Please update a copy of
scan-of-declaration-of-authorship.pdf

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

EFMD EQUIS ACCREDITED

Bachelor Thesis

Ein systematischer Vergleich von Ansätzen zum "Student Course Timetable Problem" am Beispiel der Semesterplanung Studierender der Wirtschaftsuniversität Wien im Bachelorstudium

Philipp Scheer

Date of Birth: 28.01.2003
Student ID: 12242922

Subject Area: Information Business

Studienkennzahl: J123456789

Supervisor: Assoz.Prof PD Dr. Stefan Sobernig

Date of Submission: XX. XXX 2025

Department of Information Systems & Operations Management, Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

DEPARTMENT FÜR WIRTSCHAFTSINFORMATIK UND OPERATIONS MANAGEMENT DEPARTMENT OF INFORMATION SYSTEMS AND OPERATIONS MANAGEMENT

Contents

List of Figures

List of Tables

Abstract

Todo

1 Einleitung

Kaum eine Entscheidung im Alltag von Studierenden erscheint so simpel und erweist sich gleichzeitig so komplex wie die Erstellung des eigenen Stundenplans. Was auf den ersten Blick nach einer organisatorischen Routineaufgabe aussieht, entpuppt sich bei näherer Betrachtung als vielschichtiges Optimierungsproblem: Überschneidungen zwischen Lehrveranstaltungen, begrenzte Kursplätze und individuelle Präferenzen wie Arbeitszeiten und Lerngewohnheiten machen die Planung zu einer Herausforderung

In einer Zeit, in der viele Entscheidungsprozesse bereits durch Algorithmen unterstützt werden, stellt sich die Frage, ob auch die Stundenplanerstellung automatisiert und optimiert werden kann. Hier setzt die folgende Arbeit an: Sie untersucht, wie sich das *Student Schedulung Problem (SSP)* mithilfe von verschiedenen Optimierungsverfahren modellieren und lösen lässt.

2 Hintergrund

2.1 Problemstellung

Die Erstellung eines optimalen Semesterplans ist eine zentrale Herausforderung für Studierende. Sie sehen sich im Hauptstudium mit einem umfangreichen Kursangebot konfrontiert, dessen Veranstaltungen sich in zahlreichen Terminkonstellationen überlappen können. Die Herausforderung besteht darin, aus diesem Pool an Lehrveranstaltungen eine zulässige und optimale Kombination zu wählen. Solche Zeittafelprobleme werden Student Scheduling Problems (SSP) genannt.

Die Problemstellung ist komplex: Einerseits müssen Hard Constraints wie Überlappungsfreiheit und Erfüllung einer Mindestanzahl von ECTS pro Semester erfüllt sein. Andererseits gilt es, Präferenzen, die als Soft Constraints ausgedrückt werden, zu maximieren, die die Lebensrealität der Studierenden widerspiegeln. Dazu zählen etwa die Berücksichtigung von Arbeitstätigkeiten, die durch Blockieren oder Priorisieren von gewissen Stunden in der Woche ausgedrückt werden, sowie die Favorisierung von Kursen, die durch eine Priorität ausgedrückt wird.

In dieser Arbeit wird das SSP für einen einzelnen Studierenden des Hauptstudiums Wirtschaftsinformatik (inklusive CBK) betrachtet und als ein Constaint-Satisfaction Problem (CSP) formuliert. Ziel ist eine möglichst akkurate Implementierung der Algorithmen, die Feldman und Golumbic im Paper "*Optimization Algorithms for Student Scheduling via Constraint Satisfiability*" [4] beschreiben, um diese anschließend miteinander zu vergleichen.

2.2 Motivation

Studierende berichten häufig von Schwierigkeiten bei der Erstellung ihrer Semesterpläne [2, 6, 7]. Zu den zentralen Ursachen zählt insbesondere die Vereinbarkeit mit einem Nebenjob; so bevorzugen 81% der Befragten mindestens einen vollständig freien Wochentag [7]. Zudem geben 74% der Studierenden an, dass eine gleichmäßige Verteilung von Lehrveranstaltungen und Prüfungen für sie das wichtigste Kriterium bei der Stundenplanerstellung darstellt [3]. Weiterhin möchten 82% vermeiden, mehr als eine Prüfung pro Tag zu absolvieren [3]. Bezüglich der zeitlichen Präferenzen werden Mittagsstunden am häufigsten gewählt, gefolgt von frühen und späten Zeitslots. Rund 31% der Studierenden bevorzugen es, an bestimmten Tagen keine Prüfungen zu haben; besonders unpopulär sind Samstag, Sonntag, Freitag und Montag [3].

Obwohl sich diese Präferenzen grundsätzlich leicht modellieren lassen,

stehen derzeit keine Programme zur Verfügung, die eine automatisierte Erstellung entsprechender Stundenpläne an der WU ermöglichen. Lediglich der Studienplaner der Österreichischen Hochschüler*innenschaft der WU [10] erlaubt die manuelle Kombination von Lehrveranstaltungen und Prüfungen im Hinblick auf Überschneidungsfreiheit, bietet jedoch keine Möglichkeit zur Optimierung nach individuellen Präferenzen.

2.3 Zielsetzung

Die vorliegende Bachelorarbeit hat zum Ziel, die beschriebene Problemstellung unter den aktuellen technischen Rahmenbedingungen mithilfe verschiedener Lösungsansätze zu untersuchen, diese anhand eines aktuellen Datensatzes zu validieren und anschließend einem Leistungsvergleich zu unterziehen.

1. **Validierung:** Im Mittelpunkt steht die exakte Nachprogrammierung der Algorithmen von Feldman und Golumbic [4]. Es soll überprüft werden, ob diese Verfahren unter modernen Rechenbedingungen und anhand realer Daten aus dem Vorlesungsverzeichnis der Wirtschaftsuniversität Wien in der Lage sind, einen optimalen Semesterplan zu erstellen.

2. **Vergleichende Leistungsanalyse:** Die implementierten Algorithmen werden hinsichtlich der Qualität der erreichten Lösungen (*Score*), ihrer Geschwindigkeit sowie ihres Speicherbedarfs systematisch miteinander verglichen.

2.4 Zeittafelprobleme

Für die Beschriebenen Zeittafelprobleme gibt es verschiedene Algorithmen/Lösungsansätze:

1. **Hill-Climbing Algorithmen:** Hill-Climbing Algorithmen gehören zur Klasse der lokalen Such- und Optimierungsverfahren. Sie finden gute oder nahezu optimale Lösungen für schwierige (*NP-hard*) Optimierungsprobleme, ohne den gesamten Lösungsraum zu durchsuchen. Der Algorithmus versucht von einem Zustand in einen benachbarten Zustand mit besserem Zielfunktionswert überzugehen — ähnlich wie das Erklimmen eines Hügels, auf einer Landschaft, wobei die "Höhe" des Hügels die Qualität der Lösung darstellt.

   Diese Vorgehensweise macht sie besonder effizient für große, aber gut strukturierte Optimierungsprobleme, da in kurzer Zeit eine gute Lösung

gefunden werden kann. Die Implementierung ist unkompliziert, was Hill-Climbing Algorithmen in vielen Anwendungsbereichen zu einem beliebten heuristischen Ansatz macht. Ein wesentlicher Nachteil liegt in der Tendenz, in lokalen Optima zu verharren. Da der Algorithmus in seiner Grundform nur Verbesserungen akzeptiert, werden lokale Minima nicht durchschritten, selbst wenn sich dahinter ein höheres Maxima befindet [5].

2. **Offering-Order Algorithmus:** Der Offering-Order Algorithmus, welcher von Feldman und Golumbic [4] beschrieben wird, kombiniert ein heuristisches Ordnungsverfahren mit einer *Tree Search*. Der Kern des Ansatzes besteht darin, eine heuristisch bestimmte Reihenfolge für die Instanziierung der Variablen (Kurse) festzulegen — die sogenannte "*offering order*". Innerhalb ihrer Planpunkte werden somit alle Kurse vorsortiert. Mittels einer Vorwärts-Suche wird der vorsortierte Suchbaum durchsucht, um dem aktuellen Zustand einen weiteren Kurs hinzuzufügen. Führt der neue Zustand zu einem ungültigen Ergebnis, wird der letzte gültige Zustand wiederhergestellt ("*back-tracking*").

   Durch die Wahl einer sinnvollen Reihenfolge lassen sich ineffiziente Suchpfade vermeiden. Auf diese Weise wird der Suchraum stark reduziert und es kann eine Lösung gefunden werden, ohne den gesamten Lösungsraum zu besuchen.

3. **Integer Linear Programming (ILP):**

2.5 Replikationsarbeit

2.6 Abgrenzung

Folgende Aspekte sind explizit vom Forschungsumfang ausgeschlossen:

1. Das Problem wird ausschließlich als SSP für einen einzelnen Studierenden gelöst. Es wird keine Rücksicht auf andere Studierende, die maximale Kapazität von Räumen, die Verfügbarkeit von Professoren oder die allgemeine universitäre Ressourcenplanung genommen.

2. Die Planung beschränkt sich auf ein einzelnes Semester des Hauptstudium Wirtschaftsinformatik.

3. Die Entwicklung einer voll funktionsfähigen Applikation zur Dateneingabe, Speicherung von Planungen, oder komplexer Visualisierung wird ausgeschlossen. Das Ergebnis des Programms wird durch ein computerlesbares Format (JSON) ausgegeben.

