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

Das Student Scheduling Problem (SSP), also 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.

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*" [1] beschreiben, um diese anschließend miteinander zu vergleichen.

2.2 Motivation

2.3 Zielsetzung

Die vorliegende Bachelorarbeit verfolgt das Ziel, die beschriebenen Optimierungsalgorithmen unter aktuellen Bedingungen und mit einem aktuellen Datensatz zu validieren und einen Leistungsvergleich anzustellen.

1. Validierung: Zentral ist die exakte Nachprogrammierung der Algorithmen von Feldman und Golumbic [1]. Es soll überprüft werden, ob diese Algorithmen auch unter modernen Rechenbedingungen und mit realen Daten aus dem Vorlesungsverzeichnis der Wirtschaftsuniversität Wien einen optimalen Semesterplan erstellen können.

2. Vergleichende Leistungsanalyse: Die implementierten Algorithmen werden hinsichtlicher der Qualität ("Score") der gefundenen Lösung, Geschwindigkeit und Speichernutzung verglichen.

2.4 Zeittafelprobleme

2.4.1 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 [2].

2.4.2 Offering-Order Algorithmus

Der Offering-Order Algorithmus, welcher von Feldman und Golumbic [1] 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.

2.4.3 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.

2.7 Aufbau

Die vorliegende Arbeit gliedert sich in vier Hauptkapitel:

Nach dieser Einleitung, welche Problemstellung, Zielsetzung und Abgrenzung erklärt, folgt das Kapitel Hintergrund und theoretische Grundlagen. Hier werden die Struktur des Studiums an der WU und die daraus resultierenden Herausforderungen bei der Stundenplanerstellung erläutert. Weiters werden die Grundlagen der Optimierung und Constraint-Satisfaction Probleme als theoretische Basis erklärt.

Das zentrale Kapitel Implementierung beschreibt die notwendige Datenbeschaffung aus dem Vorlesungsverzeichnis. Anschließend wird die technische Implementierung der Algorithmen in Python detailliert erklärt.

Das Kapitel Zusammenfassung und Arbeit zeigt die Ergebnisse der vergleichenden Analyse der implementierten Algorithmen hinsichtlich der Lösungsqualität, Geschwindigkeit und Speichernutzung. Die Ergebnisse der Analyse werden mit den Ergebnissen im Paper verglichen. Die Arbeit schließt mit dem Fazit und der Skizzierung möglicher Weiterentwicklungen und Anwendungsfälle.

3 Datenbereitstellung

3.1 Erhebung der Vorlesungsdaten

Die Datenextraktion der Vorlesungsdaten erfolgte über einen automatisierten Web-Scraping-Prozess, implementiert in Python 3.11 unter Verwendung der Bibliotheken `requests` (HTTP-Requests), `BeautifulSoup` (HTML-Parsing) und `pandas` (Datenstrukturierung). Der Prozess umfasste folgende, drei Phasen:

1. Extraktion relevanter Datenpunkte für alle Vorlesungen der Wirtschaftsuniversität Wien,
2. Identifikation relevanter Lehrveranstaltungen (Planpunkte) und
3. ECTS-Credits-Zuordnung gemäß Studien-Planpunkt.

3.1.1 Web-Scraping von Vorlesungsdaten

Jede Lehrveranstaltung an der Wirtschaftsuniversität Wien wird durch eine eindeutige LV-ID identifiziert. LV-IDs werden durch eine Zahl von 0 bis 9999 dargestellt. Jede Lehrveranstaltung kann unter folgender URL mit Angabe des Semesters (Sommersemester 25 = `"S25"`) abgerufen werden:

```
https://vvz.wu.ac.at/cgi-bin/vvz.pl?C=S&LANG=DE\
  &U=H&S=25S&LV=3&L2=S&L3=S&T=&L=&I=1337&JOIN=AND
```

Beispiel: Abfrage der Lehrveranstaltung mit ID 1337 im Semester "25S".

