



# Practically solving the VRPTW

Bachelor Thesis

Verfasser:

Pascal Lüscher

Janik Meyer

Studiengang:

Informatik

Auftraggeber:

Fabian Affolter im Namen der Chlausgesellschaft Niederwil-Nesselnbach

Betreuer:

Simon Felix

Fabian Affolter

März, 2019

## Zusammenfassung

Diese Projektarbeit befasst sich mit Chläusen, die Familien besuchen. Familien haben gewisse Zeiten, an denen sie nicht besucht werden können und gewisse Wunschzeiten, an denen sie vorzugsweise besucht werden wollen. Es kommen mehrere Chläuse zum Einsatz, die an mehreren Tagen unterwegs sind. Jeder Chlaus hat pro Tag nur eine beschränkte Kapazität. Die Frage ist, wann und in welcher Reihenfolge die Familien von den Chläusen besucht werden sollen, sodass die Kosten minimal werden. Das Problem wird in der Fachliteratur “Vehicle Routing Problem with Time Windows” genannt.

In einem Vorgängerprojekt wurde bereits ein Algorithmus basierend auf ganzzahlig linearer Programmierung entwickelt, der das oben beschriebene Problem lösen kann. Im Rahmen dieser Arbeit werden vier neue Lösungsansätze vorgestellt, die dieses Optimierungsproblem für unseren Kunden lösen sollen. Diese Ansätze werden in einer Evaluation miteinander verglichen, dabei werden verschiedene Aspekte bewertet.

Aus der Evaluation geht hervor, dass gemessen an der Zielfunktion, die neu entwickelten Ansätze bessere Routenpläne erzeugen, als die von Hand erstellten. Die Erkenntnisse aus dieser Arbeit ermöglichen es der Chlausgesellschaft Kosten zu sparen. Es können der Arbeitsaufwand für die Planung und die Kosten für die Besuche reduziert werden. Es wird gezeigt, dass mittels optimierter Routen in den Jahren 2017 und 2018 insgesamt Einsparungen von 230 CHF hätten erzielt werden können, was 15.6% der Gesamtkosten der Besuche entspricht.

# Inhaltsverzeichnis

Zusammenfassung .....	1
1 Einleitung .....	4
2 Problemdefinition .....	6
3 Lösungsansätze .....	10
3.1 IP5 - Integer Linear Programming .....	11
3.2 ILP2 – Vollständiges Modell .....	13
3.3 Genetischer Algorithmus .....	23
3.4 LocalSolver .....	34
3.5 Google OR-Tools Routing .....	41
4 Evaluation .....	46
4.1 Messmethodik .....	46
4.2 Übersicht .....	47
4.3 Laufzeit .....	48
4.4 Skalierbarkeit .....	49
4.5 Lösungsqualität .....	50
4.6 Vergleich manuell geplante Route .....	52
4.7 Lösungsstabilität .....	54
4.8 Auswirkung Wunschzeit & Nichtverfügbarkeit .....	55
4.9 Zusätzliche Chläuse .....	59
4.10 Lizenzkosten .....	60
5 Schlussfolgerung / Empfehlung .....	61
6 Verzeichnisse .....	63
6.1 Abbildungsverzeichnis .....	63
6.2 Tabellenverzeichnis .....	64
6.3 Literaturverzeichnis .....	64
7 Ehrlichkeitserklärung .....	67
8 Anhang .....	68
8.1 Source Code .....	68
8.2 Resultate der Evaluationsdurchläufe .....	68
8.3 IP5 Gurobi Logfile Clustering TIU .....	72
8.4 ILP2 TI Logfile Beweis Optimum .....	73

8.5 Test GA Parallel ..... 76

# 1 Einleitung

Die Jungwacht Niederwil sorgt jedes Jahr dafür, dass der Samichlaus in Niederwil AG und Nesselbach AG zahlreiche Familien zu Hause besucht. Die Familien können sich entweder auf der Webseite oder mittels schriftlichem Formular anmelden. Bei der Anmeldung können Terminwünsche geäußert werden. Nach Anmeldeschluss treffen sich die Organisatoren der Chlausgesellschaft und planen einen Abend lang den Ablauf. Dabei werden alle Anmeldungen nach bestem Wissen den Chläusen zugeteilt. Die Chläuse erstellen dann eine individuelle Routenplanung. Diese Aufteilung und Routenplanung werden aber zunehmend komplizierter. Gründe dafür sind die steigenden Anmeldungen und die zunehmend fehlende Flexibilität der Familien. Ein weiteres Problem ist, dass die Routenplanung den Chläusen viel Wissen und Routine abverlangt.

In einem Vorgängerprojekt wurde bereits eine Applikation entwickelt, die solche Routen für die Chläuse erstellen kann. Bei diesem Vorgängerprojekt kam ein Algorithmus zum Einsatz, der auf ganzzahliger linearer Programmierung basiert. Im Rahmen der vorliegenden Bachelorarbeit sollen weitere Ansätze entwickelt werden. Diese Ansätze sollen dann miteinander verglichen werden, um am Schluss mindestens einen Algorithmus in die bestehende Webapplikation zu integrieren. Das Ziel ist, dass am Ende des Projekts das Optimierungsproblem für den Kunden gelöst wird.

Das beschriebene Problem ist in der Fachliteratur unter dem Namen Vehicle Routing Problem with Time Windows (VRPTW) bekannt. Das Vehicle Routing Problem with Time Windows ist eine Erweiterung des Vehicle Routing Problem (VRP). Einfach gesagt, befasst sich das VRP mit einer Flotte von Fahrzeugen, welche Güter aus einem zentralen Depot zu Kunden liefern soll. Bei der Erweiterung mit Time Windows besteht die Einschränkung darin, dass Kunden nur in einem bestimmten Zeitfenster, beispielsweise Öffnungszeiten, besucht werden dürfen. Bei unserer Problemstellung werden die Kunden durch Familien repräsentiert, welche von einem Chlaus besucht werden. Dementsprechend wird die Fahrzeugflotte durch die Chläuse repräsentiert. Das VRP sowie das VRPTW gehören zur Klasse der NP-schweren Probleme [1].

