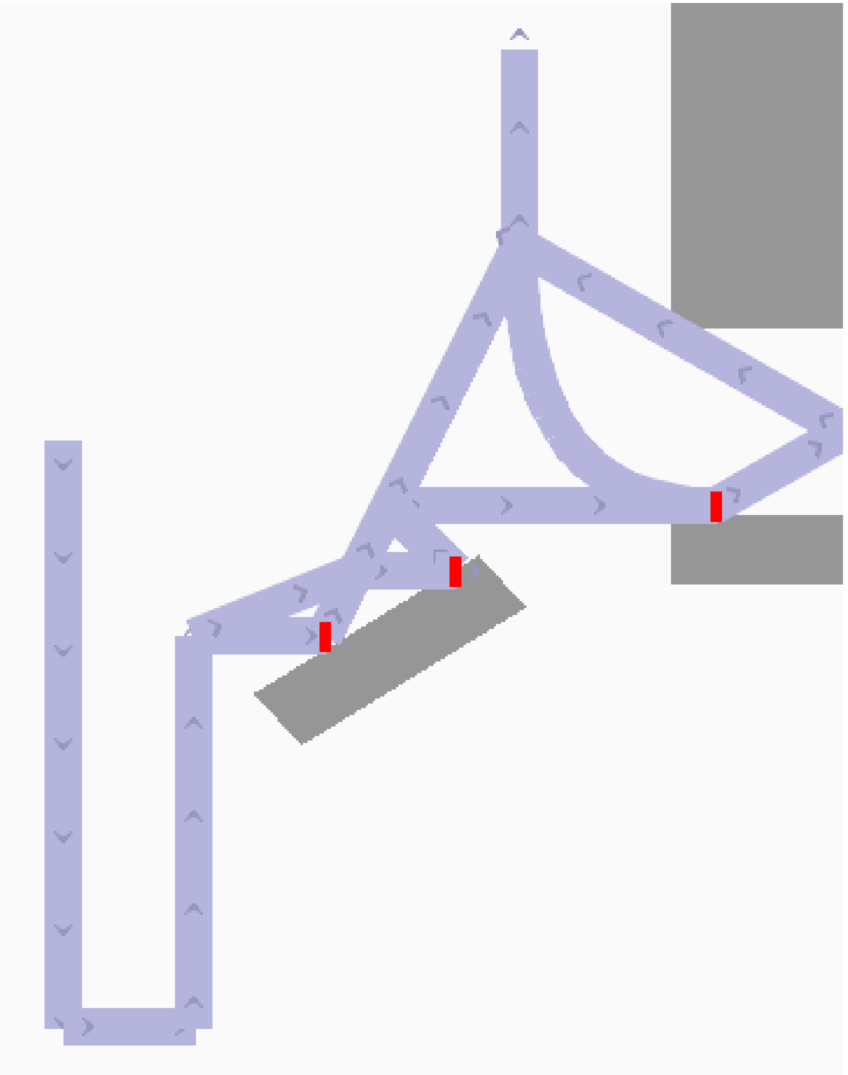
Studentenerzeuger



[weights.essen_3_west, {'path': [0, 1, 2, 4, 7, 8, *road(28), 11]}]

- Codevorlage: konstante Studentenrate

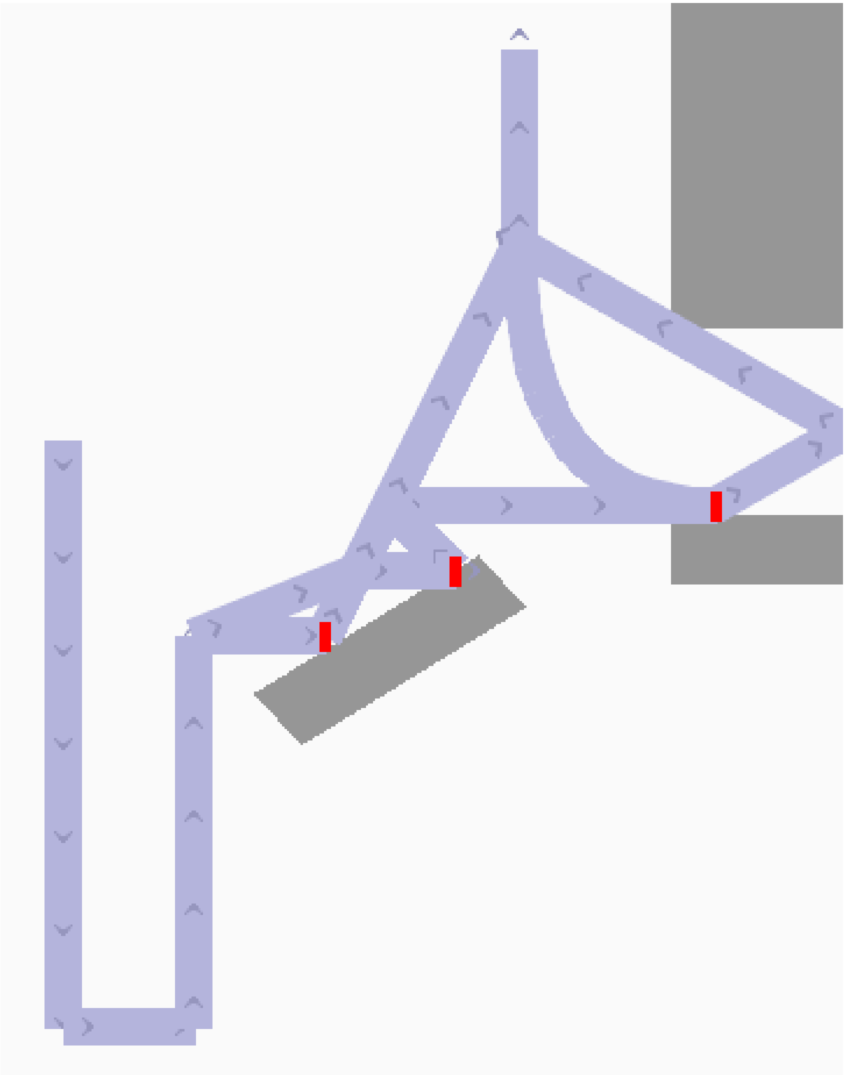
Studentenerzeuger



[weights.essen_3_west, {'path': [0, 1, 2, 4, 7, 8, *road(28), 11]}]

- Codevorlage: konstante Studentenrate
- Studentenflussfunktion $\phi(t)$ übergeben und diese als zeitabhängige Studentenrate nutzen

Studentenerzeuger



[weights.essen_3_west, {'path': [0, 1, 2, 4, 7, 8, *road(28), 11]}]

- Codevorlage: konstante Studentenrate
- Studentenflussfunktion $\phi(t)$ übergeben und diese als zeitabhängige Studentenrate nutzen
- Verteilungsfunktion $\Theta(\omega, t)$ zur Spezifizierung der einzelnen Studenten nutzen

Kreuzungen

```python
# Check merging road
if first.current_road_index < len(first.path) - 1:
    next_road = self.sim.roads[first.path[first.current_road_index + 1]]
    if next_road.is_merging and (first.x >= self.length - self.intersection_slow_distance):

        # add to waiting queue
        if next_road.merging_queue.count(first) == 0:
            next_road.merging_queue.append(first)

        # slow down if not first
        if next_road.merging_queue.index(first) != 0:
            factor = (self.length - first.x)/(self.intersection_slow_distance) \
                        * self.intersection_slow_factor
            first.slow(factor * first._v_max)
            if first.x >= self.length - self.intersection_stop_distance and\
                first.x <= self.length - self.intersection_stop_distance / 2:
                # Stop vehicles in the stop zone
                first.stop()
        else:
            first.unstop()
            for vehicle in self.vehicles:
                vehicle.unslow()
```

Warum Kreuzungen?

Implementierung: Queue

Kreuzungen

```python
# Check merging road
if first.current_road_index < len(first.path) - 1:
    next_road = self.sim.roads[first.path[first.current_road_index + 1]]
    if next_road.is_merging and (first.x >= self.length - self.intersection_slow_distance):

        # add to waiting queue
        if next_road.merging_queue.count(first) == 0:
            next_road.merging_queue.append(first)

        # slow down if not first
        if next_road.merging_queue.index(first) != 0:
            factor = (self.length - first.x)/(self.intersection_slow_distance) \
                        * self.intersection_slow_factor
            first.slow(factor * first._v_max)
            if first.x >= self.length - self.intersection_stop_distance and\
                first.x <= self.length - self.intersection_stop_distance / 2:
                # Stop vehicles in the stop zone
                first.stop()
        else:
            first.unstop()
            for vehicle in self.vehicles:
                vehicle.unslow()
```

Warum Kreuzungen?