Aus der HTTP-Antwort jeder Anfrage wurden mit `BeautifulSoup4` folgende Eigenschaften ausgelesen:

- Zeitliche Parameter: Termine, Dauer
- Räumliche Parameter: Raumzuweisungen (regex-bereinigt)
- Personelle Zuordnung: LV-Leiter:innen
- Curriculare Verknüpfungen: Planpunkte-ID via RegEx `r"P=([0-9]+);"` aus URL extrahiert

Für jede Planpunkt-ID erfolgte ein zweiter Scraping-Durchlauf, um die Zahl der ECTS Credits zu ermitteln. Die finalen Daten wurden als pandas-DataFrame mit folgenden Spalten strukturiert:

- `id`: Eindeutige LV-Kennung

- `dates`: Liste von Dicts mit Start/End-Zeitstempeln
- `lvLeiter`: Verantwortliche Lehrperson
- `planpunkte_ids`: Planpunkt-IDs
- `ects`: Vom Planpunkt abgeleitete ECTS-Punkte

3.1.2 Datenbereinigung

Das Ergebnis sieht wie folgt aus:

courseId	dates	lvLeiter	planpunkte_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 Pandas DataFrame der VVZ-Struktur

3.2 Studienstruktur und Herausforderungen bei der Stundenplanerstellung

Todo

4 Analyse

4.1 Implementierung des ILP Baseline Ansatzes

Feldman und Golumbic vergleichen in Ihrem Paper [1] 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. Im Folgenden wird das mathematische Modell des ILP präsentiert.

Modell des Integer Linear Programming (ILP)

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

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 Lehrveranstaltung 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. Der Nutzen Mark(i) wird hierbei als folgende Funktion definiert: TODO (Berechnung von Mark aus dem Paper übernehmen)

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.

4.2 Implementierung der Hill-Climbing Algorithmen

Basierend auf der Methodik von Feldman und Golumbic [1] 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*.

4.2.2 Suchstrategie

Kernkomponente des Algorithmus ist die Funktion `schedule_course`. Diese Funktion berechnet die *Mark* jedes möglichen nächsten Zustandes, indem sie die *Mark* des aktuellen Stundenplans, ergänzt um ein verfügbares *Offering* (schedule $\cup \{a\}$), berechnet. Das *Offering*, das den Stundenplan mit der höchsten *Mark* erzeugt, wird gewählt und zum Stundenplan hinzugefügt.

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

Dabei werden vorab nur jene *Offerings* in die Auswahl (`available_offerings`) aufgenommen, welche die Nebenbedingungen nicht 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.

4.2.3 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.

4.2.4 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.

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. 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 [1] 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**)

4.4 Vergleich der Algorithmen

4.4.1 Versuchsaufbau

Das Paper von Feldman und Golumbic [1] 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, Offering Order) werden folglich relativ zur optimalen Lösung des ILP-Ansatzes bewertet.

Die folgenden Algorithmen wurden im Rahmen dieser Arbeit verglichen:

- **ILP-Ansatz** (Baseline)
- **Hill Climbing v1**: Eine Hill-Climbing-Variante, die den gesamten Suchbereich berücksichtigt
- **Hill Climbing v3**: Eine modifizierte Hill-Climbing-Variante, die den Suchbereich um die Hälfte reduziert.
- **Offering Order**: TODO

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.

4.4.2 Berechnung der Mark

Die *Mark* dient als Bewertungsfunktion für die Qualität eines erstellten Stundenplans und entspricht der im Paper von Feldman und Golumbic [1] 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 $M(S)$.

4.4.3 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:

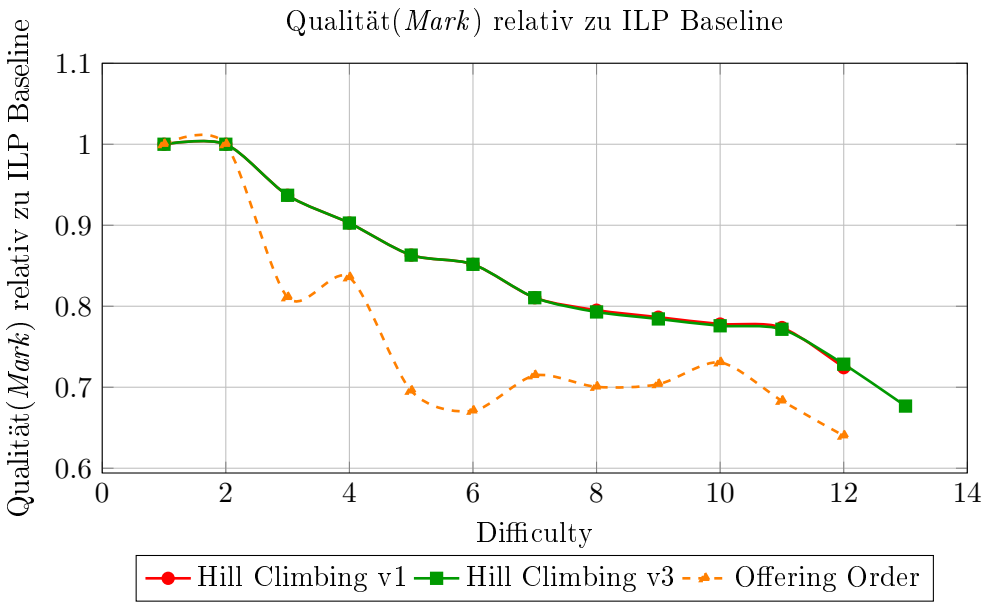
   - Median ($t_{\text{median}}, M_{\text{median}}$)
   - Mittelwert ($t_{\text{mean}}, M_{\text{mean}}$)
   - Minimalwert ($t_{\text{min}}, M_{\text{min}}$)
   - Maximalwert ($t_{\text{max}}, M_{\text{max}}$)
   - Standardabweichung ($t_{\text{stdev}}, M_{\text{stdev}}$)