3 Datenbereitstellung

Um einen möglichst aktuellen Datensatz zu erzeugen, wurden die Vorlesungsdaten aus dem Vorlesungsverzeichnis der Wirtschaftsuniversität Wien [1] automatisiert ausgelesen. Jedem Kurs (*Offering*) ist eine Lehrveranstaltungs-ID zugewiesen, welche durch eine Zahl von 1 bis 9999 dargestellt wird. Weiters gehört jeder Kurs einem Planpunkt (*Course*) an.

3.1 Struktur des Datensatzes

Der Datensatz des Vorlesungsverzeichnisses der Wirtschaftsuniversität Wien für das Sommersemester 2025 bildet die Grundlage für die vorliegende Arbeit. Die Daten wurden automatisiert unter Verwendung des beschriebenen Prozesses 1 extrahiert.

Die Modellierung des Studiums an der WU basiert auf zwei zentralen Entitäten, die für das Verständnis der Optimierungsalgorithmen wichtig sind:

- **Course (Planpunkt):** Ein *Course* repräsentiert ein zu absolvierendes Modul, das mit einer festen ECTS-Anzahl assoziiert ist. Die Nebenbedingung 4 sorgt dafür, dass aus den verfügbaren Lehrveranstaltungen eines Planpunkts genau eines ausgeähnlt wird, um das Modul abzuschließen.
- **Offering (Lehrveranstaltung):** Ein *Offering* ist eine Lehrveranstaltung, die einem Planpunkt angehört. Jedes Offering besitzt eine eindeutige ID (von $1 - 9999$) und ist mit konkreten Termin- und Zeitangaben hinterlegt, die für die Prüfung von zeitlichen Konflikten verwendet wird.

Der extrahierte Datensaz umfasst für jedes *Offering* die folgenden relevanten Attribute:

3.2 Extraktionsprozess

Der Prozess der automatisierten Extraktion wurde folgendermaßen realisiert:

Algorithm 1 Datenextraktion aus Vorlesungsverzeichnis

1: $offerings \leftarrow \emptyset$
2: $modules \leftarrow \emptyset$
3: **for** $id \leftarrow 1, 9999$ **do**
4: $\quad url \leftarrow \textsc{CraftCourseURL}(id)$
5: $\quad page \leftarrow \textsc{HttpGet}(url)$
6: $\quad$ **if** $\textsc{Contains}(page"Keine\ Lehrveranstaltung\ gefunden")$ **then**
7: $\quad\quad$ **continue**
8: $\quad$ **end if**
9: $\quad (module, dates) \leftarrow \textsc{ExtractDates}(page)$
10: $\quad$ **if** $module \notin \mathrm{dom}(modules)$ **then**
11: $\quad\quad ects \leftarrow \textsc{ExtractECTSFromModule}(module)$
12: $\quad\quad modules \leftarrow modules \cup \{(module, ects)\}$
13: $\quad$ **end if**
14: $\quad offerings \leftarrow offerings \cup \{(id, dates, modules_{module})\}$
15: **end for**

Das Ergebnis sieht wie folgt aus:

offeringId	dates	lvLeiter	module_ids	ects
5877	[{"start": "2025-03-11 09:00", "end": "2025-03-11 11:00"}, ...]	Assoz.Prof PD Dr. Sabrina Kirrane	["6590", "9485"]	5
6427	[{"start": "2025-03-13 16:30", "end": "2025-03-13 20:30"}, ...]	Dr. Robert Kosik	['5160']	6
4676	[{"start": "2025-05-06 14:30", "end": "2025-05-06 17:00"}, ...]	Dr. Stefan Treitl	['6012']	4
6295	[{"start": "2025-03-06 08:00", "end": "2025-03-06 12:00"}, ...]	Anna-Lena Klug, MSc. MSc.	['5155']	8
5745	[{"start": "2025-03-10 18:00", "end": "2025-03-10 20:00"}, ...]	Dr. Reinhard Zuba	['5108']	4
6258	[{"start": "2025-04-01 14:00", "end": "2025-04-01 19:00"}, ...]	Dr. Nikolaus Obwegeser	['5162']	3
5944	[{"start": "2025-05-08 09:30", "end": "2025-05-08 12:00"}, ...]	Univ.Prof. Dr. Jonas Puck	['5107']	4
5724	[{"start": "2025-03-10 16:00", "end": "2025-03-10 20:00"}, ...]	Alexander Oberreiter, M.Sc.	['5105']	8
5752	[{"start": "2025-05-06 12:00", "end": "2025-05-06 14:30"}, ...]	Univ.Prof. Dipl.Wirtsch.-Math.Dr. Birgit Rudloff	['6023']	4
6354	[{"start": "2025-05-06 15:30", "end": "2025-05-06 18:00"}, ...]	Zhenyi Wang, M.Sc.	['5059', '6059']	4
6447	[{"start": "2025-03-13 12:30", "end": "2025-03-13 16:30"}, ...]	Anita Neumannova, PhD, MSc (WU), MIM (CEMS)	['5161']	4
5329	[{"start": "2025-09-01 14:00", "end": "2025-09-01 17:00"}, ...]	Dr. Tingmingke Lu	['5056', '6056']	4
6317	[{"start": "2025-03-05 14:00", "end": "2025-03-05 18:00"}, ...]	Dipl.-Wirt.-Ing.(FH) Mark Nicolas Grünsteidl	['5158']	8
6130	[{"start": "2025-05-06 09:30", "end": "2025-05-06 12:00"}, ...]	Jana Hlavinova, Ph.D.	['6024']	4
6182	[{"start": "2025-03-11 10:00", "end": "2025-03-11 12:00"}, ...]	Dr. Sonja Sperber	['5136', '6911']	3
6089	[{"start": "2025-09-10 09:00", "end": "2025-09-10 13:00"}, ...]	Dr. Nikolai Neumayer	['5106']	4
4086	[{"start": "2025-05-06 14:30", "end": "2025-05-06 17:00"}, ...]	Vanessa Kofler, LL.M. (WU)	['5109', '6021']	4
5762	[{"start": "2025-03-10 16:30", "end": "2025-03-10 18:00"}, ...]	Univ.Prof. Jonas Bunte, Ph.D.	['5117']	4

Table 1: Auszug des Ergebnisses der Datenbereitstellung

3.3 Deskriptive Analyse

Um die Komplexität des Datensatzes und des Optimierungsproblems zu quantifizieren, wurde eine deskriptive Analyse durchgeführt.

Der vollständige Datensatz besteht aus insgesamt 381 *Offerings* ($|\mathcal{O}|$) und 28 Planpunkten, woraus sich eine durchschnittliche Anzahl von 13.6 *Offerings* pro Planpunkt ergibt.

Ein wesentlicher Faktor für die Schwierigkeit der Stundenplanerstellung ist die zeitliche Dichte der angebotenen Kurse. Tabelle 2 zeigt die Verteilung der Lehrveranstaltungen über die Wochentage.

Wochentag	**Anteil (%)**
Montag	21.1%
Dienstag	25.4%
Mittwoche	21.8%
Donnerstag	19.7%
Freitag	11.3%
Samstag	0.7%
Sonntag	0%
Total	**100.0%**

Table 2: Prozentuale Verteilung der Kurse auf Wochentage

Wie aus den Daten ersichtlich wird, konzentriert sich der Großteil des Lehrangebots (ca. 88 %) auf die Kernzeit von Montag bis Donnerstag, während der Freitag mit 11.3 % bereits deutlich seltener belegt ist. Samstage spielen im regulären Lehrbetrieb nahezu keine Rolle. Diese ungleiche Verteilung verschärft die Problematik zeitlicher Überschneidungen ($\mathcal{F}_{Zeit}$) an Tagen mit besonders vielen Kursen.

Zusätzlich zur Verteilung auf die Tage ist die zeitliche Platzierung innerhalb eines Tages entscheidend für die Modellierung von Präferenzen. Tabelle 3 schlüsselt die Startzeiten der einzelnen Kurseinheiten auf.

Stunde	**Anteil (%)**
7:00	0.6%
8:00	16.9%
9:00	8.3%
10:00	10.1%
11:00	6.1%
12:00	8.0%
13:00	11.4%
14:00	8.1%
15:00	5.9%
16:00	10.3%
17:00	5.6%
18:00	8.8%
19:00	0.0%

Table 3: Prozentuale Verteilung der Start-Uhrzeit der Kurse

Die Analyse der Startzeiten verdeutlicht, dass die WU Wien ein sehr breites Zeitfenster nutzt. Ein signifikanter Anteil der Kurse beginnt bereits um 08:00 Uhr (16,9 %), was für Studierende mit Präferenzen für einen späteren Vorlesungsbeginn (vgl. Szenario 4.4.1) eine hohe Anzahl an potenziell zu vermeidenden Einheiten bedeutet.

Neben dem Beginn der Einheiten variiert auch die Dauer der einzelnen Termine, was die Berechnung von Überschneidungsfreiheiten komplexer gestaltet als bei starren Zeitslots. Tabelle 4 stellt die Dauer der Einheiten dar.