Erstmals erwähnt wurde das Vehicle Routing Problem im Jahre 1959. Damals wurde das Problem von Dantzig und Ramser als "The truck dispatching problem" beschrieben. Sie haben ein lineares Optimierungsverfahren entwickelt, welches auch von Hand durchgeführt werden kann. Dieser Ansatz ist heute unter dem Namen "savings algorithm" bekannt. Dieser Algorithmus gehört zur Kategorie der Greedy-Algorithmen [2].

Christophides und Elion haben ein Verfahren entwickelt, welches aus dem VRP ein Traveling Salesman Problem (TSP) macht. Der von ihnen präsentierte Ansatz löst das VRP nicht optimal, ist aber eine gute Annäherung. Der Ansatz von Christophides und Elion wandelt das VRP in ein TSP um, indem das Depot ein Mal pro Fahrzeug als Kunde eingetragen wird. Die Distanz zwischen den Depots wird auf unendlich gesetzt [3].

Viele betriebliche Entscheidungsprobleme können als exakte Optimierungsprobleme formuliert werden. Dies trifft auch auf unser VRPTW zu. In der Praxis kommen für betriebliche Entscheidungsprobleme jedoch häufig heuristische Verfahren zur Anwendung. Der Grund dafür ist, dass exakte Formulierungen häufig zu komplex sind, um innert nützlicher Frist gelöst zu werden. Bedingt durch die heuristischen Verfahren, ist es oft nicht möglich, das Problem exakt zu lösen. Dennoch wurde die hohe Leistungsfähigkeit guter Heuristiken beim Lösen des VRPTW erkannt. Als Beispiele werden Simulated Annealing, Tabu Search und genetische Algorithmen genannt [1].

Im kommerziellen Bereich gibt es verschiedene Anbieter, die Werkzeuge anbieten, um unterschiedlichste Routen- und Zeitplanungsprobleme zu lösen. Dabei kommen diverse Technologien zum Einsatz, die oftmals miteinander kombiniert werden. Bei Quintiq kommt beispielsweise ein Hybridverfahren zum Einsatz. Dabei wird unter anderem Neighborhood-Search, Constraint Programming und Mixed Integer Programming verwendet. IBM bietet mit dem CPLEX CP Optimizer ein Tool an, das auf Constraint Programming basiert. Der VSR Optimizer, welcher im SAP Modul Transportation Management enthalten ist, benutzt einen erweiterten "Local Search"-Ansatz [4] [5] [6].

Das VRPTW ist heute noch Gegenstand der Forschung. Einer der am häufigsten referenzierten Benchmarks ist der Solomon Benchmark [7]. Dieser Benchmark besteht aus Probleminstanzen und den besten bekannten Lösungen. Bei diesem Benchmark wird ein hierarchisches Ziel verfolgt. Erstens soll die Anzahl benötigter Fahrzeuge minimiert werden. Zweitens soll die totale Wegdistanz minimiert werden. Die Probleminstanzen umfassen 25, 50 oder 100 Kunden. Eine Erweiterung stellt der Gehring & Homberger Benchmark dar, der Problemgrößen von 200 - 1000 Kunden behandelt. An der Probleminstanz mit 1000 Kunden ist erkennbar, dass aktuell noch am VRPTW geforscht wird, denn die Mehrheit der Lösungen stammen aus den Jahren 2018 und 2019 [8].

## 2 Problemdefinition

Das Optimierungsproblem, welches die Chlausgesellschaft lösen will, unterscheidet sich in mehreren Punkten vom klassischen VRPTW.

Allem voran sind die Zeitfenster komplexer. Das ursprüngliche VRPTW kennt nur ein Zeitfenster pro Kunde, das zwingend eingehalten werden muss. Das heisst, wenn das Fahrzeug vor diesem Zeitpunkt beim Kunden eintrifft, muss bis zum Start des Zeitfensters gewartet werden. Trifft das Fahrzeug zu spät ein, so kann der Besuch nicht mehr durchgeführt werden. In unserem Fall darf der Kunde vor und nach dem Zeitfenster besucht werden, es entstehen dabei jedoch Mehrkosten. Generell entsprechen die Zeitfenster des VRPTW den Nichtverfügbarkeiten.

Ein weiterer Unterschied besteht darin, dass die Kunden in unserer Problemstellung Zeitfenster angeben können, in denen sie besucht werden möchten. Wenn Besuche in diesem gewünschten Zeitfenster stattfinden, wird ein Bonus zur Zielfunktion hinzugefügt. Eine zusätzliche Besonderheit ist, dass es mehrere Zeitfenster geben darf, sowohl bei der Nichtverfügbarkeit wie auch bei der Wunschzeit.

Im Gegensatz zum verbreiteten VRPTW dürfen die Chläuse auch Pausen machen. Diese Pausen müssen an einer bestimmten Adresse gemacht werden und sind fest einem Chlaus zugeordnet. Eine Pause muss vom Chlaus an jedem Tag eingelegt werden, sofern der Chlaus an dem Tag mindestens eine Familie besucht. Die Pausen haben ebenfalls eine Wunschzeit, in der der Chlaus diese machen will. Auch hier gibt es einen Bonus für das Einhalten dieses Zeitfensters.

Ein weiterer Punkt ist, dass auch zusätzliche Chläuse verwendet werden dürfen. Ein zusätzlicher Chlaus ist ein Chlaus, der in der originalen Menge der Chläuse nicht vorhanden ist und demnach künstlich hinzugefügt wird. Wie unten in der Zielfunktion ersichtlich ist, kosten diese zusätzlichen Chläuse mehr als die vorgesehenen Chläuse. Jedoch haben zusätzliche Chläuse keine Pausen.