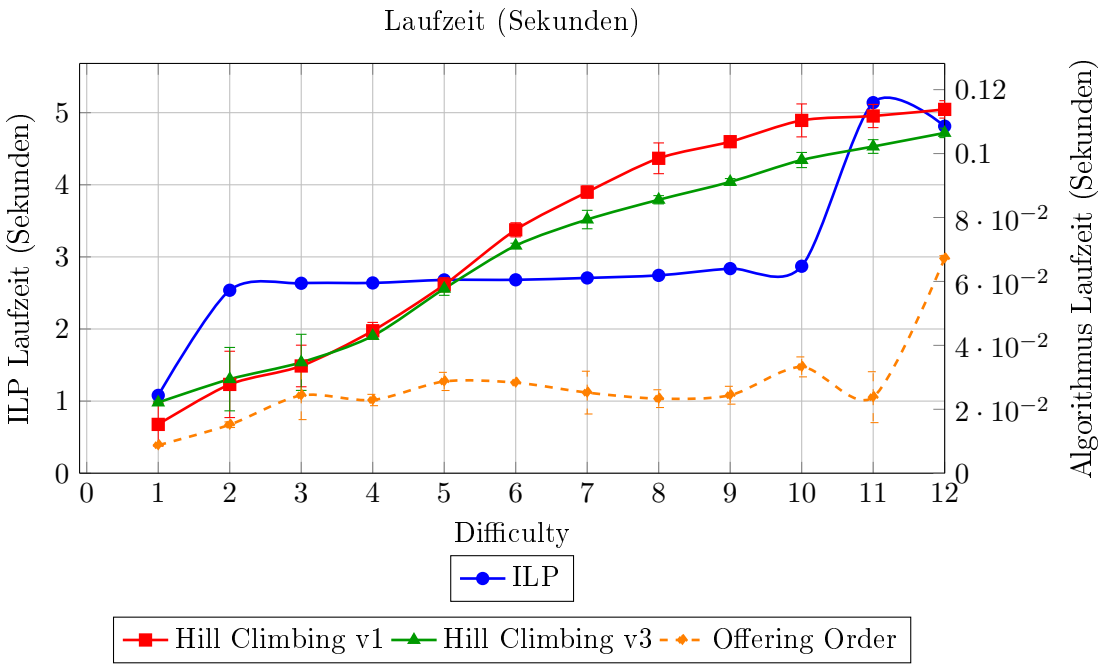
Um die Algorithmen vergleichbar zu halten, wurde der gemessene Zeit- und Speicherbedarf auf die Ausführung des Algorithmus selbst begrenzt. Das Preprocessing — insbesondere die Vorabberechnung der *Mark* für jedes Offering, die einmalig vor der Hauptschleife durchgeführt wird — ist nicht in den Messwerten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