Stunden	**Anteil (%)**
1	0.7%
2	27.1%
3	41.9%
4	27.2%
5	1.8%
6	0.8%
7	0.1%
8	0.3%

Table 4: Prozentuale Verteilung der Dauer der Kurse

Die überwiegende Mehrheit der Veranstaltungen (über 96 %) dauert zwischen zwei und vier Stunden. Da die Termine jedoch individuell als Zeitstempel (Start/Ende) vorliegen und nicht in ein fixes Raster gepresst sind, muss der Algorithmus präzise Zeitintervall-Prüfungen durchführen, anstatt lediglich Slots zu vergleichen.

4 Analyse

4.1 Implementierung des ILP Baseline Ansatzes

Feldman und Golumbic vergleichen in Ihrem Paper [4] die Outputs der Algorithmen "Hill Climbing" und "Offering Order" mit dem optimalen Stundenplan, der mittels Brute-Force ermittelt wurde. Da ein Brute-Force Algorithmus bei der großen Anzahl der Vorlesungen im VVZ der WU eine zu lange Laufzeit hätte, wurde eine Integer Linear Programming (ILP) Lösung als Benchmark gewählt, die ebenfalls unter Berücksichtigung aller Nebenbedingungen den optimalen Stundenplan erzeugt.

Die Implementierung des ILP-Modells basiert auf der `PuLP`-Bibliothek in Python und dient der Maximierung des Gesamtnutzens (*Mark*) eines Stundenplans. Das mathematische ILP-Modell sieht wie folgt aus:

Mathematische Formulierung des ILP-Modells

$$\text{maximiere:} \quad \sum_{i \in \mathcal{O}} \text{Mark}(i) \cdot y_i \tag{1}$$

unter den Nebenbedingungen:

$$\sum_{i \in \mathcal{O}} y_i \geq C_{\min} \tag{2}$$

$$\sum_{i \in \mathcal{O}} y_i \leq C_{\max} \tag{3}$$

$$y_i + y_j \leq 1 \quad \forall (i,j) \in \mathcal{F}_{\text{Planpunkt}} \tag{4}$$

$$y_i + y_j \leq 1 \quad \forall (i,j) \in \mathcal{F}_{\text{Zeit}} \tag{5}$$

$$y_i = 1 \quad \forall i \in \mathcal{M}_{\text{Muss}} \tag{6}$$

$$y_i = 0 \quad \forall i \in \mathcal{M}_{\text{Blockiert}} \tag{7}$$

$$y_i \in \{0,1\} \quad \forall i \in \mathcal{O} \tag{8}$$

Figure 1: Mathematische Formulierung des ILP-Modells zur Stundenplanoptimierung.

4.1.1 Variablen, Parameter und Zielfunktion

Die Kernkomponente des Modells sind die binären Entscheidungsvariablen y_i, definiert für jede mögliche Lehrveranstaltung i. Eine Lehrveranstaltung ist ausgewählt, wenn $y_i = 1$, während $y_i = 0$ bedeutet, dass die Lehrveranstal-

tung nicht gewählt wird. Die Menge aller zur Auswahl stehenden Lehrveranstaltungen wird durch $\mathcal{O}$ repräsentiert (vgl. 8).

Die Zielfunktion (1) maximiert den Gesamtnutzen (engl. *Mark*) des erstellten Stundenplans.

4.1.2 Nebenbedingungen (Constraints)

- **Anzahl der Lehrveranstaltungen** (Gleichungen 2 und 3): Begrenzen die Gesamtzahl der gewählten Lehrveranstaltungen, wobei $C_{\min}$ und $C_{\max}$ die definierten Mindest- bzw. Maximalanzahlen darstellen.
- **Ausschlusskriterien (*Forbidden Pairs*)** (Gleichungen 4 und 5): Verhindern die Auswahl von sich gegenseitig ausschließenden Angeboten.
    1. $\mathcal{F}_{\text{Planpunkt}}$ (4): Enthält alle Paare (i, j), die dem selben Planpunkt angehören. Es darf maximal eine Lehrveranstaltung aus dem Planpunkt gewählt werden.
    2. $\mathcal{F}_{\text{Zeit}}$ (5): Enthält alle Paare (i, j), deren Termine sich zeitlich überlappen. Es darf auch hier maximal eine Lehrveranstaltung aus dem Paar gewählt werden.
- **Feste Zuordnungen** (Gleichungen 6 und 7):
    1. $\mathcal{M}_{\text{Muss}}$ (6): Erzwingt die Auswahl von Lehrveranstaltungen, die als zwingend notwendig (Priorität = 100) definiert sind.
    2. $\mathcal{M}_{\text{Blockiert}}$ (7): Verbietet die Auswahl von Lehrveranstaltungen, die nicht gewählt werden dürfen (Priorität = -100).

Durch das Lösen dieses Modells wird ein optimaler Satz an Entscheidungsvariablen y_i (Stundenplan) bestimmt, der die Zielfunktion (*Mark*) maximiert, während alle Nebenbedingungen erfüllt werden. Das Ergebnis ist der optimal mögliche Stundenplan gemäß der definierten Constraints.

4.1.3 CPU vs. GPU

Zwar läuft `PuLP` standardmäßig auf der CPU, jedoch ermöglicht eine einfache Anpassung die Nutzung einer NVIDIA-GPU über CUDA. Daher wurde das ILP-Modell sowohl auf der CPU als auch auf einer modernen GPU gelöst, um eine realistische Bewertung unter aktuellen technischen Voraussetzungen zu ermöglichen.

4.2 Implementierung der Hill-Climbing Algorithmen

Basierend auf der Methodik von Feldman und Golumbic [4] wurden die Optimierungsalgorithmen *Hill Climbing v1* und *Hill Climbing v3* implementiert. Das Ziel dieser Algorithmen ist es, inkrementell, durch lokal optimale Entscheidungen einen Stundenplan zu finden, der die Zielfunktion (*Mark*) maximiert.

4.2.1 Hill Climbing v1

Der *Hill Climbing v1*-Algorithmus wählt bei jedem Schritt das beste verfügbare *Offering* aus **allen verfügbaren Offerings** gemessen am *Mark* des Stundenplans inklusive des gewählten *Offering* [8].

4.2.2 Hill Climbing v3

Der *Hill Climbing v3*-Algorithmus unterscheidet sich vom *v1*-Ansatz dadurch, dass der Suchraum für einen neuen Kurs reduziert wird. Anstatt alle verfügbaren *Offerings* zu evaluieren, beschränkt *v3* die Auswahl auf eine Teilmenge der nach *Mark* sortierten Liste. Konkret wird in der Implementierung nur die zweite Hälfte — die Offerings mit der individuell besten *Mark* — der verfügbaren *Offerings* betrachtet [9].

4.2.3 Suchstrategie

Ausgehend vom aktuellen Stundenplan wird für jeden möglichen nächsten Zustand die *Mark* berechnet, indem der Stundenplan ergänzt um ein weiteres *Offering* durch die Evaluationsfunktion bewertet wird. Anschließend wird jenes Lehrangebot ausgewählt, das zu der höchsten Bewertung führt, und in den Stundenplan übernommen.

$$a^* = \arg \max_{a \in \text{available\_offerings}} (\text{get\_schedule\_mark}(\text{schedule} \cup \{a\})) \qquad (9)$$

Nur jene *Offerings* werden in die Auswahl aufgenommen, die keine Nebenbedingungen verletzen (zeitliche Überschneidung, Planpunkt bereits erfüllt, *Offering* bereits gewählt, *Offering* in Liste verbotener Kurse). Die Schleife wird fortgesetzt, solange ein gültiges *Offering* gefunden wird, das zu einer Verbesserung oder Erweiterung des Stundenplans führt.

4.2.4 Abbruchkriterien

Der Algorithmus bricht ab, wenn einer der folgenden Zustände eintritt:

1. Der Stundenplan ist valide und hat die maximale Anzahl von Lehrveranstaltungen erreicht.

2. Der Stundenplan ist valide, und es kann keine weitere gültige *Offering* gefunden werden, ohne dass eine Nebenbedingung verletzt wird.

3. Es kann keine weitere *Offering* gefunden werden, die keine Constraints verletzt. Wenn die minimale Anzahl der Lehrveranstaltungen noch nicht erfüllt ist, führt das zu einem ungültigen Ergebnis.

Algorithm 2 Hill Climbing

```
1: schedule ← ∅
2: availableOfferings ← FilterConflicts(offerings)
3: while availableOfferings ≠ ∅ do
4:     nextOffering ← arg max_{o∈availableOfferings} Mark(schedule ∪ {o})
5:     if IsValid(schedule ∪ {nextOffering}) then
6:         schedule ← schedule ∪ {nextOffering}
7:     end if
8:     if |schedule| = MaxScheduleLength then
9:         return schedule
10:    end if
11:    availableOfferings ← FilterConflicts(offerings)
12: end while
13: return schedule
```

4.2.5 Zur Nicht-Implementierung von Hill Climbing v2

Der von Feldman und Golumbic vorgeschlagene *Hill Climbing v2*-Algorithmus priorisiert zusätzlich auf der Ebene von *Gruppen* (*Groups*). Im Originalkontext des Papers beziehen sich Gruppen auf die Unterscheidung zwischen Pflichtveranstaltungen (*Mandatory*) und Wahlveranstaltungen (*Elective*), wobei die Priorität der Gruppen die Auswahl von Pflichtveranstaltungen vor Wahlveranstaltungen erzwingt [4]. Da es im Hauptstudium Wirtschaftsinformatik keine Unterscheidung zwischen Pflicht- und Wahlveranstaltungen gibt, wäre die Implementierung funktional identisch mit *v1*.