Letztlich unterscheidet sich auch die Zielfunktion. In der Literatur [7] wird oftmals die Anzahl Fahrzeuge und die Wegzeit minimiert. In unserer Zielfunktion wird zwar auch die Wegzeit minimiert, jedoch sollen die Routen auch ähnlich lang sein. Dies wird erreicht, indem der längste Tag proportional zur Dauer bestraft wird. Die Zielfunktion besteht aus der Summe der Komponenten, die in Tabelle 1 abgebildet sind. Die Konstanten repräsentieren die tatsächlichen Kosten.

Gewicht in CHF	Was
560	Anzahl nicht besuchter Familien / Pausen
400	Anzahl zusätzlich benötigter Chläuse
40	Arbeitszeit der zusätzlichen Chläuse [h]
120	Besuchszeit ausserhalb der verfügbaren Zeit [h]
120	Wegzeit ausserhalb der verfügbaren Zeit [h]
-20	Besuchszeit innerhalb der gewünschten Zeit [h]
40	Arbeitszeit [h]
30	Dauer des längsten Arbeitstages [h]

Tabelle 1: Zielfunktion der Problemstellung

Um die Flexibilität der Lösungsansätze zu gewährleisten, werden mehrere Input-Datensätze benötigt. Es liegen nur zwei reale Beispiele vor, weshalb weitere künstliche Probleminstanzen generiert werden. Diese Probleminstanzen werden in Anlehnung an die reale Problemstellung unseres Kunden erzeugt. Jeder Datensatz wird nach demselben Schema generiert. Die Koordinaten der Besuche werden in einem Feld der Grösse 4000 auf 4000 verteilt. Dies stellt ein Gebiet von 16 Quadratkilometern dar. 45% der Punkte werden aus der Normalverteilung generiert, die den Mittelwert  $(\frac{1}{3} 4000, \frac{1}{3} 4000)$  und die Standardabweichung 400 hat. Weitere 45% der Punkte werden aus der Normalverteilung generiert, die den Mittelwert  $(\frac{2}{3} 4000, \frac{2}{3} 4000)$  und die Standardabweichung 400 hat. Die letzten 10% werden Anhand einer Gleichverteilung zufällig im Feld verteilt. Ein Beispiel mit 1000 Besuchen ist in der Abbildung 1 ersichtlich.



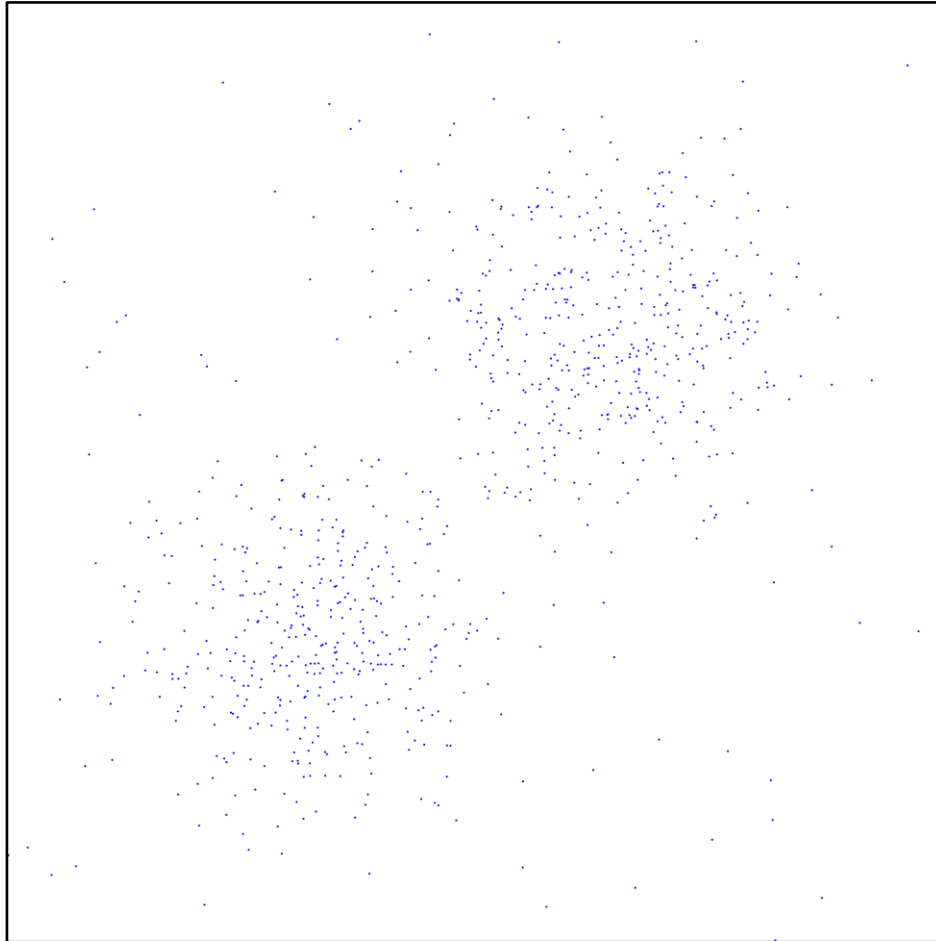


Abbildung 1: Datensatz mit 1000 Besuchen

Die Wegzeiten entsprechen den euklidischen Distanzen zwischen zwei Koordinaten.

Besuche werden in Familienbesuche und Pausen unterteilt. Wobei sich die Anzahl der Besuche aus der Summe der Anzahl Familienbesuchen und der Anzahl Pausen multipliziert mit der Anzahl Tage zusammensetzt.

Die Wunschzeiten der Familienbesuche werden innerhalb eines einzigen Arbeitstages zufällig eingetragen. Die Dauer der Wunschzeit rechnet sich zusammen aus der Besuchsdauer plus einem zufälligen Anteil des Arbeitstages. Dasselbe gilt für die Nichtverfügbarkeit, wobei sichergestellt wird, dass Wunschzeiten und Nichtverfügbarkeiten nicht am selben Tag eingetragen sind.

Die Wunschzeiten der Pausen werden wie bei den Familien erstellt, mit dem Unterschied, dass an jedem Arbeitstag eine Wunschzeit generiert wird.