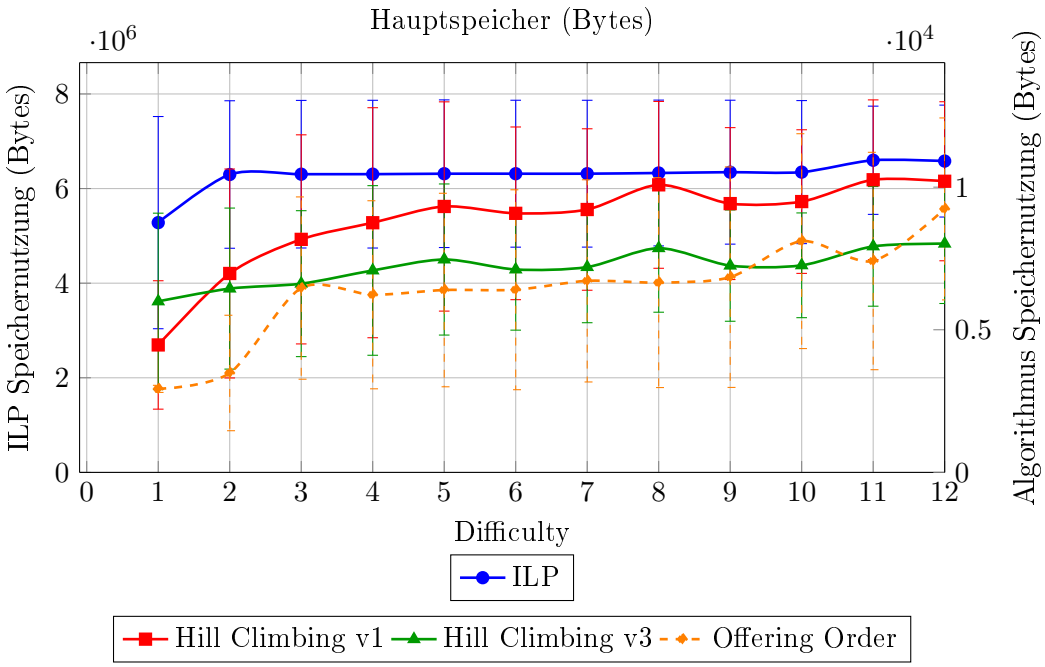
4.4.4 Ergebnisse

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,
    "COURSE_COUNT_CONSTRAINT": null
}
```

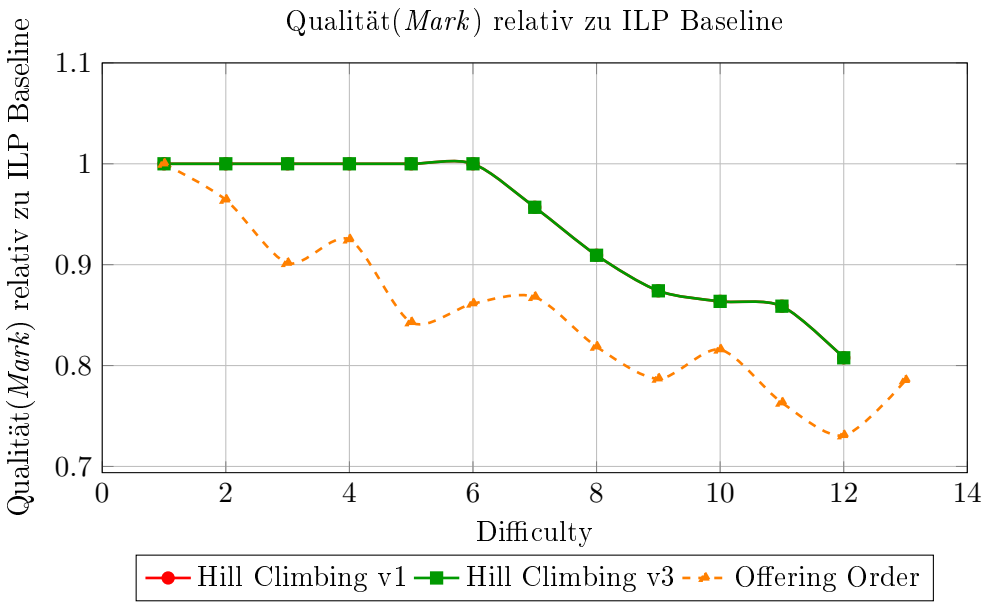


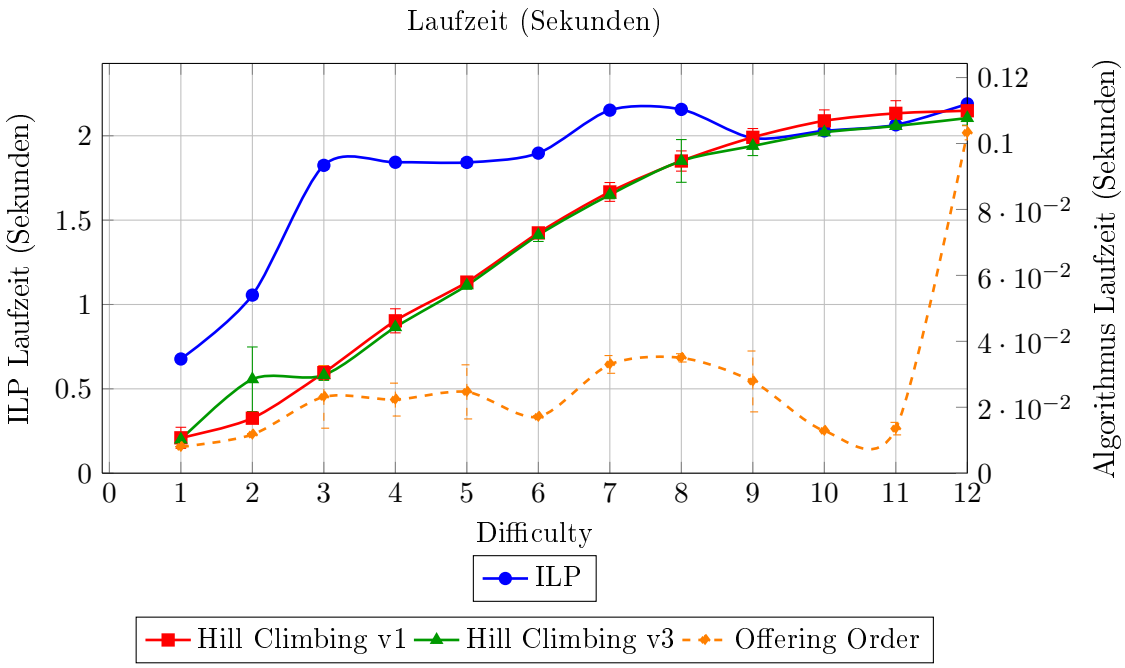

Laufzeit (Sekunden)
ILP Laufzeit (Sekunden)
Algorithmus Laufzeit (Sekunden)