- Problem: Studentenrouten schneiden sich

Implementierung: Queue

Kreuzungen

```python
# Check merging road
if first.current_road_index < len(first.path) - 1:
    next_road = self.sim.roads[first.path[first.current_road_index + 1]]
    if next_road.is_merging and (first.x >= self.length - self.intersection_slow_distance):

        # add to waiting queue
        if next_road.merging_queue.count(first) == 0:
            next_road.merging_queue.append(first)

        # slow down if not first
        if next_road.merging_queue.index(first) != 0:
            factor = (self.length - first.x)/(self.intersection_slow_distance) \
                        * self.intersection_slow_factor
            first.slow(factor * first._v_max)
            if first.x >= self.length - self.intersection_stop_distance and\
                first.x <= self.length - self.intersection_stop_distance / 2:
                # Stop vehicles in the stop zone
                first.stop()
        else:
            first.unstop()
            for vehicle in self.vehicles:
                vehicle.unslow()
```

Warum Kreuzungen?

- Problem: Studentenrouten schneiden sich
- Zusammenstöße sollen vermieden werden

Implementierung: Queue

Kreuzungen

```python
# Check merging road
if first.current_road_index < len(first.path) - 1:
    next_road = self.sim.roads[first.path[first.current_road_index + 1]]
    if next_road.is_merging and (first.x >= self.length - self.intersection_slow_distance):

        # add to waiting queue
        if next_road.merging_queue.count(first) == 0:
            next_road.merging_queue.append(first)

        # slow down if not first
        if next_road.merging_queue.index(first) != 0:
            factor = (self.length - first.x)/(self.intersection_slow_distance) \
                        * self.intersection_slow_factor
            first.slow(factor * first._v_max)
            if first.x >= self.length - self.intersection_stop_distance and\
                first.x <= self.length - self.intersection_stop_distance / 2:
                # Stop vehicles in the stop zone
                first.stop()
        else:
            first.unstop()
            for vehicle in self.vehicles:
                vehicle.unslow()
```

Warum Kreuzungen?

- Problem: Studentenrouten schneiden sich
- Zusammenstöße sollen vermieden werden
- Lösung: Schnittstelle wird zur Kreuzung

Implementierung: Queue

Kreuzungen

```python
# Check merging road
if first.current_road_index < len(first.path) - 1:
    next_road = self.sim.roads[first.path[first.current_road_index + 1]]
    if next_road.is_merging and (first.x >= self.length - self.intersection_slow_distance):

        # add to waiting queue
        if next_road.merging_queue.count(first) == 0:
            next_road.merging_queue.append(first)

        # slow down if not first
        if next_road.merging_queue.index(first) != 0:
            factor = (self.length - first.x)/(self.intersection_slow_distance) \
                        * self.intersection_slow_factor
            first.slow(factor * first._v_max)
            if first.x >= self.length - self.intersection_stop_distance and\
                first.x <= self.length - self.intersection_stop_distance / 2:
                # Stop vehicles in the stop zone
                first.stop()
        else:
            first.unstop()
            for vehicle in self.vehicles:
                vehicle.unslow()
```

Warum Kreuzungen?

- Problem: Studentenrouten schneiden sich
- Zusammenstöße sollen vermieden werden
- Lösung: Schnittstelle wird zur Kreuzung

Implementierung: Queue

- Studenten können immer aufgenommen werden

Kreuzungen

```python
# Check merging road
if first.current_road_index < len(first.path) - 1:
    next_road = self.sim.roads[first.path[first.current_road_index + 1]]
    if next_road.is_merging and (first.x >= self.length - self.intersection_slow_distance):

        # add to waiting queue
        if next_road.merging_queue.count(first) == 0:
            next_road.merging_queue.append(first)

        # slow down if not first
        if next_road.merging_queue.index(first) != 0:
            factor = (self.length - first.x)/(self.intersection_slow_distance) \
                        * self.intersection_slow_factor
            first.slow(factor * first._v_max)
            if first.x >= self.length - self.intersection_stop_distance and\
                first.x <= self.length - self.intersection_stop_distance / 2:
                # Stop vehicles in the stop zone
                first.stop()
        else:
            first.unstop()
            for vehicle in self.vehicles:
                vehicle.unslow()
```

Warum Kreuzungen?

- Problem: Studentenrouten schneiden sich
- Zusammenstöße sollen vermieden werden
- Lösung: Schnittstelle wird zur Kreuzung

Implementierung: Queue

- Studenten können immer aufgenommen werden
- Vorderster Student darf die Kreuzung passieren und wird aus der Queue gelöscht

Kreuzungen

```python
# Check merging road
if first.current_road_index < len(first.path) - 1:
    next_road = self.sim.roads[first.path[first.current_road_index + 1]]
    if next_road.is_merging and (first.x >= self.length - self.intersection_slow_distance):

        # add to waiting queue
        if next_road.merging_queue.count(first) == 0:
            next_road.merging_queue.append(first)

        # slow down if not first
        if next_road.merging_queue.index(first) != 0:
            factor = (self.length - first.x)/(self.intersection_slow_distance) \
                        * self.intersection_slow_factor
            first.slow(factor * first._v_max)
            if first.x >= self.length - self.intersection_stop_distance and\
                first.x <= self.length - self.intersection_stop_distance / 2:
                # Stop vehicles in the stop zone
                first.stop()
        else:
            first.unstop()
            for vehicle in self.vehicles:
                vehicle.unslow()
```

Warum Kreuzungen?

- Problem: Studentenrouten schneiden sich
- Zusammenstöße sollen vermieden werden
- Lösung: Schnittstelle wird zur Kreuzung

Implementierung: Queue

- Studenten können immer aufgenommen werden
- Vorderster Student darf die Kreuzung passieren und wird aus der Queue gelöscht
- Nach kurzer Wartezeit ist der nächste Student dran

Ampeln

```python
    def update(self, sim):
        if self.fixed_cycle:
            cycle_length = 30
            k = (sim.t // cycle_length) % 2
            self.current_cycle_index = int(k)
        else:
            self.delay = max(0, self.delay - 1)
            if self.delay > 0:
                self.current_cycle_index = 0
            else:
                self.current_cycle_index = 1

    def increment(self):
        self.passed_cars += 1
        self.delay += self.cycle_delay # self.cycle_delay = 150
        if self.passed_cars % 6 == 0:
            self.delay += 300
```

Ampeln

```python
    def update(self, sim):
        if self.fixed_cycle:
            cycle_length = 30
            k = (sim.t // cycle_length) % 2
            self.current_cycle_index = int(k)
        else:
            self.delay = max(0, self.delay - 1)
            if self.delay > 0:
                self.current_cycle_index = 0
            else:
                self.current_cycle_index = 1

    def increment(self):
        self.passed_cars += 1
        self.delay += self.cycle_delay # self.cycle_delay = 150
        if self.passed_cars % 6 == 0:
            self.delay += 300
```

- Modellieren Essensausgabe und Kassen

Ampeln

```python
    def update(self, sim):
        if self.fixed_cycle:
            cycle_length = 30
            k = (sim.t // cycle_length) % 2
            self.current_cycle_index = int(k)
        else:
            self.delay = max(0, self.delay - 1)
            if self.delay > 0:
                self.current_cycle_index = 0
            else:
                self.current_cycle_index = 1

    def increment(self):
        self.passed_cars += 1
        self.delay += self.cycle_delay # self.cycle_delay = 150
        if self.passed_cars % 6 == 0:
            self.delay += 300
```

- Modellieren Essensausgabe und Kassen
- Kurze Rotphase nach jedem Studenten

Ampeln

```python
    def update(self, sim):
        if self.fixed_cycle:
            cycle_length = 30
            k = (sim.t // cycle_length) % 2
            self.current_cycle_index = int(k)
        else:
            self.delay = max(0, self.delay - 1)
            if self.delay > 0:
                self.current_cycle_index = 0
            else:
                self.current_cycle_index = 1

    def increment(self):
        self.passed_cars += 1
        self.delay += self.cycle_delay # self.cycle_delay = 150
        if self.passed_cars % 6 == 0:
            self.delay += 300
```

- Modellieren Essensausgabe und Kassen
- Kurze Rotphase nach jedem Studenten
- Thekenampeln: Alle 6 Studenten kommt es zu einer längeren Rotphase

Parameter

Zahlreiche Faktoren beeinflussen das Geschehen in der Mensa

Parameter

Zahlreiche Faktoren beeinflussen das Geschehen in der Mensa

Unveränderliche Faktoren

Parameter

Zahlreiche Faktoren beeinflussen das Geschehen in der Mensa

Unveränderliche Faktoren

- Grundriss

Parameter

Zahlreiche Faktoren beeinflussen das Geschehen in der Mensa

Unveränderliche Faktoren

- Grundriss
- Studentenzahl und zeitliche Aufteilung

Parameter

Zahlreiche Faktoren beeinflussen das Geschehen in der Mensa

Unveränderliche Faktoren

- Grundriss
- Studentenzahl und zeitliche Aufteilung
- Beliebtheit einzelner Gerichte

Parameter

Zahlreiche Faktoren beeinflussen das Geschehen in der Mensa

Unveränderliche Faktoren

- Grundriss
- Studentenzahl und zeitliche Aufteilung
- Beliebtheit einzelner Gerichte
- Geschwindigkeit der Essensausgabe

Parameter

Zahlreiche Faktoren beeinflussen das Geschehen in der Mensa

Unveränderliche Faktoren

- Grundriss
- Studentenzahl und zeitliche Aufteilung
- Beliebtheit einzelner Gerichte
- Geschwindigkeit der Essensausgabe