Die Datensätze sind anhand ihrer Merkmale benannt. Der erste Teil des Namens gibt an, wie viele Besuche (V für Visits) der Datensatz enthält. Darauf folgen Buchstaben, die angeben, ob der Datensatz Pausen (B für Breaks), Wunschzeiten (D für Desired) und Nichtverfügbarkeiten (U für Unavailable) enthält.

Datensatz	Familien- besuche	Pausen	Chläuse	Tage	Wunsch- zeiten	Nichtverfüg- barkeiten
10V	10	0	1	2	0	0
10VD	10	0	1	2	10	0
10VU	10	0	1	2	0	10
20V	20	0	2	2	0	0
20VBD	16	2	2	2	16	0
20VU	20	0	2	2	0	20
34VDU	34	0	3	2	23	15
34VBDU	32	1	3	2	31	24
50VBDU	40	5	5	2	30	22
100VBDU	80	10	10	2	70	40
200VBDU	160	20	20	2	150	80
1000VBDU	800	100	100	2	600	300

Tabelle 2: Kennzahlen der Datensätze

In Tabelle 2 sind die Spezifikationen aufgelistet, nach denen die Datensätze generiert werden. Die ersten drei Datensätze sind rein synthetische Datensätze, welche die verschiedenen Auswirkungen von Wunschzeiten und Nichtverfügbarkeiten ausweisen sollen. Die nächsten drei sind eine hochskalierte Version der ersten drei, wobei beim Datensatz 20VBD bereits Pausen eingebaut wurden. Als Datensatz 34VDU und 34VBDU werden die realen Daten der Jahre 2017 und 2018 verwendet. Die Datensätze mit mehr als 34 Besuchen sind hochskalierte Abbildungen der realen Daten.

Der Datensatz 1000VBDU stellt die Obergrenze mit 1000 Besuchen dar, um Limits zu testen. Diese Obergrenze ist so gewählt, weil die grösste Problem Instanz des Gehring & Homberg Benchmarks [8] ebenfalls 1000 Kunden beinhaltet. Allgemein sind die Datensätze entsprechend der Anzahl Besuche sortiert. Es werden bei jedem Datensatz zwei Arbeitstage festgelegt. Dies resultiert aus der Situation der Chlausgesellschaft, die ebenfalls zwei Arbeitstage aufwendet, um alle Familien zu besuchen.

## 3 Lösungsansätze

Die in Kapitel 2 definierte Problemstellung kann auf verschiedene Arten gelöst werden. Im Rahmen dieses Projekts werden fünf mögliche Ansätze betrachtet, die in diesem Kapitel beschrieben werden. Zuerst wird erläutert warum diese fünf Ansätze gewählt werden.

Der IP5-Ansatz stammt aus einem Vorgängerprojekt, das sich mit derselben Problemstellung befasst hat. Das IP5 ist eine erste Implementierung und damit ein Referenzwert.

Der zweite Ansatz, mit der Bezeichnung ILP2, basiert auf ganzzahliger linearer Programmierung und ist eine vollständige Formulierung des Problems.

Der genetische Algorithmus ist ein weiterer Ansatz, der evaluiert wird. Gemäss [9] eignen sich genetische Algorithmen gut für kombinatorische Probleme, insbesondere wenn der Suchraum viele lokale Extremalstellen aufweist. Diese beiden Eigenschaften besitzt das VRPTW.

LocalSolver ist ein Solver, welcher in einigen der schwersten MIPLIB Probleminstanzen innerhalb vorgegebener Zeit zur besten Lösung gekommen ist [10]. Dabei war er leistungsfähiger als die besten kommerziellen MILP-Solver wie Gurobi oder IBM Ilog Cplex. Der Ansatz basiert auf einem spezialisierten Local Search Algorithmus. Wie in Kapitel 1 beschrieben, verwenden viele kommerzielle Lösungen Local Search Algorithmen.

Google Routing ist der letzte Ansatz, der auf den Google OR-Tools basiert. Dabei kommt die Routing Bibliothek zum Einsatz, die unter anderem mit Constraint-Programmierung implementiert ist. Dieser letzte Ansatz soll die Vielfalt der verwendeten Technologien vergrössern. Es ist zu bemerken, dass mehrere kommerzielle Ansätze Constraint-Programmierung benutzen.

## 3.1 IP5 - Integer Linear Programming

In dem Vorgängerprojekt wurde ein Ansatz mit ganzzahliger linearer Programmierung erstellt. Dieser Ansatz besteht im Wesentlichen aus zwei Phasen, dem Clustering und dem Scheduling. Im Clustering wird das VRP mit einigen Zusätzen gelöst. Die Verteilung der Besuche auf die Chläuse geschieht mit dem Ziel, die Wegzeiten möglichst gering zu halten. Zudem werden Besuche nicht an einem Tag stattfinden, an dem sie ein Nichtverfügbarkeits-Zeitfenster haben. Die Pausen werden fest den Routen zugewiesen. Die Fairnessbedingung, die längste Route zu bestrafen, wird ebenfalls optimiert.

Das Resultat der ersten Phasen ist eine Routenplanung. Die Routen sagen aus, welcher Chlaus an welchem Tag welche Familien besucht und wann die Pausen gemacht werden. Dazu kommt, dass die Familien und Pausen einer bestimmten Reihenfolge unterliegen, sodass die Wege und der längste Tag möglichst kurz ausfallen. Details zur Formulierung können aus dem IP5 Bericht entnommen werden [11].