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

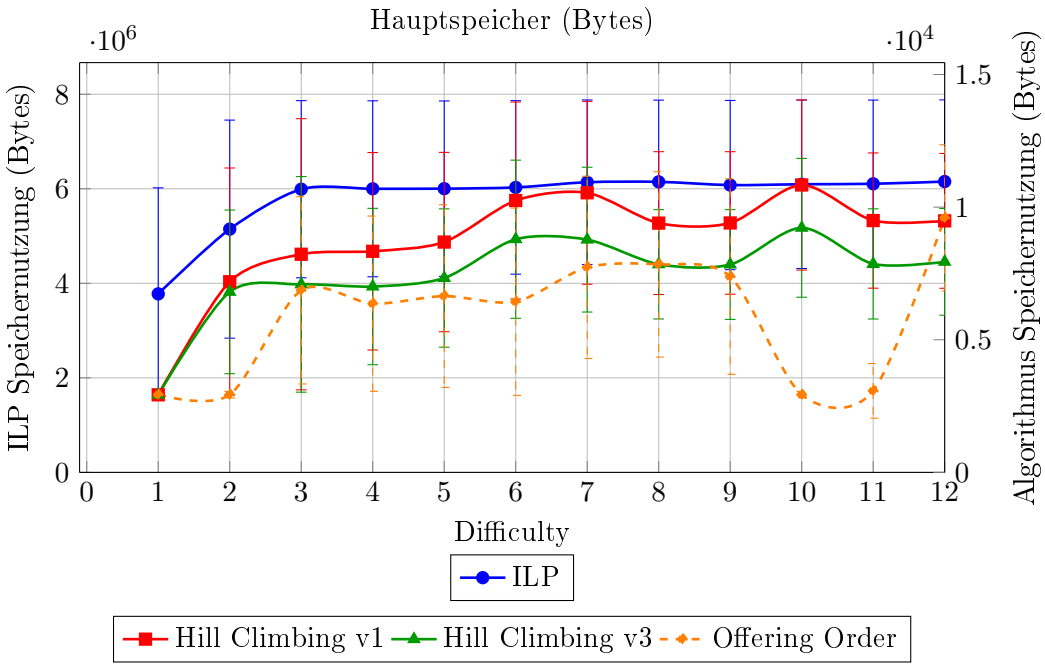

Hauptspeicher (Bytes)
ILP Speichernutzung (Bytes)
Algorithmus Speichernutzung (Bytes)
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

Szenario 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,
    "COURSE_COUNT_CONSTRAINT": null
}
```