4.3 Implementierung des Offering-Order Algorithmus

Basierend auf der Methodik von Feldman und Golumbic [4] wurde der Optimierungsalgorithmus *Offering Order* implementiert, der auf einer heuristischen Sortierung basiert.

4.3.1 Sortierung

Der Algorithmus nutzt die berechneten *Marks*, um eine Suchreihenfolge festzulegen:

1. Alle *Offerings* innerhalb des selben Planpunkts werden absteigend nach *Mark* sortiert. Innerhalb eines Planpunkts wird somit das Offering mit dem höchsten *Mark* zuerst in Betracht gezogen.

2. Die Planpunkte selbst werden nach dem *Mark* des besten *Offering* innerhalb des Planpunkts sortiert.

4.3.2 Suchstrategie

Der optimale Stundenplan wird durch eine *Forward-Checking Backtracking* Strategie gesucht. Es wird folgendermaßen vorgegangen:

1. Die Planpunkte werden iterativ durchlaufen. Gestartet wird mit dem Planpunkt, der das *Offering* mit der höchsten *Mark* beinhaltet (siehe 4.3.1).

2. Für jeden Planpunkt wird versucht, das *Offering* mit der höchsten *Mark* auszuwählen. Bevor die Auswahl getroffen wird, wird geprüft, ob das Hinzufügen des *Offerings* zu dem aktuellen Teilstundenplan zu einer zeitlichen Überschneidung führt oder andere Constraints verletzt (**Forward-Checking**)

3. Führt das Hinzufügen eines *Offerings* zu keinem gültigen Endergebnis in den nachfolgenden Planpunkten, wird die Auswahl rückgängig gemacht und das nächstbeste *Offering* der aktuellen Gruppe wird getestet (**Backtracking**)

Algorithm 3 Offering Order Algorithm

```
1: schedule ← ∅
2: i ← 0
3: loop
4:     select an offering o ∈ groups_i
5:     if there is no o then
6:         backtrack to preceding state
7:     else
8:         schedule ← schedule ∪ {o}
9:         for all groups_j (i < j ≤ n) do
10:            remove from groups_j all violating offers
11:        end for
12:        if any groups_j becomes empty then
13:            backtrack to preceding state
14:        else
15:            i ← i + 1
16:            if i = n then
17:                return schedule
18:            end if
19:        end if
20:    end if
21: end loop
```

4.4 Szenariendefinition

Um die beschriebenen Algorithmen unter realistischen Bedingungen zu testen, wurden folgende Szenarien definiert:

4.4.1 Freier Tag

Laut einer Umfrage an der Hochschule Trier gaben 81% der Studierenden an, gerne einen komplett freien Tag zu haben, um einem Nebenjob nachzugehen. 13% der Studierenden gaben an, jeden Tag erst um 9:45 Uhr beginnen zu wollen, und 6% möchten einen oder mehrere freie Vormittage haben. Dabei wurde am häufigsten der Freitag (31%) vor dem Montag (6%) gewünscht. 64% der Studierenden wäre es egal, welchen Tag sie freibekommen [7]. Aus diesen Ergebnissen wurden folgende Szenarien abgeleitet:

- Keine Kurse an Montagen
- Keine Kurse an Freitagen
- Jeder Tag der Woche bis 9:45 frei
- Keine Kurse an Donnerstagen und Freitagen

4.4.2 Bliebte Zeiten

Laut einer weiteren Umfrage [3] bevorzugen 74% der Studierenden Prüfungen und Kurse, die über das Semester verteilt stattfinden. Teilt man den Tag in die drei Zeitslots Vormittag, Nachmittag und Abend ein, so war der Nachmittag der beliebteste Zeitslot, gefolgt vom Vormittag und dem Abend. Aus diesen Ergebnissen wurden folgende Szenarien abgeleitet:

- Erhöhte Priorität (+90) für Nachmittags, negative Priorität (−90) für Abend
- Erhöhte Priorität (+70) für Vormittag, neutrale Priorität (0) für Nachmittag, negative Priorität (−80) für Abend

4.4.3 Ähnlicher Problemstellungen

Hinzu kommen klassische Constraints, die in University-Course-Timetabling-Problemen häufig vorkommen [2]. Diese Constraints betreffen meist die universitäre Planung des Curriculums und die Zuteilung von Kursen zu Räumen, es gibt jedoch auch Constraints, die die Erstellung individueller Stundenpläne der Studierenden betreffen:

- Studierende müssen von 12:00 bis 13:00 Zeit für ein Mittagessen haben
- Studierende müssen mehr als einen Kurs pro Tag besuchen
- Studierende dürfen nicht mehr als drei Kurse pro Tag besuchen

4.4.4 Vergleichbarkeit

Zusätzlich zu den oben genannten Constraints sollen die Algorithmen auch in Extremfällen hinsichtlich ihrer Performance getestet werden. Daher werden ergänzend folgende Szenarien definiert, die in verwandten Arbeiten nicht explizit untersucht wurden, jedoch zur Vergleichbarkeit zwischen den Algorithmen sinnvoll sind:

- Alle Kurse können frei von Constraints gewählt werden.
- 50% aller Kurse können nicht mehr belegt werden.
- 7 Kurse sind bereits vorgegeben
- Keine Kurse zwischem 6:00 morgens und 13:00.
- Keine Kurse Montags, Dienstags und Mittwochs

4.5 Versuchsaufbau

Das Paper von Feldman und Golumbic [4] wählte einen Brute-Force-Algorithmus als Baseline für Performance-Vergleiche. Da ein Brute-Force Algorithmus bei der großen Anzahl der Vorlesungen im VVZ der WU eine zu lange Laufzeit hätte, wurde eine Integer Linear Programming (ILP) Lösung als neue Baseline gewählt. Diese gewährleistet, ebenso wie der Brute-Force-Ansatz für kleine Instanzen, die Ermittlung des optimalen Stundenplans (mit der höchsten *Mark*). Die Ergebnisse der heuristischen Algorithmen (*Hill Climbing v1*, *Hill Climbing v3*, *Offering Order*) werden folglich relativ zur optimalen Lösung des ILP-Ansatzes bewertet. Zusätzlich zu den heuristischen Algorithmen wird auch eine *GPU-optimierte Variante des ILP-Ansatzes* getestet.

Als Maß für die Schwierigkeit (*Difficulty*) der Probleminstanz wird in dieser Arbeit nicht die Laufzeit des Brute-Force-Algorithmus verwendet, sondern die Anzahl der zu planenden Kurse (N).

Die Algorithmen wurden anhand der folgenden Leistungskennzahlen evaluiert:

1. **Mark**: Die Qualität der gefundenen Lösung (in Prozent relativ zum Baseline Ansatz).
2. **Laufzeit**: Die zur Lösungsfindung benötigte Zeit.
3. **Speicherbedarf**: Der benötigte Hauptspeicher.

4.5.1 Zielfunktion

Die *Mark* dient als Bewertungsfunktion für die Qualität eines erstellten Stundenplans und entspricht der im Paper von Feldman und Golumbic [4] definierten Zielfunktion. Sie quantifiziert, wie gut ein Stundenplan die Präferenzen und Constraints der Studierenden erfüllt. Die Berechnung erfolgt ausschließlich für *gültige* Stundenpläne, d. h. solche, die keine restriktiven Constraints ($|p| = P$) verletzen.

Jede Präferenz oder Restriktion ist im Modell durch eine *Priorität* p im Wertebereich $[-P, P]$ definiert. Positive Prioritäten kennzeichnen wünschenswerte Eigenschaften (z. B. bevorzugte Lehrveranstaltungen oder Zeitfenster), negative Prioritäten hingegen unerwünschte. Die Stärke des Präferenzwerts ist proportional zum Absolutwert der Priorität.

Die *Mark* eines gültigen Stundenplans S ergibt sich aus der Summe der positiven Beiträge durch gewählte Kurse und den Abzügen für Verletzungen sogenannter *fixed* und *non-fixed* Constraints:

$$Mark(S) = \underbrace{\sum_{c \in C_S} q_c}_{\text{Kursprioritäten}} - \underbrace{\sum_{h \in H^{(f)}_{\text{violated}}} |p^{(f)}_h|}_{\text{Strafen für fixed-Constraints}} - \underbrace{\sum_{t \in T^{(n)}_{\text{violated}}} v_t \cdot |p^{(n)}_t|}_{\text{Strafen für non-fixed-Constraints}} , \tag{10}$$

wobei gilt:

- C_S: Menge aller im Stundenplan S enthaltenen Kurse.
- q_c: Priorität des Kurses c.
- $H^{(f)}_{\text{violated}}$: Menge aller Stunden, in denen ein *fixed*-Constraint verletzt wurde.
- $p^{(f)}_h$: Priorität der Stunde h. Eine Verletzung liegt vor, wenn
  - $p^{(f)}_h < 0$ und die Stunde aktiv ist (der Studierende hat in einer ungewünschten Stunde Unterricht), oder
  - $p^{(f)}_h > 0$ und die Stunde inaktiv ist (der Studierende hat in einer gewünschten Stunde keinen Unterricht).
- $T^{(n)}_{\text{violated}}$: Menge der verletzten *non-fixed*-Constraints.
- $p^{(n)}_t$: Priorität des Constraints t.
- v_t: Anzahl der verletzten Stunden (bzw. des Zeitumfangs), der durch Constraint t betroffen ist.