Optimierbare Faktoren

Parameter

Zahlreiche Faktoren beeinflussen das Geschehen in der Mensa

Unveränderliche Faktoren

- Grundriss
- Studentenzahl und zeitliche Aufteilung
- Beliebtheit einzelner Gerichte
- Geschwindigkeit der Essensausgabe

Optimierbare Faktoren

- Treppengewichte

Parameter

Zahlreiche Faktoren beeinflussen das Geschehen in der Mensa

Unveränderliche Faktoren

- Grundriss
- Studentenzahl und zeitliche Aufteilung
- Beliebtheit einzelner Gerichte
- Geschwindigkeit der Essensausgabe

Optimierbare Faktoren

- Treppengewichte
- Geschwindigkeit der Studenten

Parameter

Zahlreiche Faktoren beeinflussen das Geschehen in der Mensa

Unveränderliche Faktoren

- Grundriss
- Studentenzahl und zeitliche Aufteilung
- Beliebtheit einzelner Gerichte
- Geschwindigkeit der Essensausgabe

Optimierbare Faktoren

- Treppengewichte
- Geschwindigkeit der Studenten
- Verteilung der Gerichte auf die Theken

Minimierungsproblem

Wartezeitfunktion

Minimierungsproblem

Wartezeitfunktion

- Minimiere W $(v, \omega) = \frac{1}{\sqrt{N}} \cdot \sqrt{\sum_{i=1}^{N} \left(T_i\left(v, \omega\right)\right)^2}$

Minimierungsproblem

Wartezeitfunktion

- Minimiere W $(v, \omega) = \frac{1}{\sqrt{N}} \cdot \sqrt{\sum_{i=1}^{N} \left(T_i\left(v, \omega\right)\right)^2}$
- Maximale Studentengeschwindigkeit $v \in [8, 20]$, ungefähr 4 bis 11 $\frac{\text{km}}{\text{h}}$

Minimierungsproblem

Wartezeitfunktion

- Minimiere W $(v, \omega) = \frac{1}{\sqrt{N}} \cdot \sqrt{\sum_{i=1}^{N} \left(T_i\left(v, \omega\right)\right)^2}$
- Maximale Studentengeschwindigkeit $v \in [8, 20]$, ungefähr 4 bis 11 $\frac{\text{km}}{\text{h}}$
- Zeit $t \in I = [0, 10\,200] \cap \mathbb{N}_0$

Minimierungsproblem

Wartezeitfunktion

- Minimiere W $(v, \omega) = \frac{1}{\sqrt{N}} \cdot \sqrt{\sum_{i=1}^{N} \left(T_i\left(v, \omega\right)\right)^2}$
- Maximale Studentengeschwindigkeit $v \in [8, 20]$, ungefähr 4 bis 11 $\frac{\text{km}}{\text{h}}$
- Zeit $t \in I = [0, 10\,200] \cap \mathbb{N}_0$
- Treppengewicht $\omega \in [0, 1]$

Minimierungsproblem

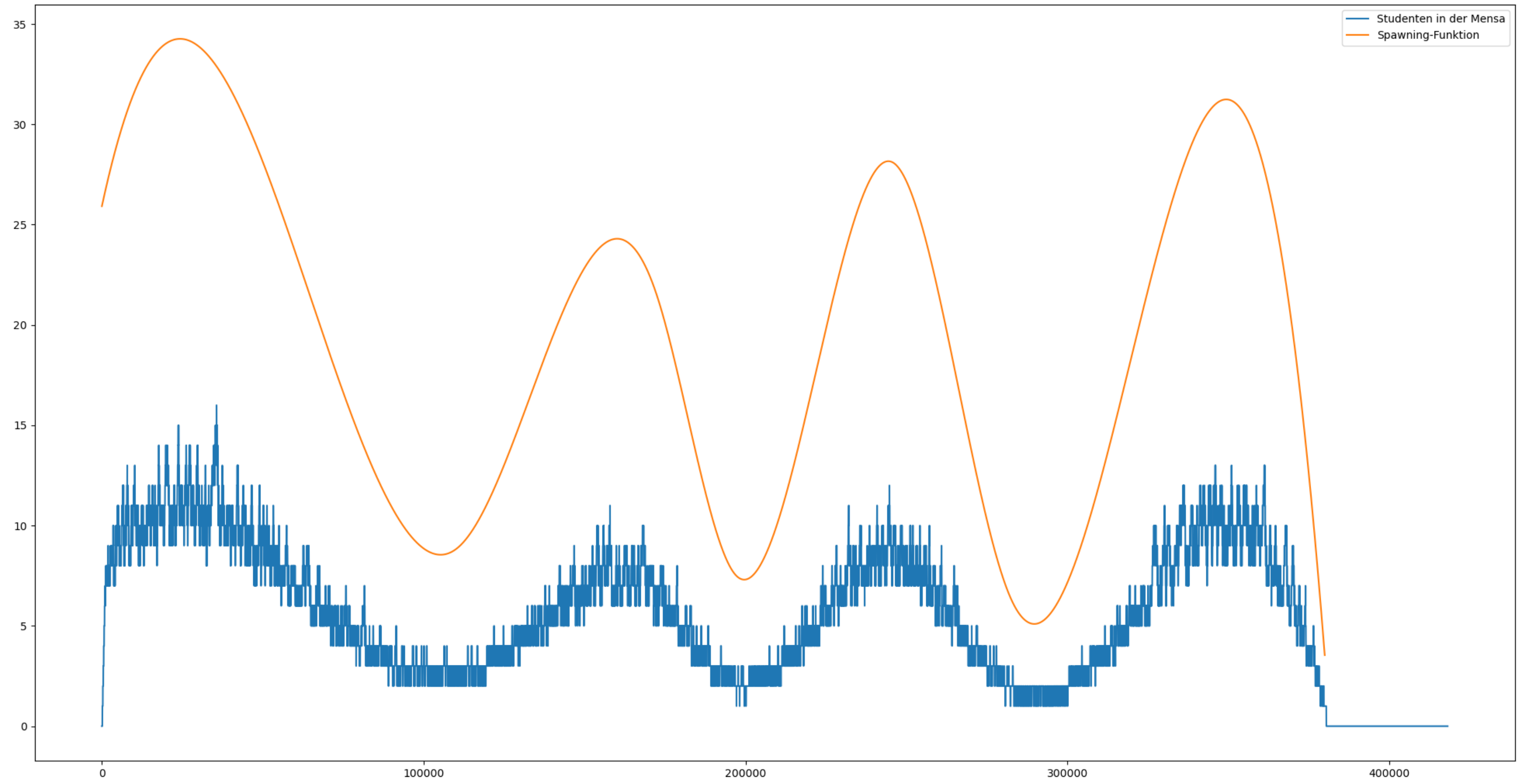
Wartezeitfunktion

- Minimiere $\mathsf{W}(v, \omega) = \frac{1}{\sqrt{N}} \cdot \sqrt{\sum_{i=1}^{N} (T_i(v, \omega))^2}$
- Maximale Studentengeschwindigkeit $v \in [8, 20]$, ungefähr 4 bis 11 $\frac{\text{km}}{\text{h}}$
- Zeit $t \in I = [0, 10\,200] \cap \mathbb{N}_0$
- Treppengewicht $\omega \in [0, 1]$

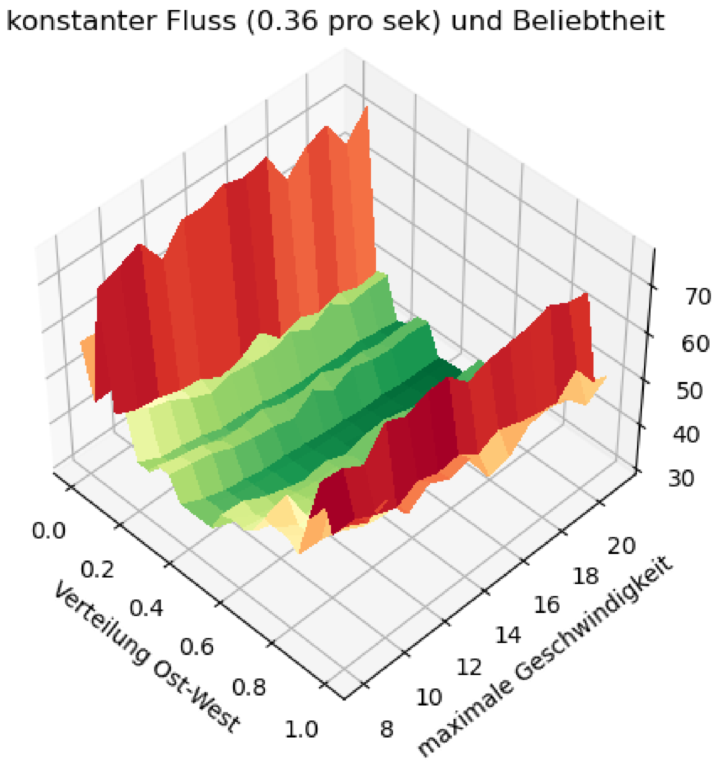
$$(v_*, \omega_*) = \arg \min_{v \in [8,20], \omega \in [0,1]} \mathsf{W}(v, \omega) \tag{2}$$