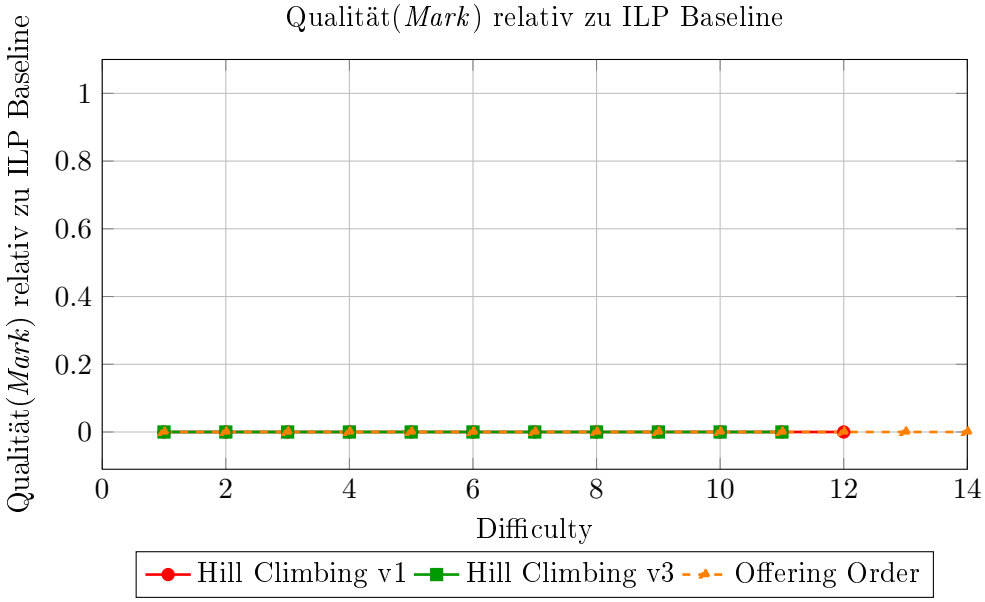


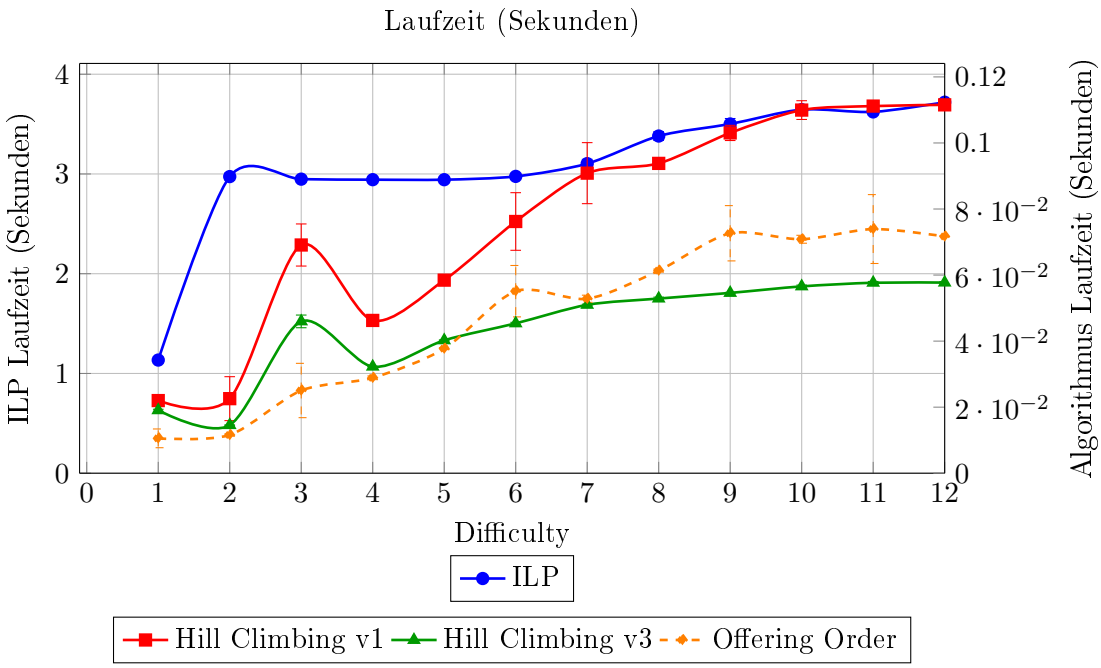

Laufzeit (Sekunden)
ILP Laufzeit (Sekunden)
Algorithmus Laufzeit (Sekunden)
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