Je höher der Wert von $M(S)$, desto besser erfüllt der Stundenplan die angegebenen Präferenzen. Ziel der Optimierungsalgorithmen ist daher die Maximierung von $M(S)$.

4.5.2 Durchführung

Die experimentelle Evaluation der Algorithmen erfolgte durch systematische Benchmarks, die Performance und Lösungsqualität der implementierten Ansätze unter verschiedenen Szenarien und Schwierigkeitsgraden testet.

Die Experimente wurden auf einer Virtual Private Server (VPS)-Instanz des Anbieters Hetzner durchgeführt, welche über eine x86 AMD CPU verfügte. Das gewährleistet eine konsistente und dedizierte Umgebung für die Messungen.

Die Algorithmen wurden in TODO (N) verschiedenen Szenarien getestet. Jedes Szenario stellt eine vorkonfigurierte Zusammensetzung von Constraints dar (definiert in JSON-Konfigurationsdateien, z.B. `constraint1.json`).

Der Schwierigkeitsgrad (*Difficulty*) einer Probleminstanz wird durch die Anzahl der zu planenden Kurse N definiert, im Gegensatz zur Laufzeit des Brute-Force-Algorithmus, die in der Originalarbeit verwendet wurde. Die Tests wurden inkrementell für jede mögliche Kursanzahl N, beginnend bei $N = 1$ bis zur maximal möglichen Kursanzahl im jeweiligen Szenario, durchgeführt. Die maximale Kursanzahl wurde dabei vorab mithilfe des ILP-Baseline-Ansatzes ermittelt.

Für jeden Algorithmus ($A \in \{\text{ILP, Offering Order, Hill Climbing V1, Hill Climbing V3}\}$), für jede Kursanzahl N und für jedes Szenario S wurde die Messreihe wie folgt durchgeführt:

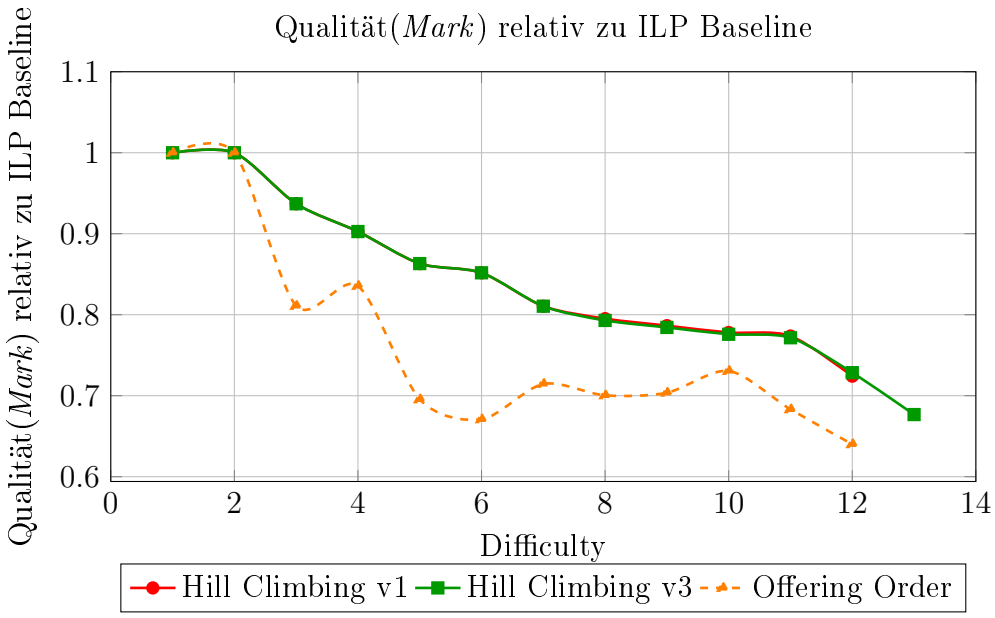
1. Jeder Algorithmus wurde für eine feste Kursanzahl N 50 Mal unabhängig voneinander ausgeführt.
2. Die 50 Messungen pro Algorithmus und Schwierigkeitsgrad wurden sequenziell durchgeführt, um sicherzustellen, dass die Messergebnisse für die Laufzeit und den Speicherbedarf unabhängig sind.
3. Es wurden folgende Metriken für jede der 50 Ausführungen erfasst:
   - Laufzeit (t): Die zur Lösungsfindung benötigte Zeit (in Sekunden).
   - Hauptspeicher (M): Der während der Ausführung belegte Hauptspeicher (in Megabyte).
   - Qualität (*Mark*)
4. Für die 50 Messwerte der Laufzeit und des Hauptspeichers wurden folgende statistische Kennzahlen berechnet:
   - Median ($t_{\text{median}}, M_{\text{median}}$)
   - Mittelwert ($t_{\text{mean}}, M_{\text{mean}}$)
   - Minimalwert ($t_{\text{min}}, M_{\text{min}}$)
   - Maximalwert ($t_{\text{max}}, M_{\text{max}}$)
   - Standardabweichung ($t_{\text{stdev}}, M_{\text{stdev}}$)

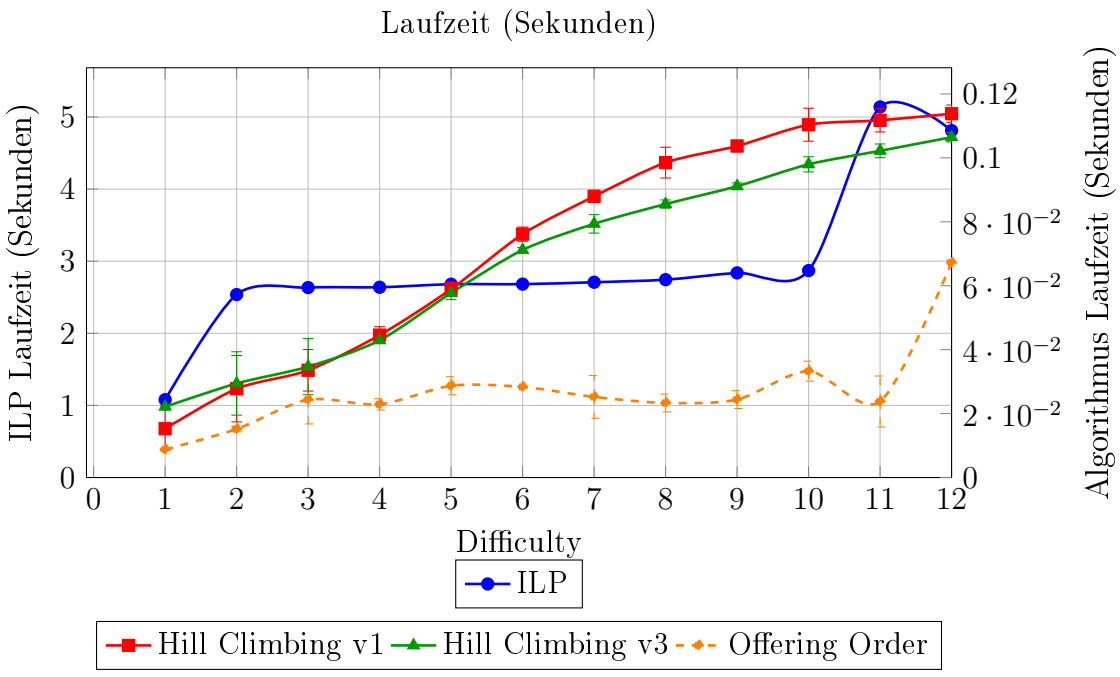
Um die Algorithmen vergleichbar zu halten, wurde der gemessene Zeit- und Speicherbedarf auf die Ausführung des Algorithmus selbst begrenzt. Das Preprocessing — insbesondere die Vorabberechnung der *Mark* für jedes Offering, die einmalig vor der Hauptschleife durchgeführt wird — ist nicht in den Messwerten enthalten.