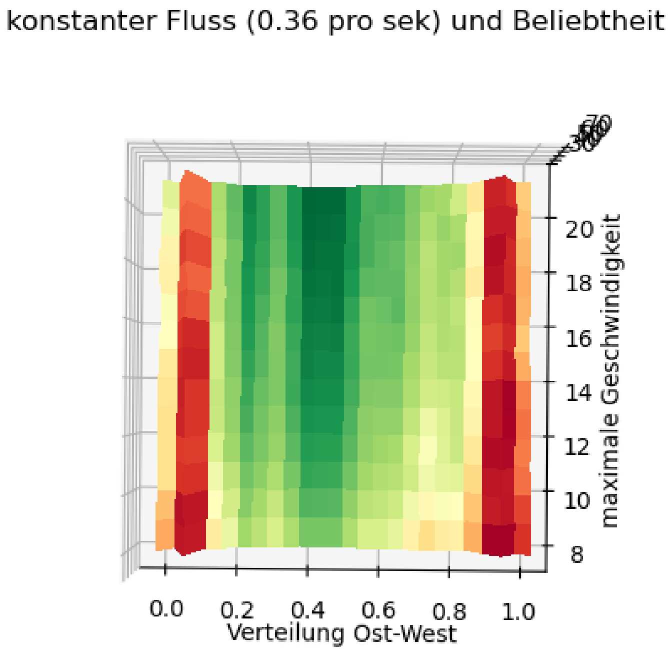
Simulation und reale Spawning-Funktion

Studenten in unserer Simulation (blau) im Vergleich zu Spawning-Funktion (orange)

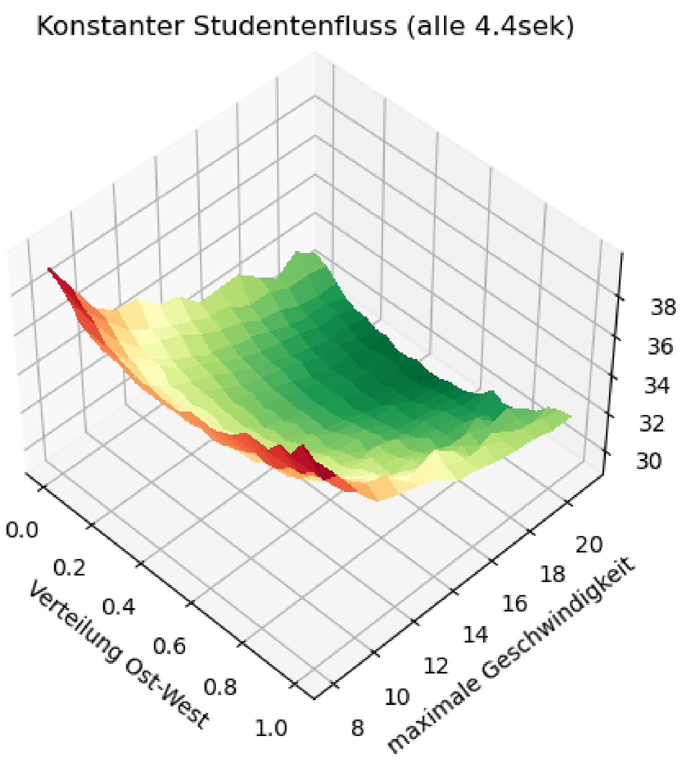


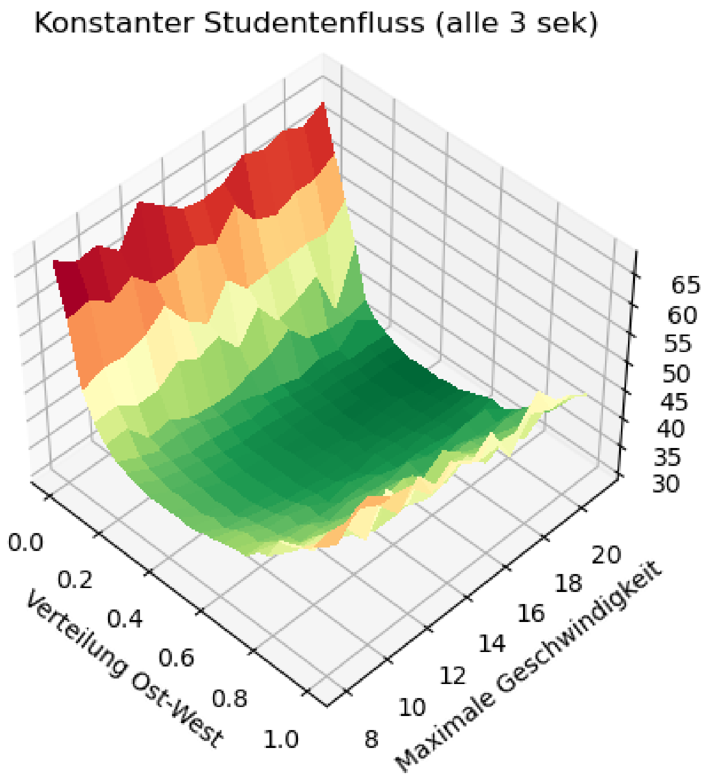
Gleich beliebte Gerichte und konstanter Studentenstrom



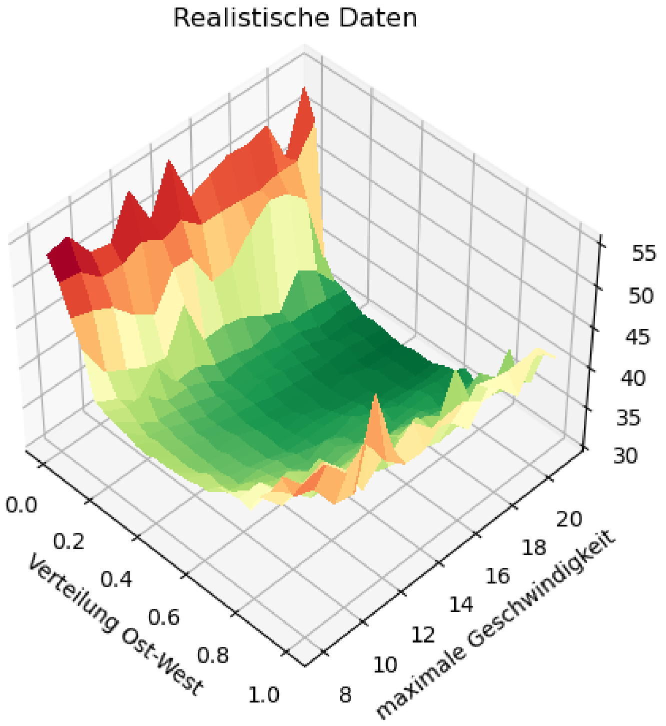


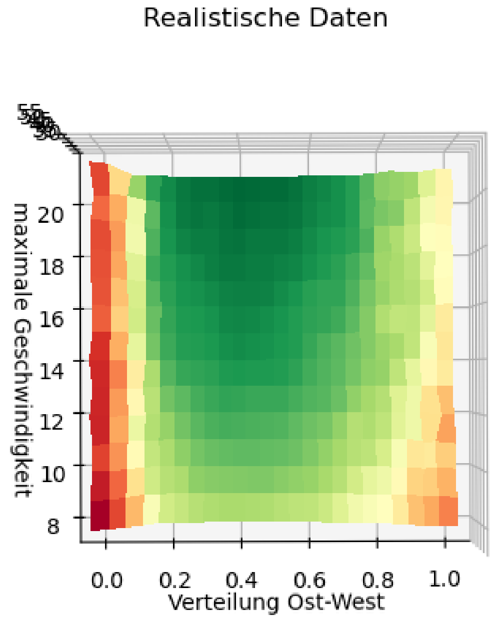
Realistische Beliebtheit der Gerichte, konstanter Studentenstrom





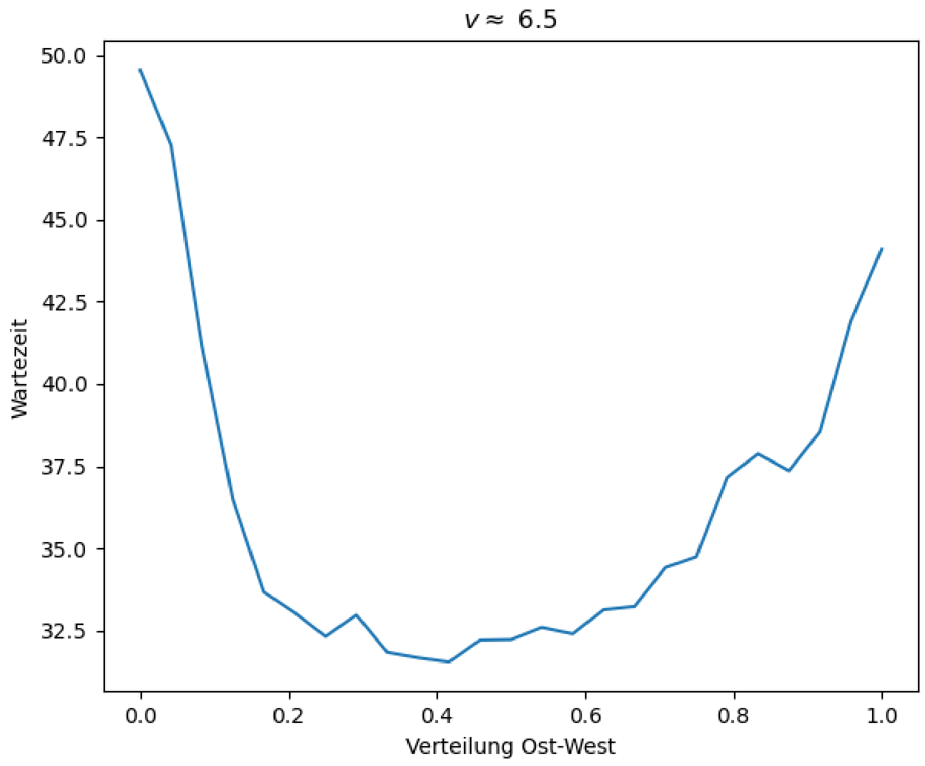
Daten aus der Zählung für Beliebtheit und Studentenstrom