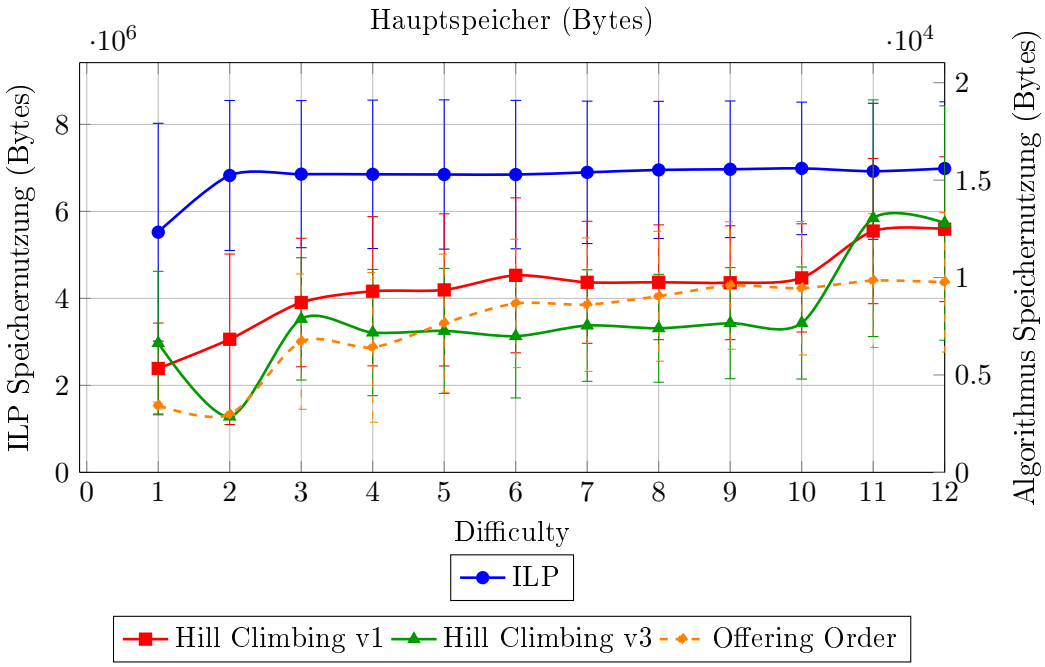

Hauptspeicher (Bytes)
ILP Speichernutzung (Bytes)
Algorithmus Speichernutzung (Bytes)
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