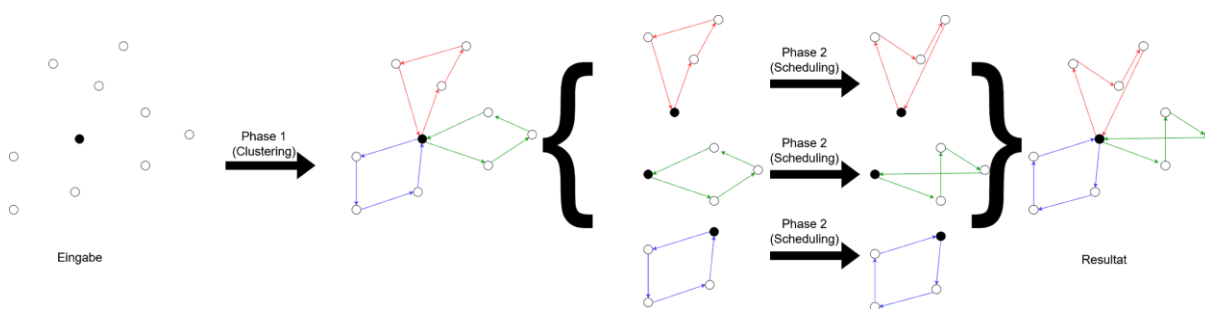


Abbildung 2: Ablauf IP5

Pro berechneter Route wird schliesslich der zweite Teil, das Scheduling, angewendet. Dieses Verfahren ist in Abbildung 2 graphisch dargestellt. Das Scheduling erhält das Resultat aus dem Clustering und versucht, die Besuche möglichst optimal anzuordnen, sodass die ursprüngliche Zielfunktion optimiert wird. Dabei muss nur noch die Arbeitszeit eines Chlaus und die Wunschzeit der Besuche beachtet werden, da Nichtverfügbarkeiten bereits durch die erste Phase ausgeschlossen werden.

In der zweiten Phase wird die Zeit gerastert dargestellt. Vereinfacht gesagt, basiert die Formulierung auf einer Tabelle von booleschen Variablen, wobei in einer Achse die Familien und in der anderen die Zeit eingezeichnet sind. Ist eine Variable auf 1 gesetzt, so bedeutet dies, dass die Familie zu dem Zeitpunkt besucht wird. In der Tabelle 3 ist ein Beispiel dieser Tabelle aufgeführt. Zur Vereinfachung werden Nullwerte nicht dargestellt. Im dargestellten Beispiel wird die Familie 1 von 17:00 bis 17:15 Uhr besucht, danach begibt sich der Chlaus zur Familie 2, bei der er von 17:20 bis 17:30 Uhr verweilt.

	17:00	17:05	17:10	17:15	17:20	17:25	17:30	17:35	17:40
Familie 1	1	1	1						
Familie 2					1	1			
Familie 3									1

Tabelle 3: Scheduling Planungsdarstellung

Aus dem Rastern der Zeit resultiert ein Parameter, mit dem die maximale Genauigkeit eingestellt werden kann.

Durch das Zusammenfassen der einzelnen Resultate der zweiten Phase ergibt sich die gesamte Routenplanung. Sollten die Routen der zweiten Phase nicht innerhalb des Zeitlimits berechenbar sein, werden die Routen der ersten Phase als Schlussresultat zurückgegeben.

Das beschriebene Modell wurde ursprünglich für SCIP implementiert. SCIP ist ein Programm zur Lösung ganzzahlig lineare Probleme. Im Rahmen dieses Projekts soll aber auch getestet werden, wie gut das Modell mit Gurobi gelöst werden kann. Gurobi ist eine Software für mathematische Optimierung, die unter anderem ganzzahlige lineare Probleme lösen kann. In den folgenden Kapiteln wird das IP5 Modell mit SCIP als "IP5 SCIP" bezeichnet. Die in diesem Projekt entwickelte Portierung des IP5 Modells auf Gurobi wird als "IP5 Gurobi" bezeichnet.

## 3.2 ILP2 – Vollständiges Modell

Aufgrund der Rückmeldungen betreffend dem IP5 wird ein weiteres Modell formuliert, welches mit ganzzahliger linearer Optimierung gelöst werden kann. Dieser Lösungsansatz bindet die Gurobi Bibliothek direkt ein. Dadurch können Gurobi-spezifische Funktionen wie general constraints (für mehr Informationen siehe [12]) verwendet werden.

Wie das IP5 hat auch dieses Modell zwei Phasen. In der ersten Phase wird das VRP gelöst. Die Lösung dieser Phase wird als Initiaillösung (sog. MIP Start) für die zweite Phase verwendet. Im Gegensatz zum IP5 ist die zweite Phase ein komplett formuliertes Modell. Dies bringt den Vorteil, dass die Problemstellung theoretisch bis zum Optimum optimiert werden kann.

Durch das Einführen der ersten Phase muss die zur Verfügung stehende Zeit aufgeteilt werden. Das richtige Zeitverhältnis wird dabei mittels einer Rastersuche berechnet. Für die Rastersuche werden 15 Datensätze mit 20 Besuchen, 2 Pausen, 2 Chläusen und 2 Tagen verwendet. Dies entspricht einer skalierten Version der Evaluations-Datensätze aus Kapitel 2. Die Resultate werden pro Durchlauf aufsummiert. Die Rastersuche wird dreimal ausgeführt. Das Resultat ist der Mittelwert der drei Durchläufe.

Zeitlimit Phase 1										
0	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
13536	12689	12698	12693	12693	12693	12692	12692	12696	12696	12977

Tabelle 4: ILP2 Rastersuche Resultat 1

In Tabelle 4 ist zu erkennen, dass die vielversprechendste Zeiteinteilung beim Verhältnis 10% für die erste Phase und 90% für die zweite Phase liegt. Dieser Wert liegt sehr nahe am schlechtesten gefundenen Wert. Es wird das Resultat für zwei weitere Werte, 5% und 15%, berechnet um zu zeigen, dass der Wert 10% geeignet ist.

Zeitlimit Phase 1				
0%	5%	10%	15%	20%
13536	12689	12689	12687	12698

Tabelle 5: ILP2 Rastersuche Resultat 2

Aus Tabelle 5 wird ersichtlich, dass 15% für die erste Phase noch besser wären. Der Unterschied ist jedoch sehr klein. Zudem wird deutlich, dass 20% schlechter ist als 10%. Der schlechteste Wert bei 0% lässt sich dadurch erklären, dass die Startlösung bei der zweiten Phase durch triviales Aufteilen der Besuche initialisiert wird. Bereits ein leicht optimierter Start hilft der zweiten Phase, wesentlich bessere Resultate zu berechnen.