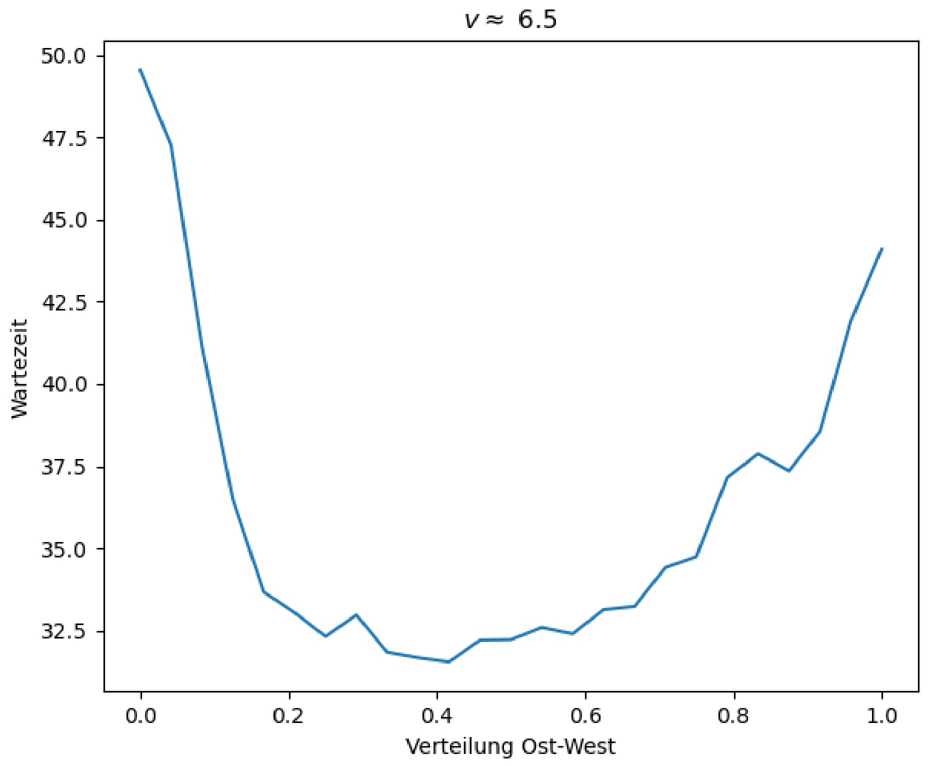
Weiteres

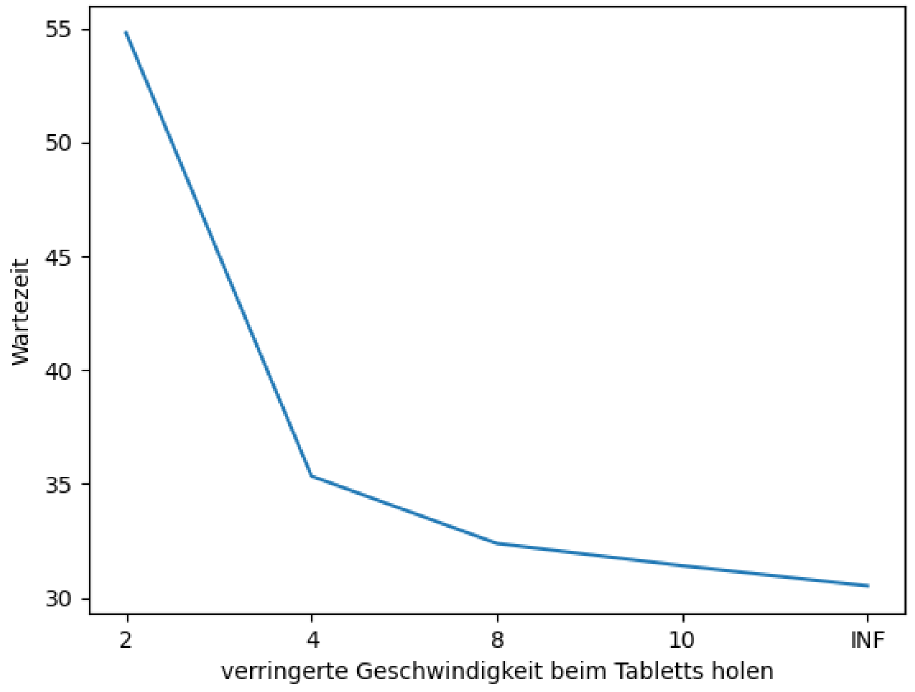
Ausschnitt aus vorherigem Plot (links) und Einfluss von langsamen Tablett/Besteck abholen (rechts)



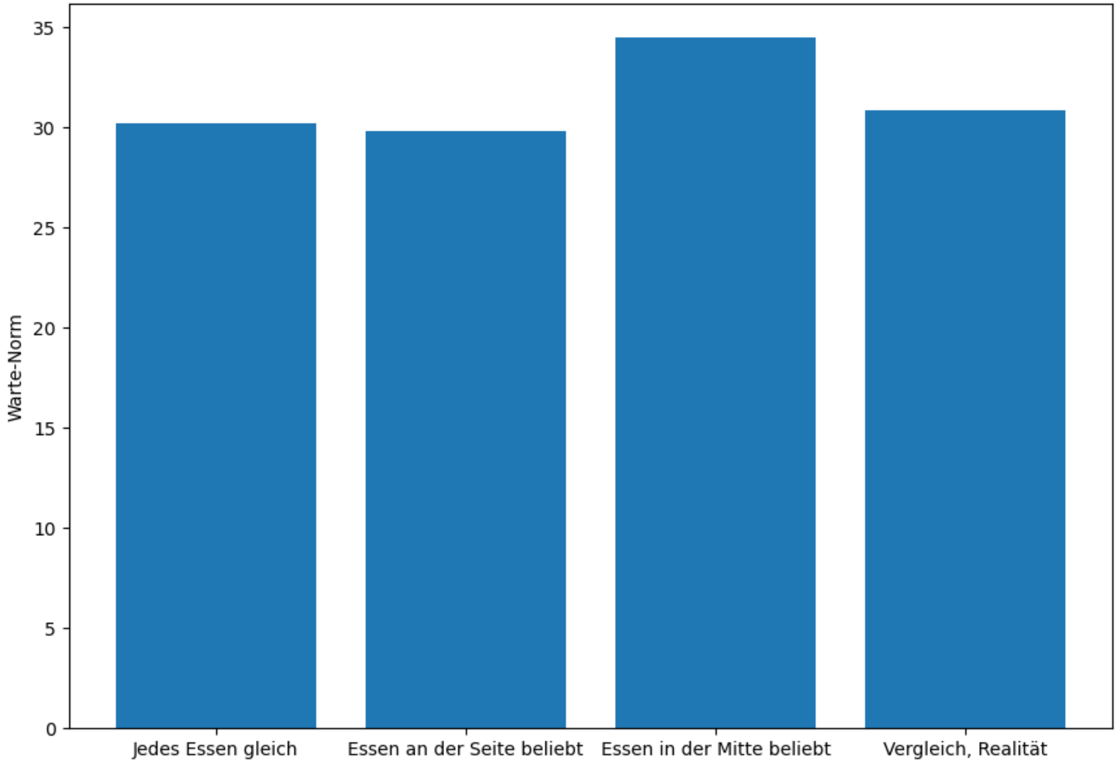
Weiteres

Ausschnitt aus vorherigem Plot (links) und Einfluss von langsamen Tablett/Besteck abholen (rechts)