Szenario 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,
    "COURSE_COUNT_CONSTRAINT": null
}
```

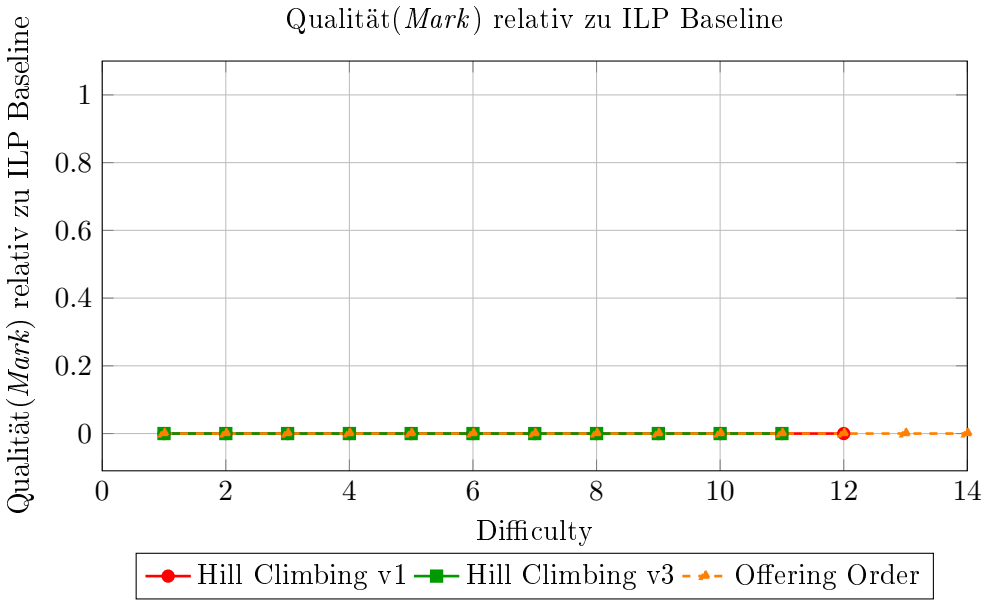



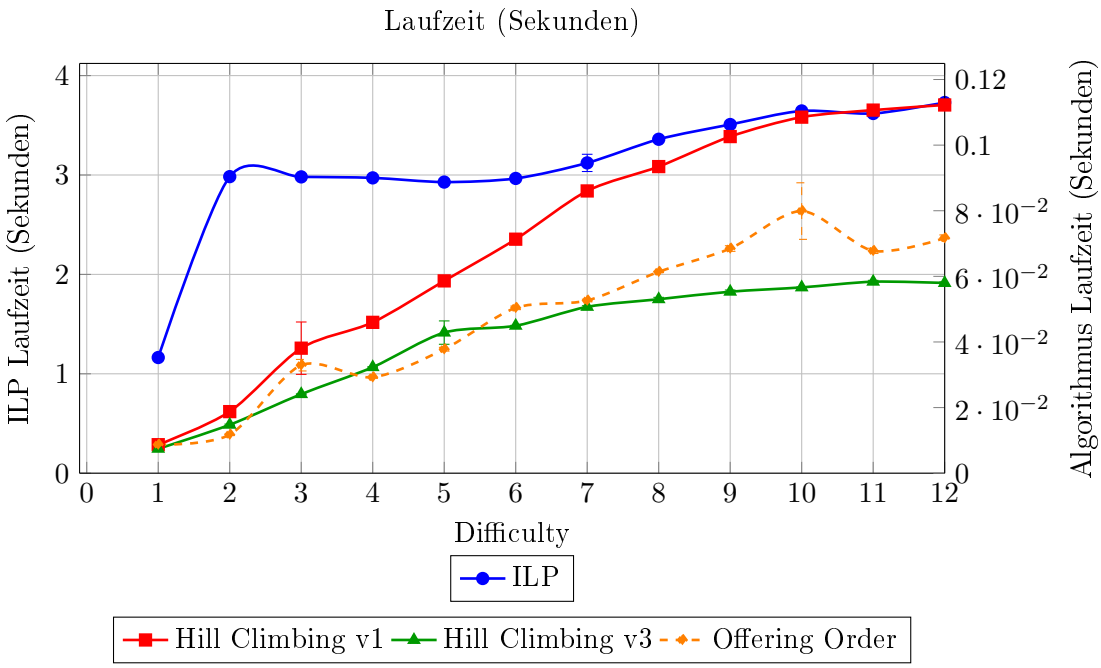

Laufzeit (Sekunden)
ILP Laufzeit (Sekunden)
Algorithmus Laufzeit (Sekunden)
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

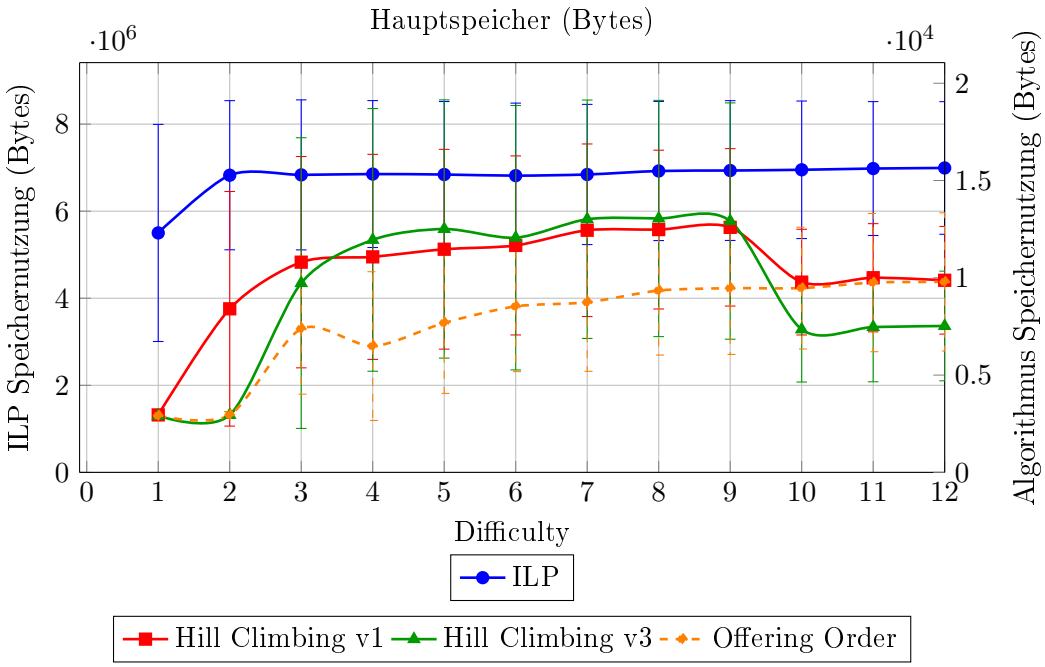

Hauptspeicher (Bytes)
ILP Speichernutzung (Bytes)
Algorithmus Speichernutzung (Bytes)
Difficulty
ILP
Hill Climbing v1
Hill Climbing v3
Offering Order

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,
    "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 [1]

4.4.5 Diskussion

Todo: Diskussion, Fazit, Potenzielle Weiterentwicklung und Anwendungsfälle

5 Ergebnisse

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

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