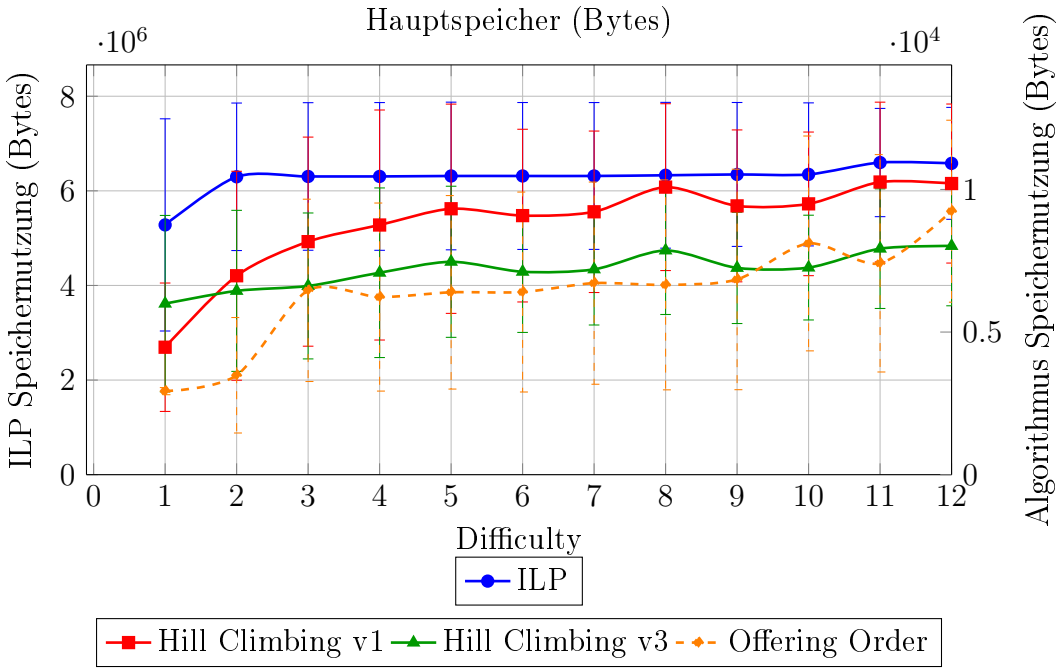
5 Ergebnisse

Szenario 1: Freie Kurswahl Der Algorithmus kann aus allen Kursen des Hauptstudiums wählen. Keine Constraints hinsichtlich Kursanzahl, Wochenstunden oder Priorität wurden gesetzt.

```
{
    "FIXED_TIME_CONSTRAINTS": null,
    "COURSE_PRIORITY_CONSTRAINTS": null,
    "HOUR_LOAD_CONSTRAINT": null,
    "TOTAL_COURSE_COUNT_CONSTRAINT": null
}
```

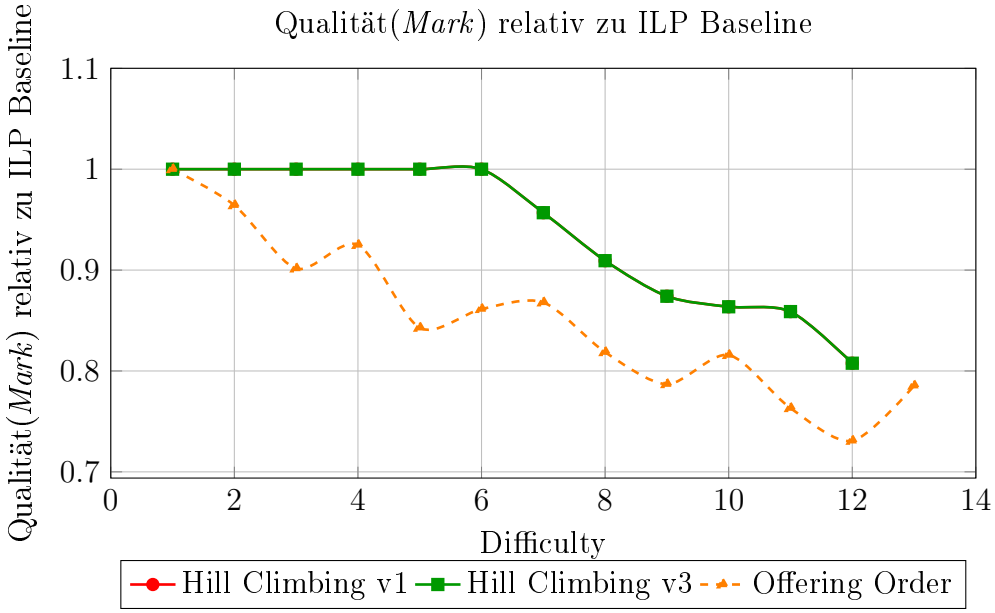


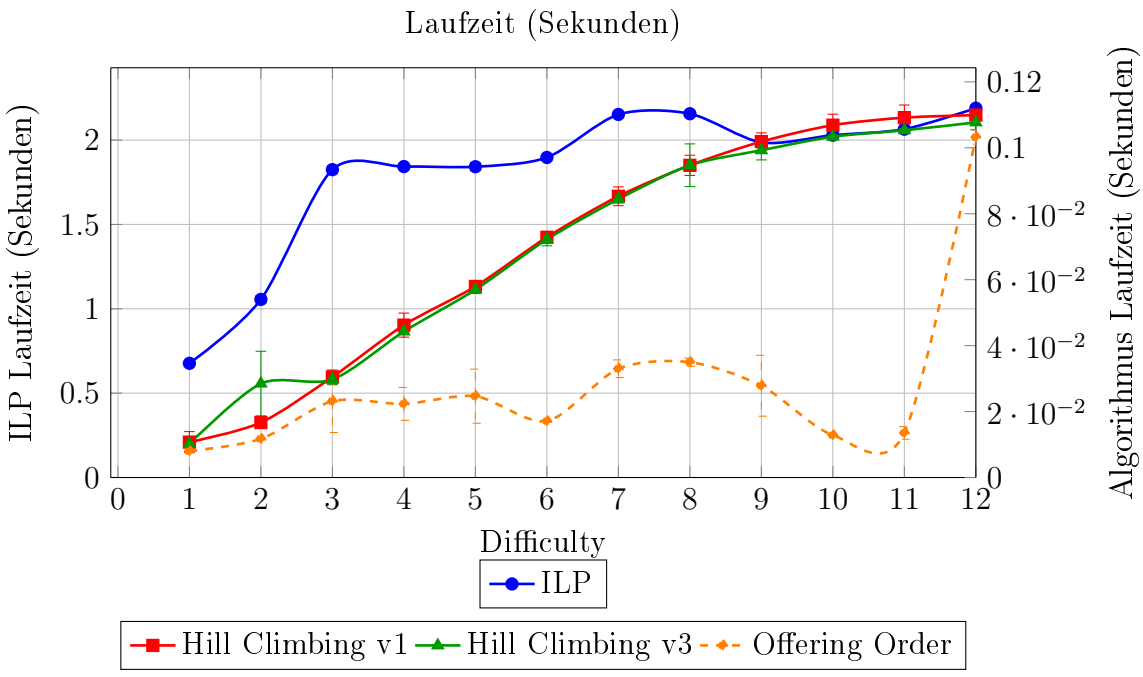

Laufzeit (Sekunden)
ILP Laufzeit (Sekunden)
Algorithmus Laufzeit (Sekunden)
0
1
2
3
4
5
0.12
0.1
$8 \cdot 10^{-2}$
$6 \cdot 10^{-2}$
$4 \cdot 10^{-2}$
$2 \cdot 10^{-2}$
0
0 1 2 3 4 5 6 7 8 9 10 11 12
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

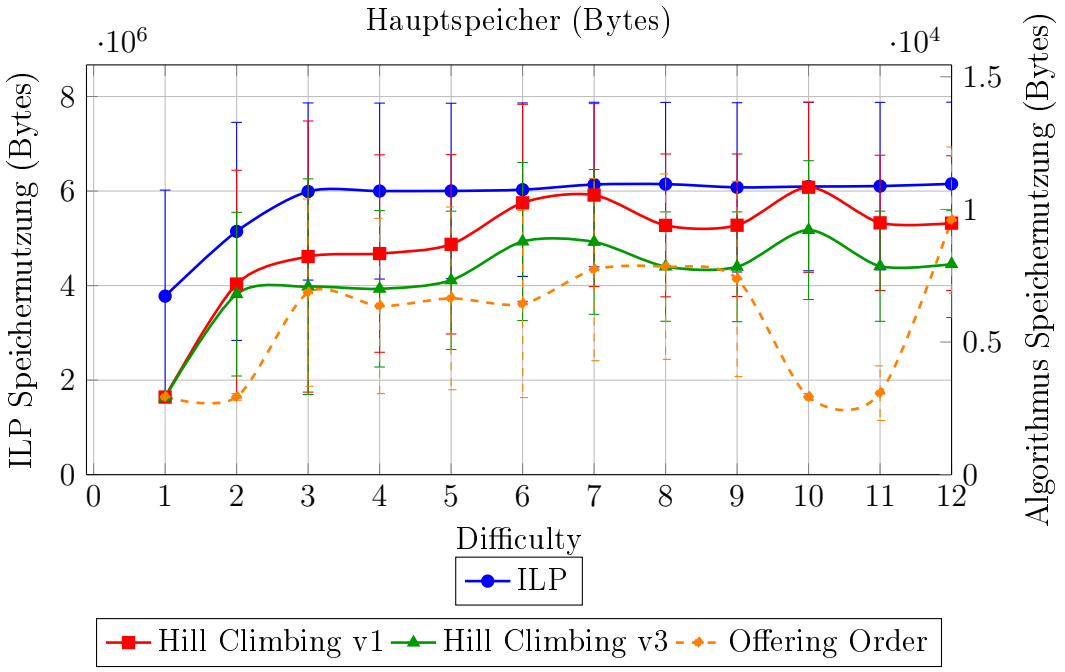

Hauptspeicher (Bytes)
$\cdot 10^6$
$\cdot 10^4$
ILP Speichernutzung (Bytes)
Algorithmus Speichernutzung (Bytes)
0
2
4
6
8
1
0.5
0
0 1 2 3 4 5 6 7 8 9 10 11 12
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