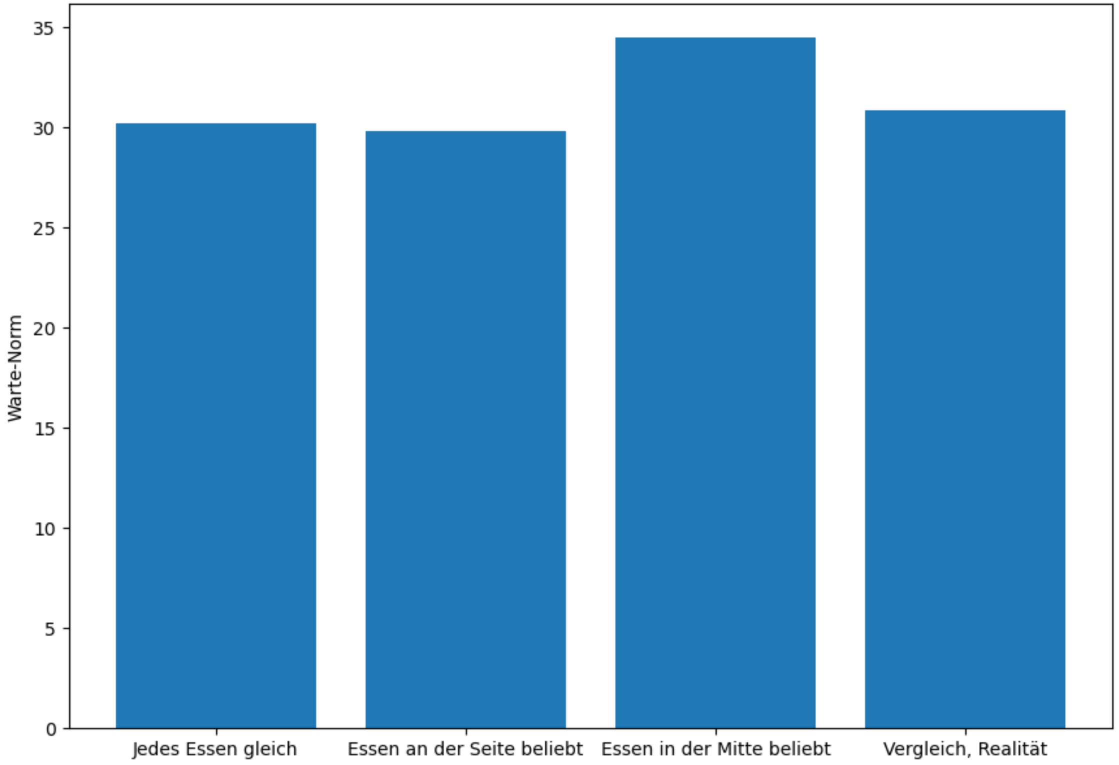




Andere Beliebtheit der Gerichte

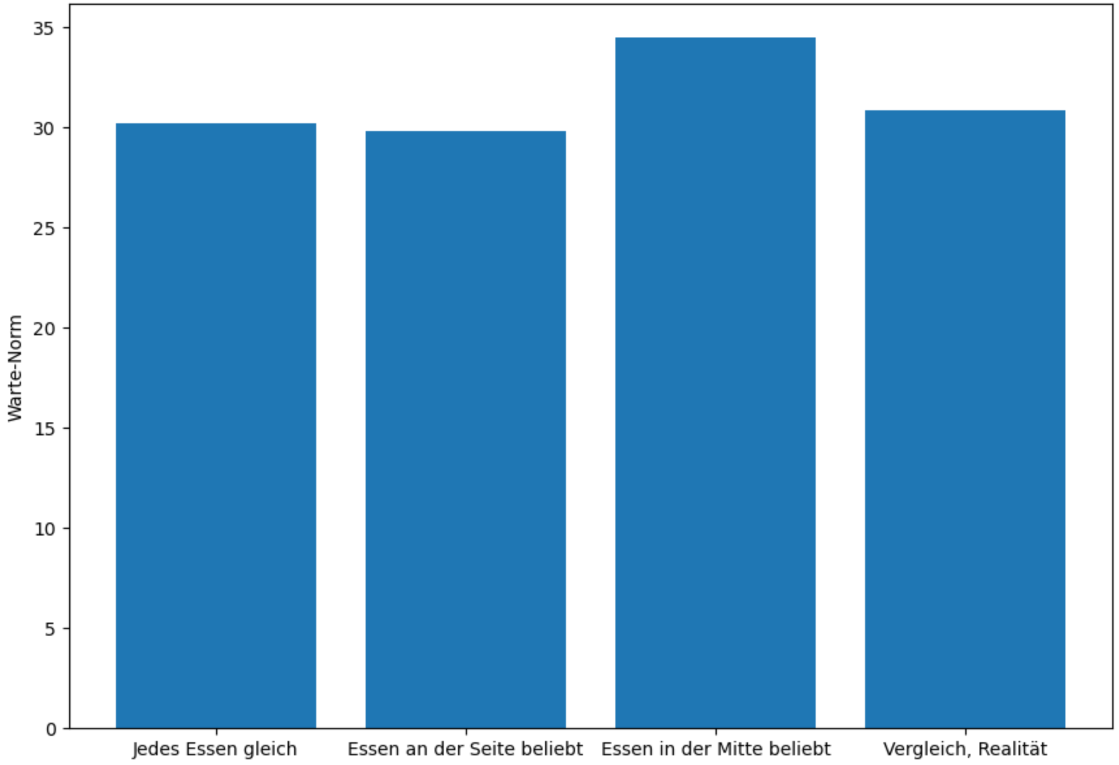


Andere Beliebtheit der Gerichte



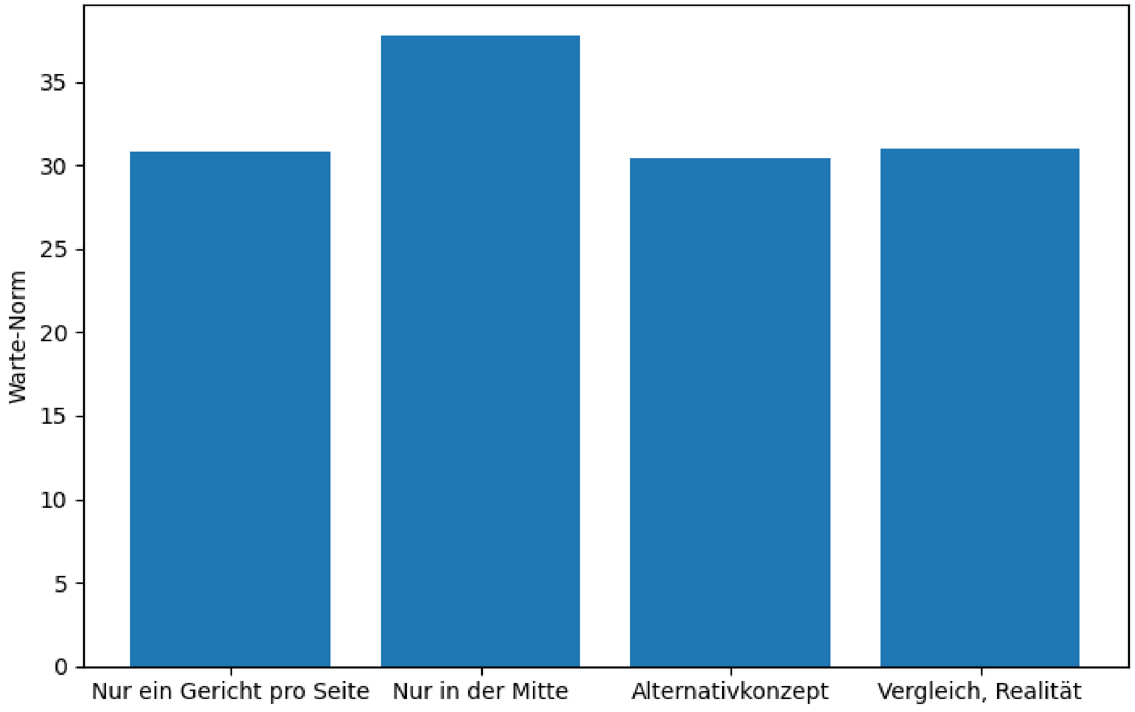
Leichte Verbesserung wenn Gerichte an den Seiten oder alle Gerichte gleich beliebt sind.