In der ersten Phase wird das Vehicle Routing Problem als ganzzahliges lineares Programm formuliert. Die verwendete Formulierung basiert auf einer Traveling Salesman Formulierung von Gurobi [13].

In der folgenden Formulierung wird die Variable  $V$  als Menge aller Besuche exklusiv Startpunkt verwendet. Die Anzahl der Besuche wird mit  $n_v = |V|$  bezeichnet. Analog zu den Besuchen ist die Menge aller Chläuse als  $S$  und die Anzahl aller Chläuse als  $n_s$  bezeichnet. Die Anzahl der Chläuse  $n_s$  entspricht der Anzahl Tage multipliziert mit der Anzahl Chläuse. Die Pause  $b_s$  für Chlaus  $s$  ist in der Menge  $B_s \subseteq V$  enthalten. Um die Wege zwischen den Besuchen abzubilden, wird die Variable  $x_{s,i,j} \in \{0,1\} \forall s \in S, \forall i, j \in V$  verwendet. Sie nimmt den Wert 1 ein, falls der Chlaus  $s$  den Weg von Besuch  $i$  nach  $j$  benutzt. Ob ein Chlaus eine Familie besucht, wird mit der Variable  $v_{s,i} \in \{0,1\} \forall s \in S, \forall i \in V$  angegeben. Sie nimmt den Wert 1 an, falls der Chlaus  $s$  die Familie  $i$  besucht. Zudem sind die Wege von und zum Startpunkt als  $w_{s,0,i} \in \{0,1\} \forall s \in S, \forall i \in V$  respektive  $w_{s,i,0} \in \{0,1\} \forall s \in S, \forall i \in V$  definiert. Analog dazu ist  $v_{s,0} \in \{0,1\} \forall s \in S$  die Variable, ob ein Chlaus den Startpunkt besucht. In der Matrix *distance* sind die Distanzen zwischen den Besuchen in Sekunden enthalten. So definiert *distance* $_{i,j}$  die Distanz zwischen Besuch  $i$  und  $j$ . In der Matrix *duration* ist die Besuchsdauer in Sekunden enthalten. Demnach ist *duration* $_i$ , die Besuchsdauer für Besuch  $i$ . Die Variable *longestRoute*  $\in \mathbb{R}$  wird für die längste Route verwendet.

Wenn aus dem Kontext heraus klar ist, was gemeint ist, wird die Notation  $\forall s$  stellvertretend für  $\forall s \in S$  sowie  $\forall i$  für  $\forall i \in V$  verwendet.

minimize

$$4 \left( \sum_{s=1}^{n_s} \sum_{i=1}^{n_v} \sum_{j=1}^{n_v} w_{s,i,j} \text{distance}_{i,j} + \sum_{s=1}^{n_s} \sum_{i=1}^{n_v} v_{s,i} \text{duration}_i \right) + 3 \text{longestRoute} \quad (1)$$

s.t.

$$\text{longestRoute} \geq \sum_{i=1}^{n_v} \sum_{j=1}^{n_v} w_{s,i,j} \text{distance}_{i,j} + \sum_{i=1}^{n_v} v_{s,i} \text{duration}_i \quad \forall s \quad (2)$$

$$w_{s,i,i} = 0 \quad \forall s, \forall i \quad (3)$$

$$\sum_{s=1}^{n_s} v_{s,i} = 1 \quad \forall i \in V \setminus B \quad (4)$$

$$\sum_{s=1}^{n_s} \sum_{j=1}^{n_v} w_{s,i,j} = \sum_{s=1}^{n_s} \sum_{j=1}^{n_v} w_{s,j,i} = 1 \quad \forall i \quad (5)$$

$$\sum_{j=1}^{n_v} w_{s,i,j} = \sum_{j=1}^{n_v} w_{s,j,i} = v_{s,i} \quad \forall s, \forall i \quad (6)$$

$$\sum_{j=1}^{n_v} w_{s,0,j} = \sum_{j=1}^{n_v} w_{s,j,0} \quad \forall s \quad (7)$$

$$\sum_{j=1}^{n_v} w_{s,0,j} \geq v_{s,i} \quad \forall s, \forall i \quad (8)$$

$$\sum_{j=1}^{n_v} w_{s,0,j} \leq \sum_{i=1}^{n_v} v_{s,i} \quad \forall s \quad (9)$$

$$\sum_{i=0}^{n_v} \sum_{j=0}^{n_v} w_{s,i,j} = \sum_{i=0}^{n_v} v_{s,i} \quad \forall s \quad (10)$$

$$v_{s,b_s} \leq \sum_{j=0}^{n_v} v_{s,j} \quad \forall s \quad (11)$$

$$v_{s,b_s} \geq v_{s,i} \quad \forall s, \forall i \quad (12)$$

$$v_{s_2, b_{s_1}} = 0 \quad \forall s_1, s_2 \in S, s_1 \neq s_2 \quad (13)$$

$$\sum_{i,j \in T} w_{s,i,j} \leq |T| - 1 \quad \forall s, \forall T \subset \{x | x \in V \wedge v_{s,x} = 1\}, T \neq \{\} \quad (14)$$

Angepasst an die Zielfunktion ist das zu minimierende Ziel bei diesem Modell die totale Wegzeit und die längste Route (1). Damit eine valide Lösung entsteht, müssen diverse Einschränkungen formuliert werden. Die längste Route wird durch die Summe aller benutzen Wege und der Besuchszeit eines Chlaus von unten beschränkt (2). Ein Chlaus darf nicht von der Familie  $i$  zur Familie  $i$  gehen (3). Jede Familie muss genau ein Mal besucht werden (4). Zudem muss ein Chlaus bei jeder Familie, die er besucht, ein Mal zur Familie kommen und ein Mal von der Familie weggehen (5). Um die Performance des Solvers zu verbessern (durch eine engere LP-Relaxation), wird diese Einschränkung auch über alle Chläuse gemacht (6). Die eingehenden und ausgehenden Wege müssen auch für den Startpunkt gleich sein (7). Jeder Chlaus muss den Startpunkt benutzen, sofern er mindestens eine Familie besucht (8)(9). Ein Chlaus muss gleich viele Wege beschreiten, wie er Orte, das heisst Familien und Startpunkt, besucht (10). Besucht ein Chlaus eine Familie, so muss er seine Pause ebenfalls durchführen (11)(12). Zudem darf kein Chlaus die Pause eines anderen Chlaus machen (13). Mit nur diesen Einschränkungen gibt es Lösungen, die Subtouren enthalten. Eine Subtour ist eine Route, die für einen Chlaus berechnet wurde und in der nicht alle Besuche zusammenhängen. In Abbildung 3 ist ein Beispiel einer solchen Subtour dargestellt.