Szenario 2: 3 Kurse gesetzt Der Studierende muss eine bestimmte Anzahl von Kursen wiederholen. Diese sind mit einer Priorität von 100 markiert und verletzen keine Hard Constraints. Abgesehen davon wurden keine Constraints gesetzt.

```
{
    "FIXED_TIME_CONSTRAINTS": null,
    "COURSE_PRIORITY_CONSTRAINTS": {
        "5576": 100, # ADP
        "5033": 100, # Makro
        "4010": 100, # Statistik
    },
    "HOUR_LOAD_CONSTRAINT": null,
    "TOTAL_COURSE_COUNT_CONSTRAINT": null
}
```

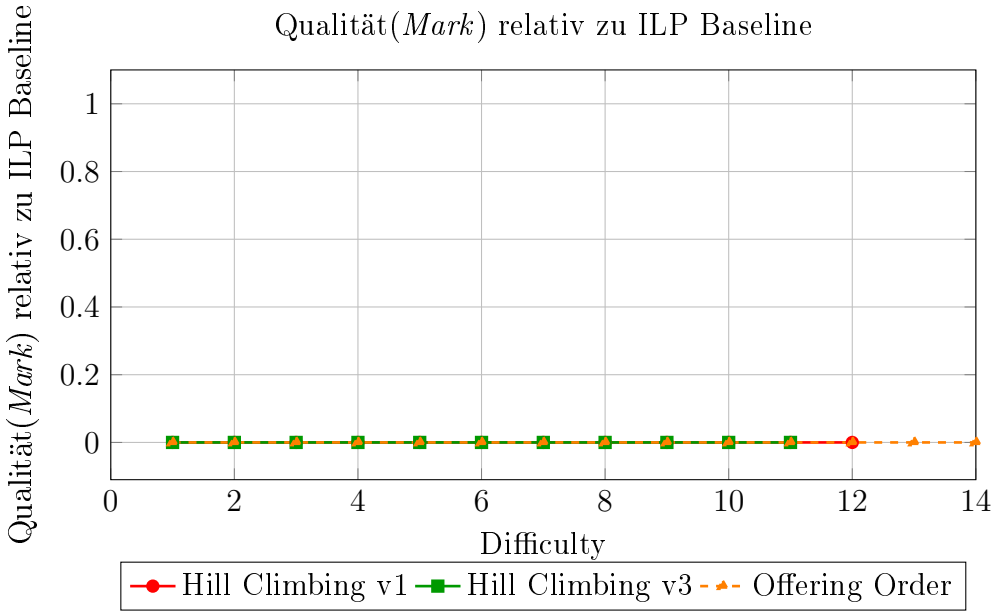


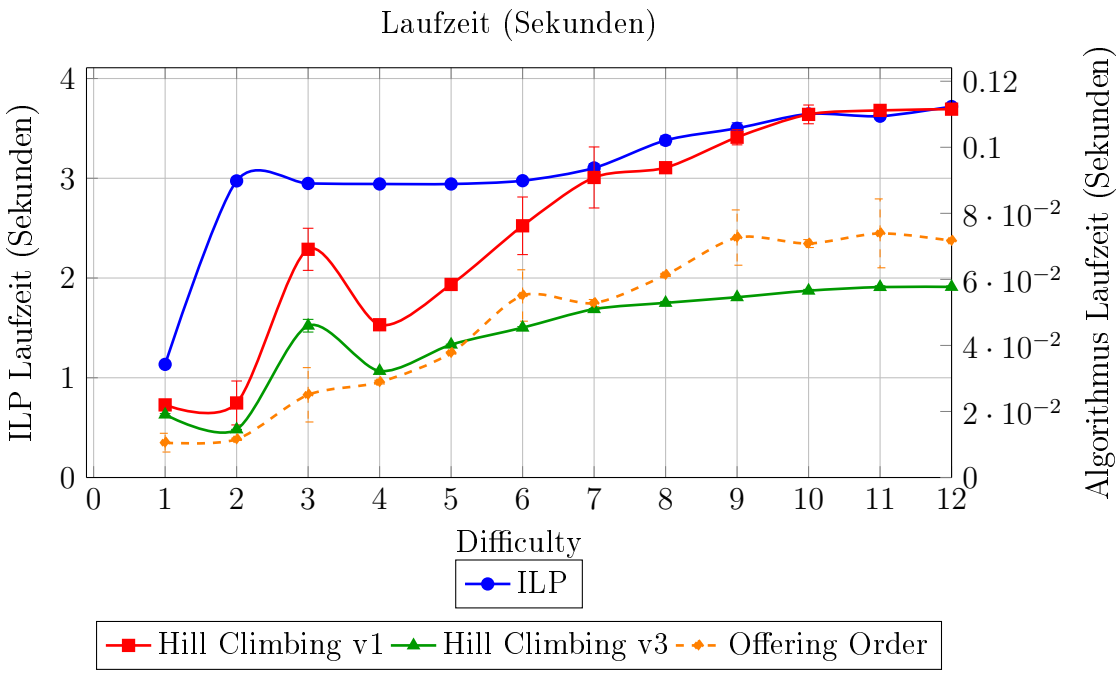

Laufzeit (Sekunden)
ILP Laufzeit (Sekunden)
Algorithmus Laufzeit (Sekunden)
0
0.5
1
1.5
2
0
$2 \cdot 10^{-2}$
$4 \cdot 10^{-2}$
$6 \cdot 10^{-2}$
$8 \cdot 10^{-2}$
0.1
0.12
0 1 2 3 4 5 6 7 8 9 10 11 12
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

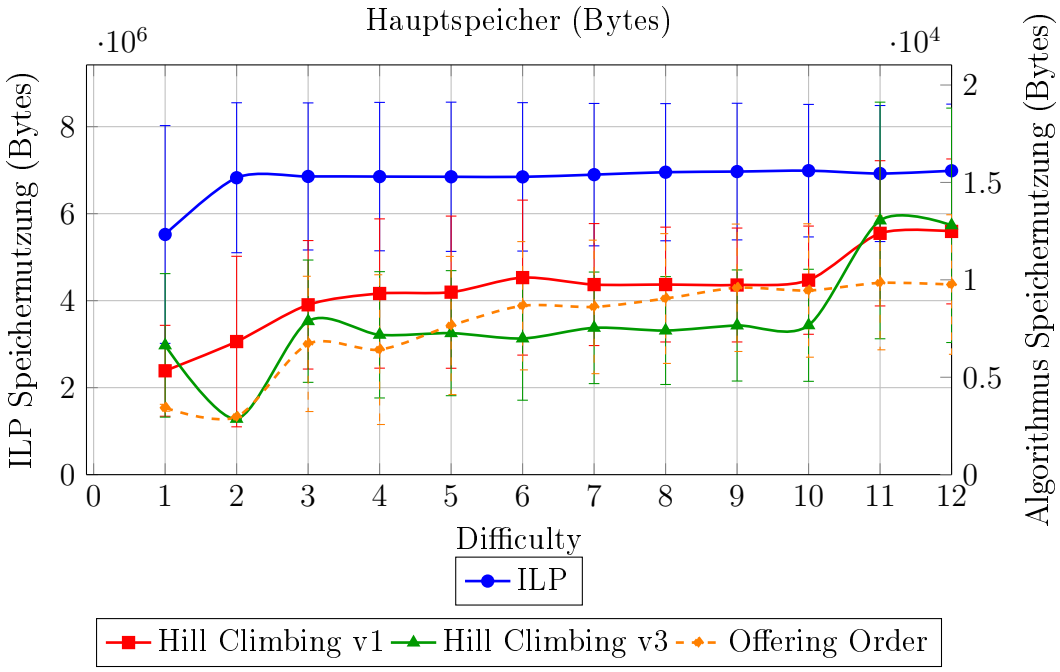

Hauptspeicher (Bytes)
$\cdot 10^6$
$\cdot 10^4$
ILP Speichernutzung (Bytes)
Algorithmus Speichernutzung (Bytes)
0
2
4
6
8
0
0.5
1
1.5
0 1 2 3 4 5 6 7 8 9 10 11 12
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

Szenario 3: 20% der Kurse blockiert 60 Kurse sind blockiert (entspricht 20% der Kurse). Diese sind mit einer Priorität von -100 markiert. Abgesehen davon wurden keine Constraints gesetzt.


```
{
    "FIXED_TIME_CONSTRAINTS": null,
    "COURSE_PRIORITY_CONSTRAINTS": {
        "6480": -100,
        "6353": -100,
        "6275": -100,
        ...
        "5722": -100,
        "6345": -100,
        "6232": -100
    },
    "HOUR_LOAD_CONSTRAINT": null,
    "TOTAL_COURSE_COUNT_CONSTRAINT": null
}
```