Andere Beliebtheit der Gerichte

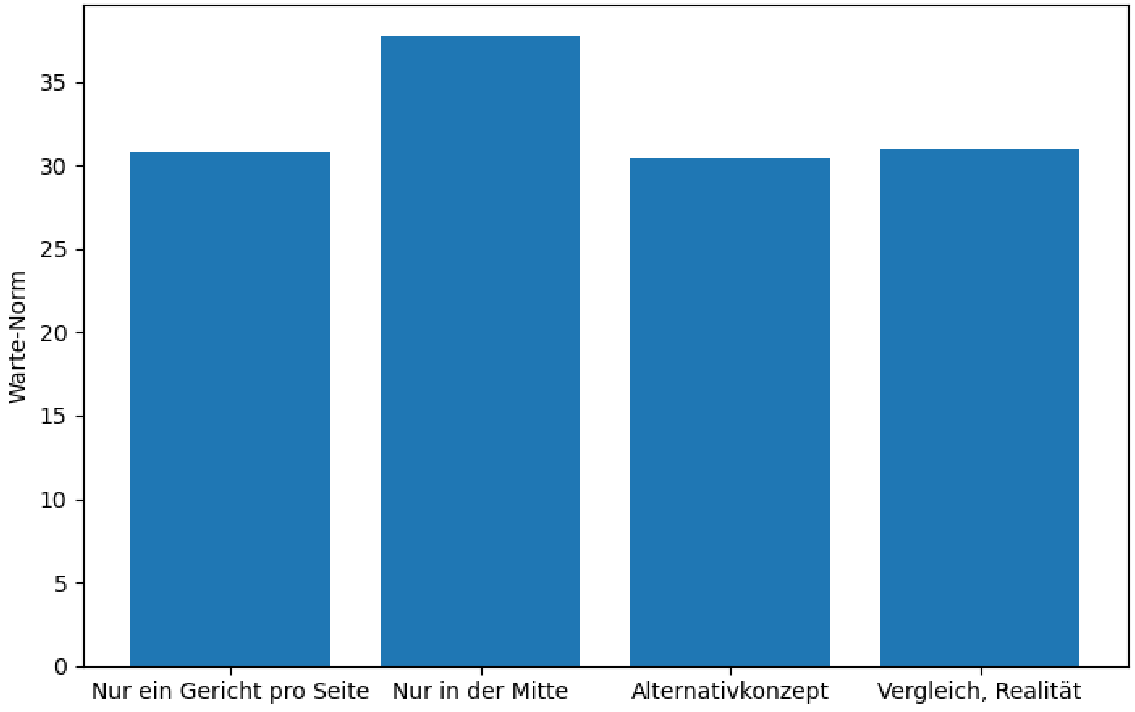


Leichte Verbesserung wenn Gerichte an den Seiten oder alle Gerichte gleich beliebt sind.
Kann vorausgesagt werden welche Gerichte beliebt sind?

Gerichte nur an ausgewählten Theken

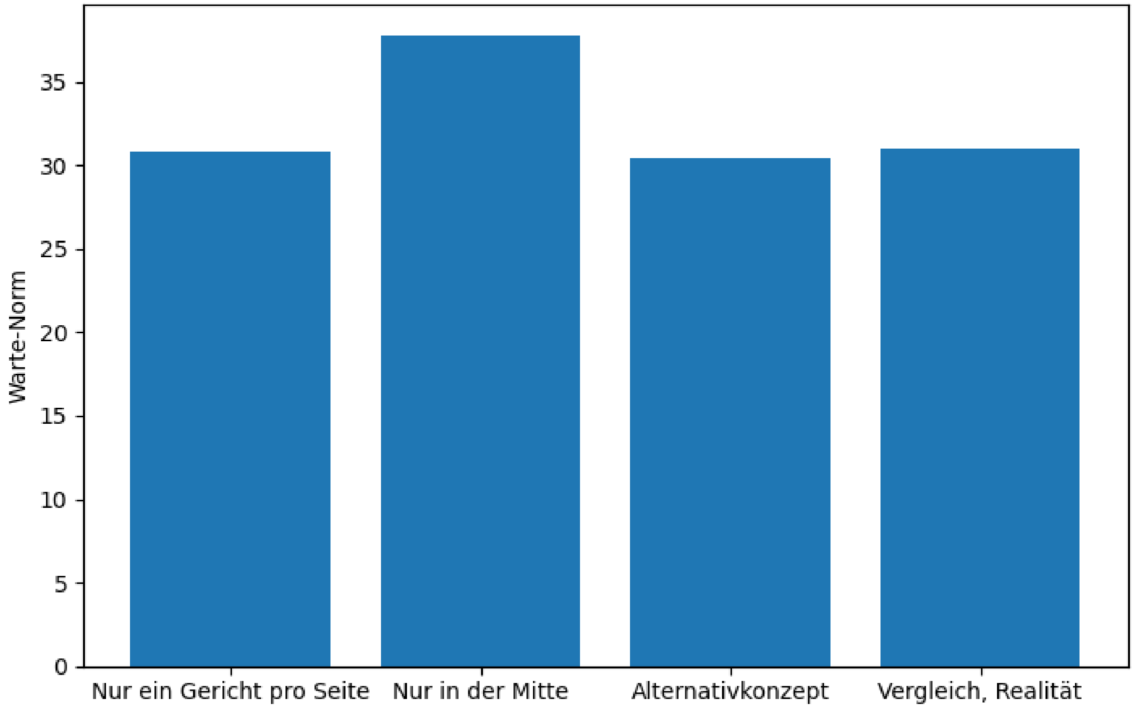


Gerichte nur an ausgewählten Theken



Angebot der Gerichte nur in der Mitte nicht sinnvoll

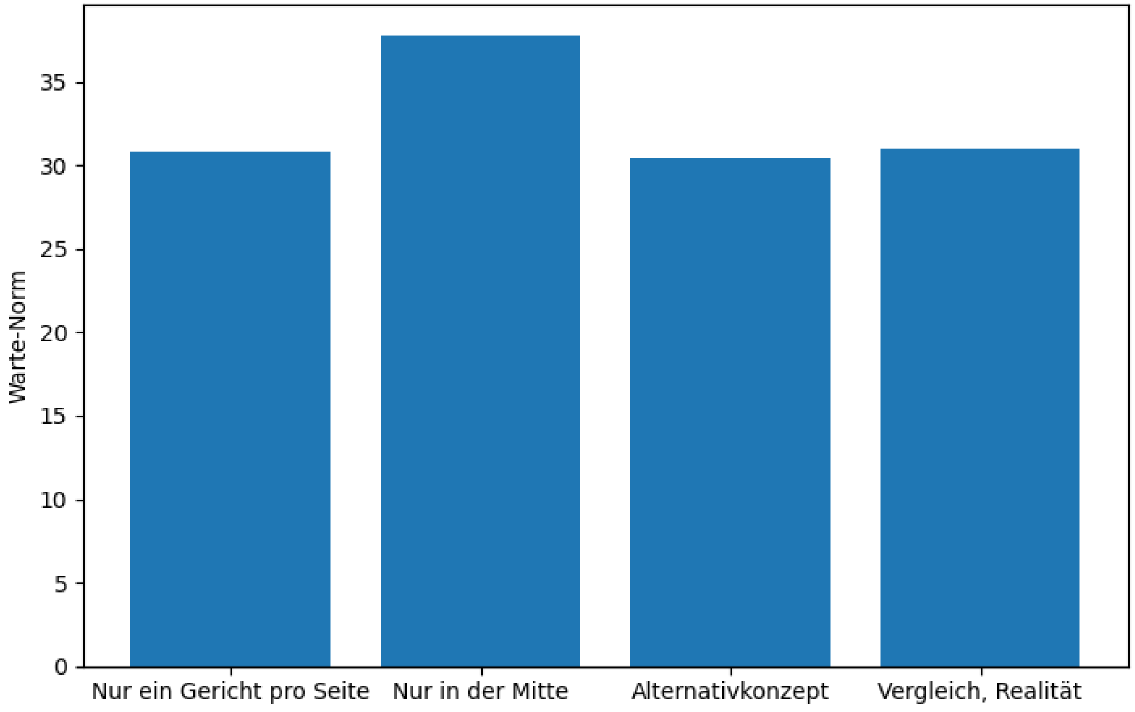
Gerichte nur an ausgewählten Theken



Angebot der Gerichte nur in der Mitte nicht sinnvoll

Minimal besser: Über Treppe West nur Essen 1 und Essen 3, über Treppe Ost nur Essen 5 und 6

Gerichte nur an ausgewählten Theken



Angebot der Gerichte nur in der Mitte nicht sinnvoll

Minimal besser: Über Treppe West nur Essen 1 und Essen 3, über Treppe Ost nur Essen 5 und 6

Das ist jedoch nur schwer umsetzbar und setzt Vorausschauende Treppenwahl voraus. Zudem tritt nur eine minimale Verbesserung auf.

Evtl aber andere Konzepte möglich, an die wir nicht gedacht haben!

Sensitivitätsanalyse

Sensitivitätsanalyse

Naiver Ansatz: Betrachte in den Plots große Abweichung der Warte-Norm zwischen Werten $\Leftrightarrow$ Ableitung des Modells in Richtung der Variable

Sensitivitätsanalyse

Naiver Ansatz: Betrachte in den Plots große Abweichung der Warte-Norm zwischen Werten $\Leftrightarrow$ Ableitung des Modells in Richtung der Variable
$\rightarrow$ Wir erinnern uns an den 3D-Plot, große Änderung wenn die Verteilung der Studenten auf die Treppen angepasst wird

Sensitivitätsanalyse

Naiver Ansatz: Betrachte in den Plots große Abweichung der Warte-Norm zwischen Werten $\Leftrightarrow$ Ableitung des Modells in Richtung der Variable
$\rightarrow$ Wir erinnern uns an den 3D-Plot, große Änderung wenn die Verteilung der Studenten auf die Treppen angepasst wird
$\Rightarrow$ Theoretisch einfache Lösung, gleichmäßige Aufteilung auf Treppen nötig um Wartezeit gering zu halten

Sensitivitätsanalyse

Naiver Ansatz: Betrachte in den Plots große Abweichung der Warte-Norm zwischen Werten $\Leftrightarrow$ Ableitung des Modells in Richtung der Variable
$\rightarrow$ Wir erinnern uns an den 3D-Plot, große Änderung wenn die Verteilung der Studenten auf die Treppen angepasst wird
$\Rightarrow$ Theoretisch einfache Lösung, gleichmäßige Aufteilung auf Treppen nötig um Wartezeit gering zu halten
Umsetzung evtl möglich indem pro Seite nur ein Gericht angeboten wird