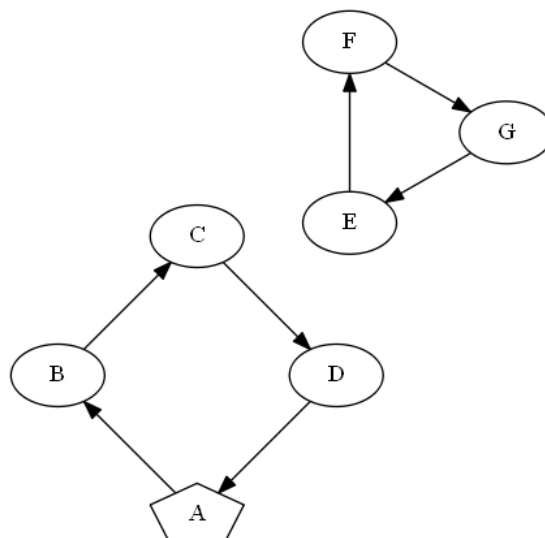


Abbildung 3: Subtour Problem, Startpunkt bei A

Dieses Subtouren-Problem wird dadurch gelöst, dass all diese Subtouren verboten werden (14). Weil das berechnen aller möglichen Subtouren ein zu grosser Aufwand ist, wird nach jeder Lösung überprüft, ob in der Lösung eine Subtour vorhanden ist. Ist dies der Fall, wird diese Subtour für alle Chläuse ausgeschlossen. Sobald die optimale Lösung ohne Subtouren gefunden wurde, ist die Phase 1 beendet.

Die zweite Phase löst das Optimierungsproblem vollständig. Bei dieser Formulierung werden deutlich mehr Variablen und Einschränkungen benötigt. Analog zur ersten Phase wird  $V \subseteq \mathbb{N}$  als Menge aller Besuche ohne Startpunkt,  $n_v = |V|$  als Anzahl der Besuche,  $B_s \subseteq V$  als Menge aller Pausen für Chlaus  $s$ ,  $S \subseteq \mathbb{N}$  als Menge aller Chläuse,  $n_s = |S|$  als Anzahl aller Chläuse,  $v_{s,i} \in \{0,1\}$  als Indikator, ob Chlaus  $s$  die Familie  $i$  besucht und  $w_{s,i,j}$  als Indikator, ob Chlaus  $s$  den Weg von Besuch  $i$  nach Besuch  $j$  benutzt, verwendet. Die Anzahl der Chläuse entspricht erneut der Anzahl Tage multipliziert mit den zur Verfügung stehenden Chläusen. Hinzu kommt die Variable  $c_{s,i} \in \mathbb{R}$ , welche angibt, zu welchem Zeitpunkt der Chlaus  $s$  den Besuch  $i$  abstattet. Dieser Zeitpunkt wird zum Tagesstart der Route dazugerechnet, um den effektiven Zeitpunkt zu erhalten. Die Startzeit in Sekunden des Arbeitstages von Chlaus  $s$  wird über die Variable  $p_{s,0}$  festgehalten. Das Ende des Arbeitstages wird von der Variable  $p_{s,1}$  bestimmt. Bei diesem Modell sind die Nichtverfügbarkeiten in der Menge  $U$  enthalten, die  $i$ -te Nichtverfügbarkeit für Besuch  $v$  wird als  $u_{v,i} \in U$  behandelt, wobei  $u_{v,i,0} \in \mathbb{R}$  den Startzeitpunkt der Nichtverfügbarkeit und  $u_{v,i,1} \in \mathbb{R}$  das Ende ist. Zur vereinfachten Schreibweise wird  $n_{u,v}$  als Anzahl aller Nichtverfügbarkeiten von Besuch  $v$  verwendet. Nach demselben Prinzip werden Wunschzeiten mit  $D$ ,  $d_{v,i} \in D$  sowie  $n_{d,v}$  verwendet.

Die Überschneidung zwischen der  $u$ -ten Nichtverfügbarkeit des Besuches  $i$  und der tatsächlichen Besuchszeit von Chlaus  $s$  wird mit der Variable  $unavailableDuration_{s,i,u} \in \mathbb{R}$  dargestellt. Analog dazu stellt die Variable  $desiredDuration_{s,i,d}$  die Überschneidung der  $d$ -ten Wunschzeit von Besuch  $i$  und der tatsächlichen Besuchszeit von Chlaus  $s$  dar. Weiter werden die Hilfsvariablen  $minRoute_s \in \mathbb{R} \forall s \in S$  für den Startzeitpunkt der Route sowie  $maxRoute_s \in \mathbb{R} \forall s \in S$  für den Rückkehrzeitpunkt des Chlauses  $s$  eingeführt. Weitere Hilfsvariablen, welche für die Wunschzeitberechnung benötigt werden, sind  $desiredStart_{s,i,d} \in \mathbb{R}$ ,  $desiredEnd_{s,i,d} \in \mathbb{R}$  und  $desiredOverlap_{s,i,d} \in \{0,1\}$  für  $\forall s \in S, \forall i \in V, \forall d \in D_i$ . Die Berechnung der Nichtverfügbarkeitsüberlappung benötigt folgende Hilfsvariablen:  $unavailableStart_{s,i,u} \in \mathbb{R}$ ,  $unavailableEnd_{s,i,u} \in \mathbb{R}$ ,  $bUnavailableStart_{s,i,u} \in \{0,1\}$ ,  $bUnavailableEnd_{s,i,u} \in \{0,1\}$  für  $\forall s \in S, \forall i \in V, \forall u \in U_i$ .