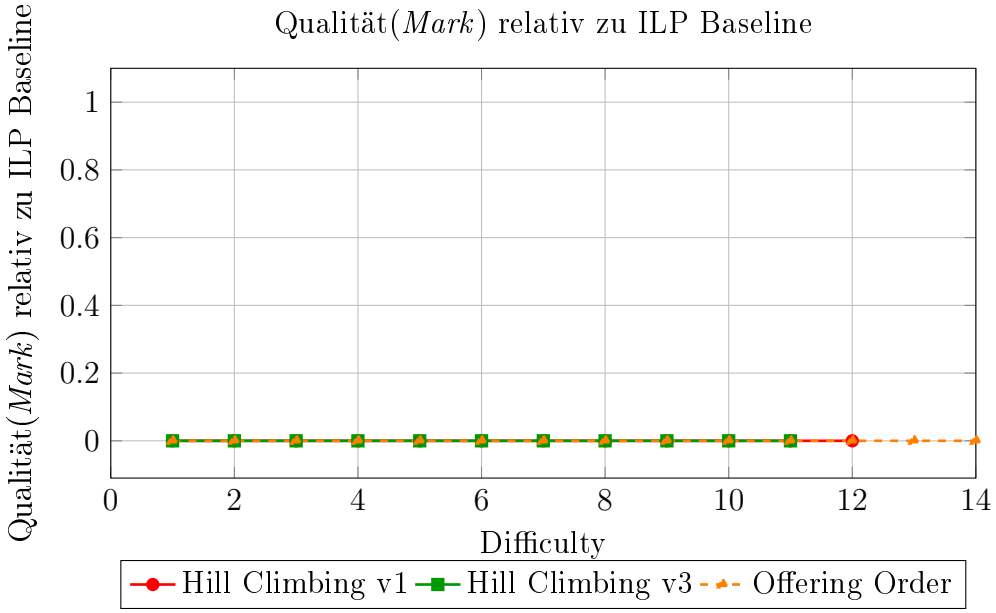



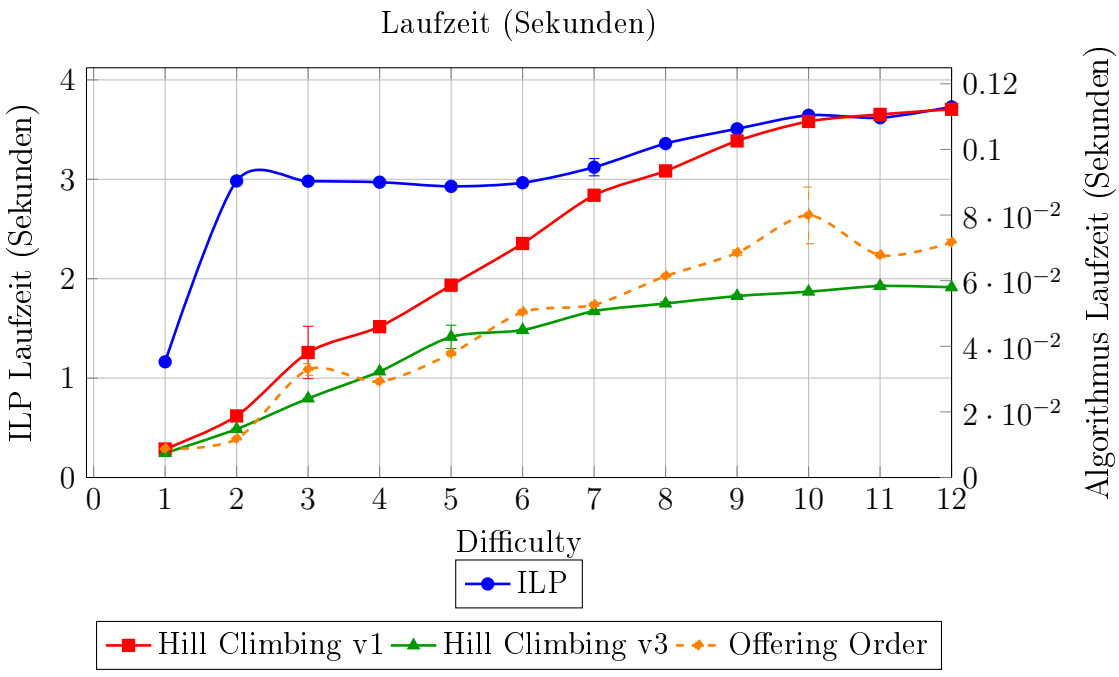

Laufzeit (Sekunden)
ILP Laufzeit (Sekunden)
Algorithmus Laufzeit (Sekunden)
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

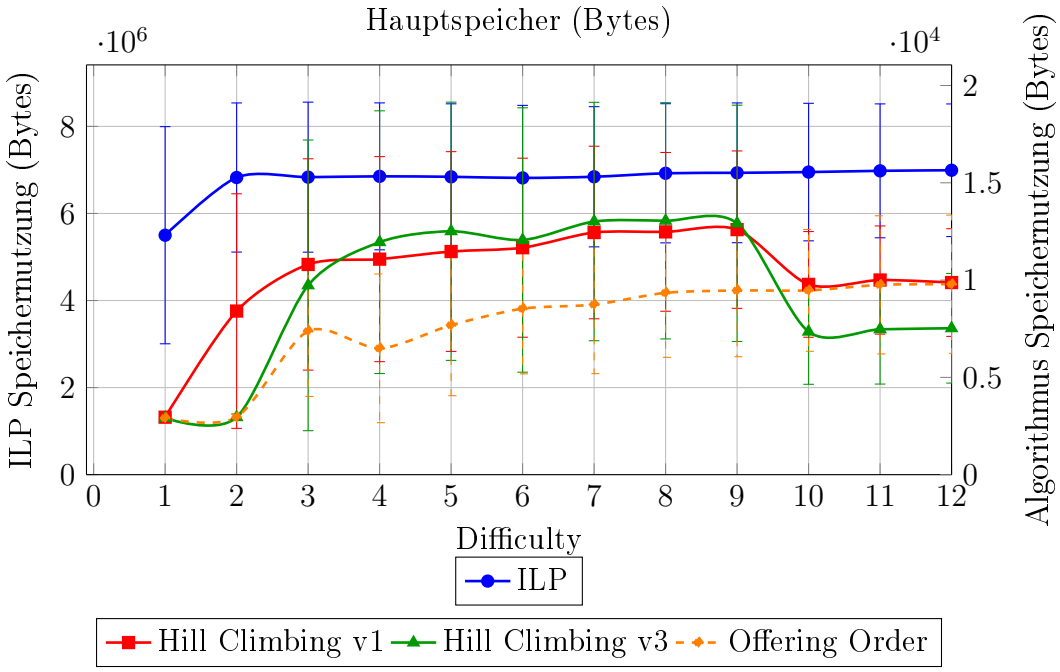

Hauptspeicher (Bytes)
ILP Speichernutzung (Bytes)
Algorithmus Speichernutzung (Bytes)
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

Szenario 4: Tage blockiert Montage und Dienstage sind blockiert. Der Student möchte Vormittags und Samstags lernen.

```
{
    "FIXED_TIME_CONSTRAINTS": [
        ["monday", 1, 23],
        ["tuesday", 1, 23],
        ["wednesday", 19, 23],
        ["thursday", 19, 23],
        ["friday", 19, 23]
    ],
    "COURSE_PRIORITY_CONSTRAINTS": null,
    "HOUR_LOAD_CONSTRAINT": null,
    "TOTAL_COURSE_COUNT_CONSTRAINT": null
}
```




Laufzeit (Sekunden)
ILP Laufzeit (Sekunden)
Algorithmus Laufzeit (Sekunden)
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order


Hauptspeicher (Bytes)
ILP Speichernutzung (Bytes)
Algorithmus Speichernutzung (Bytes)
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

Todo: Szenario 3 und 4 Score Chart nicht korrekt. Weitere Szenarien (10-15 gesamt), Diskussion mit Ergebnissen aus dem Paper [4]

6 Diskussion

6.1 Vergleich der Ergebnisse mit Ursprungsarbeit?

6.2 Einschränkungen

7 Fazit

7.1 Verwandte Arbeiten

7.2 Anwendungsfälle und Weiterentwicklung

7.3 Schlussworte

References

[1] Vorlesungsverzeichnis der wirtschaftsuniversität wien. `https://vvz.wu.ac.at/`.

[2] Mei Ching Chen, San Nah Sze, Say Leng Goh, Nasser R. Sabar, and Graham Kendall. A survey of university course timetabling problem: Perspectives, trends and opportunities. *IEEE Access*, 9:106515–106529, 2021.

[3] Peter Cowling, Graham Kendall, and Naimah Mohd Hussin. A survey and case study of practical examination timetabling problems. *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002*, 01 2002.

[4] R. Feldman and M. C. Golumbic. Optimization Algorithms for Student Scheduling via Constraint Satisfiability. *The Computer Journal*, 33(4):356–364, 4 1990.

[5] Sheldon H. Jacobson and Enver Yücesan. Analyzing the Performance of Generalized Hill Climbing Algorithms. *Journal of Heuristics*, 10(4):387–405, 7 2004.

[6] Ahmad Muklason, Andrew J. Parkes, Ender Özcan, Barry McCollum, and Paul McMullan. Fairness in examination timetabling: Student preferences and extended formulations. *Applied Soft Computing*, 55:302–318, 1 2017.

[7] Hochschule Trier Trier University of Applied Sciences. Evaluation zu den Freiräumen im Stundenplan. 1 2024.

[8] Philipp Scheer. `https://github.com/philippscheer/bachelorarbeit/blob/main/models/hill_climbing_v1.py`.

[9] Philipp Scheer. `https://github.com/philippscheer/bachelorarbeit/blob/main/models/hill_climbing_v3.py`.

[10] Öh-wu studienplaner.