Sensitivitätsanalyse

Naiver Ansatz: Betrachte in den Plots große Abweichung der Warte-Norm zwischen Werten $\Leftrightarrow$ Ableitung des Modells in Richtung der Variable
$\rightarrow$ Wir erinnern uns an den 3D-Plot, große Änderung wenn die Verteilung der Studenten auf die Treppen angepasst wird
$\Rightarrow$ Theoretisch einfache Lösung, gleichmäßige Aufteilung auf Treppen nötig um Wartezeit gering zu halten
Umsetzung evtl möglich indem pro Seite nur ein Gericht angeboten wird

Andere Möglichkeiten wie schnelleres Laufen oder erhöhte Ausgabegeschwindigkeit der Theken auch fördernd aber deutlich schwerer umsetzbar

Fazit

Fazit

- Optimales Treppengewicht: $\omega \approx 0,5$

Fazit

- Optimales Treppengewicht: $\omega \approx 0,5$
- Studentengeschwindigkeit: Auch bei höherer Geschwindigkeit bleibt die Ausgabegeschwindigkeit der limitierende Faktor

Fazit

- Optimales Treppengewicht: $\omega \approx 0,5$
- Studentengeschwindigkeit: Auch bei höherer Geschwindigkeit bleibt die Ausgabegeschwindigkeit der limitierende Faktor
- Verteilung der Gerichte: Optimal, falls alle Gerichte gleich beliebt sind

Fazit

- Optimales Treppengewicht: $\omega \approx 0,5$
- Studentengeschwindigkeit: Auch bei höherer Geschwindigkeit bleibt die Ausgabegeschwindigkeit der limitierende Faktor
- Verteilung der Gerichte: Optimal, falls alle Gerichte gleich beliebt sind
- Thekenwahl: spezielle, asymmetrische Thekenauswahl

Fazit

- Optimales Treppengewicht: $\omega \approx 0,5$
- Studentengeschwindigkeit: Auch bei höherer Geschwindigkeit bleibt die Ausgabegeschwindigkeit der limitierende Faktor
- Verteilung der Gerichte: Optimal, falls alle Gerichte gleich beliebt sind
- Thekenwahl: spezielle, asymmetrische Thekenauswahl

Durchschnittliche Wartezeit

Fazit

- Optimales Treppengewicht: $\omega \approx 0,5$
- Studentengeschwindigkeit: Auch bei höherer Geschwindigkeit bleibt die Ausgabegeschwindigkeit der limitierende Faktor
- Verteilung der Gerichte: Optimal, falls alle Gerichte gleich beliebt sind
- Thekenwahl: spezielle, asymmetrische Thekenauswahl

Durchschnittliche Wartezeit

- real: 33,6s

Fazit

- Optimales Treppengewicht: $\omega \approx 0,5$
- Studentengeschwindigkeit: Auch bei höherer Geschwindigkeit bleibt die Ausgabegeschwindigkeit der limitierende Faktor
- Verteilung der Gerichte: Optimal, falls alle Gerichte gleich beliebt sind
- Thekenwahl: spezielle, asymmetrische Thekenauswahl

Durchschnittliche Wartezeit

- real: 33,6s
- optimiert: 31,3s

Fazit

- Optimales Treppengewicht: $\omega \approx 0,5$
- Studentengeschwindigkeit: Auch bei höherer Geschwindigkeit bleibt die Ausgabegeschwindigkeit der limitierende Faktor
- Verteilung der Gerichte: Optimal, falls alle Gerichte gleich beliebt sind
- Thekenwahl: spezielle, asymmetrische Thekenauswahl

Durchschnittliche Wartezeit

- real: 33,6s
- optimiert: 31,3s

Wartezeit während Stoßzeiten

Fazit

- Optimales Treppengewicht: $\omega \approx 0,5$
- Studentengeschwindigkeit: Auch bei höherer Geschwindigkeit bleibt die Ausgabegeschwindigkeit der limitierende Faktor
- Verteilung der Gerichte: Optimal, falls alle Gerichte gleich beliebt sind
- Thekenwahl: spezielle, asymmetrische Thekenauswahl

Durchschnittliche Wartezeit

- real: 33,6s
- optimiert: 31,3s

Wartezeit während Stoßzeiten

- real: 39,4s

Fazit

- Optimales Treppengewicht: $\omega \approx 0,5$
- Studentengeschwindigkeit: Auch bei höherer Geschwindigkeit bleibt die Ausgabegeschwindigkeit der limitierende Faktor
- Verteilung der Gerichte: Optimal, falls alle Gerichte gleich beliebt sind
- Thekenwahl: spezielle, asymmetrische Thekenauswahl

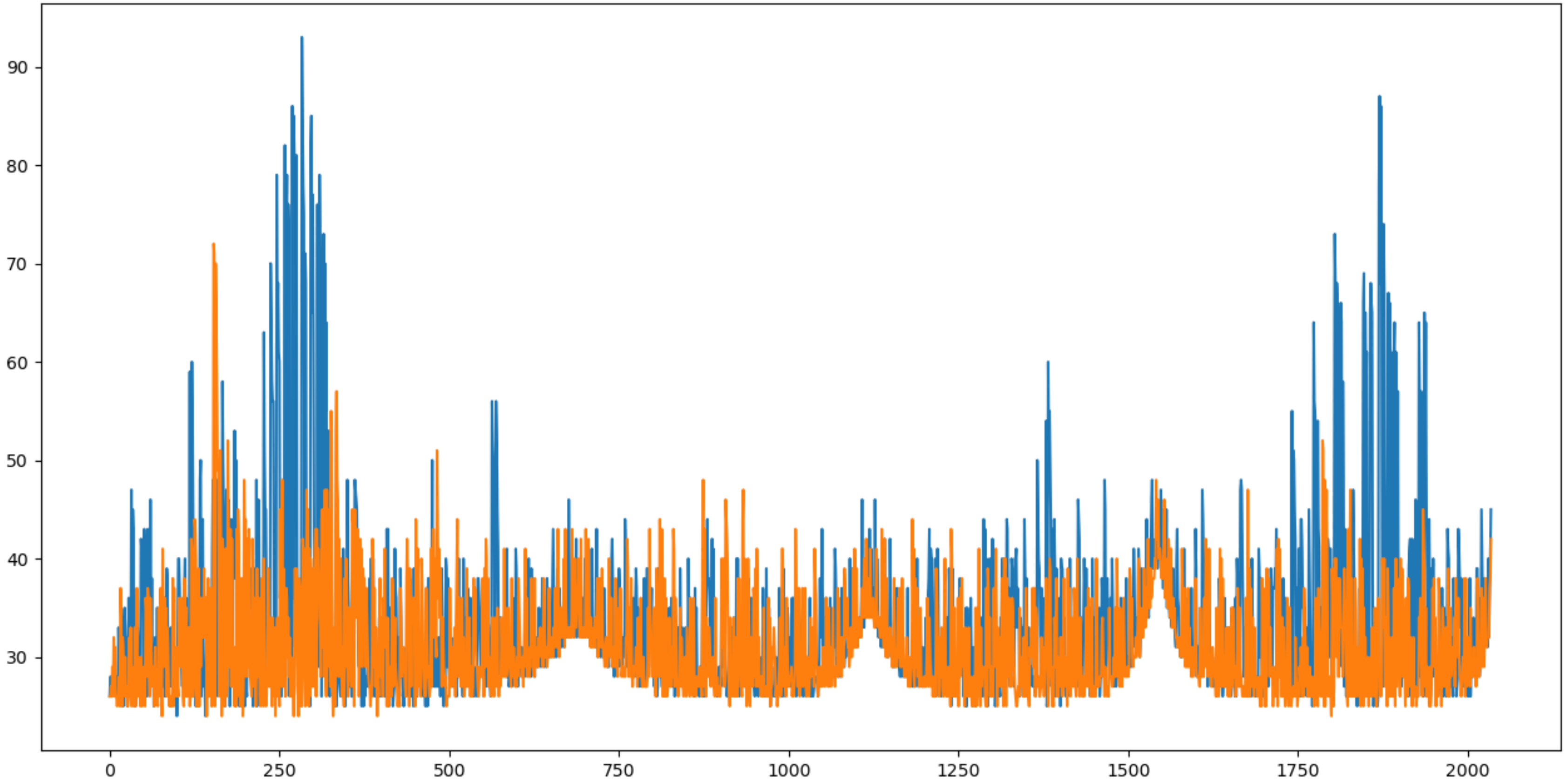
Durchschnittliche Wartezeit

- real: 33,6s
- optimiert: 31,3s

Wartezeit während Stoßzeiten

- real: 39,4s
- optimiert: 34s

Fazit



Fazit

"Wenn Ost- und Westtreppe gleichermaßen genutzt werden, kann die Wartezeit um bis zu 17% verringert werden!"

References

References

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

GitHub

https://github.com/9VZZH7/Mensa-Optimierung

Vielen Dank für eure Aufmerksamkeit!