Die Abkürzung  $\forall sik$  steht nachfolgend für  $\forall s \in S, \forall i \in V, \forall k \in D_i$  sowie  $\forall sil$  für  $\forall s \in S, \forall i \in V, \forall l \in U_i$  steht.

minimize

$$\begin{aligned}
 & 12 \sum_{s=1}^{n_s} \sum_{i=1}^{n_v} \sum_{u=1}^{n_u(v)} unavailableDuration_{s,i,u} \\
 & + 4 \sum_{s=1}^{n_s} (maxRoute_s - minRoute_s) \\
 & - 2 \sum_{s=1}^{n_s} \sum_{i=1}^{n_v} \sum_{k=1}^{n_d(v)} desiredDuration_{s,i,k} \\
 & + 3 longestRoute
 \end{aligned} \tag{15}$$

s. t.

$$longestRoute \geq maxRoute_s - minRoute_s \quad \forall s \tag{16}$$

$$w_{s,i,i} = 0 \quad \forall s, \forall i \tag{17}$$

$$\sum_{s=1}^{n_s} v_{s,i} = 1 \quad \forall i \in V \setminus B \tag{18}$$

$$\sum_{s=1}^{n_s} \sum_{j=1}^{n_v} w_{s,i,j} = \sum_{s=1}^{n_s} \sum_{j=1}^{n_v} w_{s,j,i} = 1 \quad \forall i \in V \tag{19}$$

$$\sum_{j=1}^{n_v} w_{s,i,j} = \sum_{j=1}^{n_v} w_{s,j,i} = v_{s,i} \quad \forall s, \forall i \tag{20}$$

$$\sum_{j=1}^{n_v} w_{s,0,j} = \sum_{j=1}^{n_v} w_{s,j,0} \quad \forall s \tag{21}$$

$$\sum_{j=1}^{n_v} w_{s,0,j} \geq v_{s,i} \quad \forall i \quad (22)$$

$$\sum_{j=1}^{n_v} w_{s,0,j} \leq \sum_{i=1}^{n_v} v_{s,i} \quad \forall s \quad (23)$$

$$\sum_{i=0}^{n_v} \sum_{j=0}^{n_v} w_{s,i,j} = \sum_{i=0}^{n_v} \sum_{j=0}^{n_v} w_{s,j,i} = \sum_{i=0}^{n_v} v_{s,i} \quad \forall s \quad (24)$$

$$v_{s,b_s} \leq \sum_{j=0}^{n_v} v_{s,j} \quad \forall s \quad (25)$$

$$v_{s,b_s} \geq v_{s,i} \quad \forall s, \forall i \quad (26)$$

$$v_{s_2, b_{s_1}} = 0 \quad \forall s_1, s_2 \in S, s_1 \neq s_2 \quad (27)$$

$$c_{s,i} \leq p_{s,1} - p_{s,0} \quad \forall s, \forall i \quad (28)$$

$$v_{s,i} = 0 \rightarrow c_{s,i} = 0 \quad \forall s, \forall i \quad (29)$$

$$w_{s,j,i} = 1 \rightarrow c_{s,i} \geq c_{s,j} + \text{distance}_{j,i} + \text{duration}_j \quad \forall s, \forall i, \forall j \in V \cup \{0\} \quad (30)$$

$$\text{maxRoute}_s \geq c_{s,i} + v_{s,i}(\text{duration}_i + \text{distance}_{i,0}) \quad \forall s, \forall i \quad (31)$$

$$\text{minRoute}_s \leq c_{s,i} - \text{distance}_{0,i} v_{s,i} + M(1 - v_{s,i}) \quad \forall s, \forall i \quad (32)$$

$$\text{minRoute}_s \leq \text{maxRoute}_s \quad \forall s \in S \quad (33)$$

$$\text{desiredDuration}_{s,i,k} \leq \min\{\text{duration}_i, d_{i,k,1} - d_{i,k,0}\} \quad \forall s, i, k \quad (34)$$

$$\text{desiredStart}_{s,i,k} \geq \max\{d_{i,k,0} - p_{s,0}, 0\} \quad \forall s, i, k \quad (35)$$

$$\text{desiredStart}_{s,i,k} \leq p_{s,1} - p_{s,0} \quad \forall s, i, k \quad (36)$$

$$\text{desiredStart}_{s,i,k} \geq c_{s,i} \quad \forall s, i, k \quad (37)$$

$$\text{desiredEnd}_{s,i,k} \leq d_{i,k,1} - d_{i,k,0} \quad \forall s, i, k \quad (38)$$

$$\text{desiredEnd}_{s,i,k} \leq c_{s,i} + v_{s,i} \text{duration}_i \quad \forall s, i, k \quad (39)$$

$$\text{desiredOverlap}_{s,i,k} = 1 \rightarrow \text{desiredEnd}_{s,i,k} - \text{desiredStart}_{s,i,k} \geq 0 \quad \forall s, i, k \quad (40)$$

$$\text{desiredOverlap}_{s,i,k} = 1 \rightarrow \text{desiredDuration}_{s,i,k} = \text{desiredEnd}_{s,i,k} - \text{desiredStart}_{s,i,k} \quad \forall s, i, k \quad (41)$$

$$\begin{aligned} & \text{desiredOverlap}_{s,i,k} = 0 \\ & \rightarrow \text{desiredEnd}_{s,i,k} - \text{desiredStart}_{s,i,k} \leq 0 \quad \forall sik \end{aligned} \quad (42)$$

$$\text{desiredOverlap}_{s,i,k} = 0 \rightarrow \text{desiredDuration}_{s,i,k} = 0 \quad \forall sik \quad (43)$$

$$\text{unavailableDuration}_{s,i,l} \leq \min\{\text{duration}_i, u_{i,l,1} - u_{i,l,0}\} \quad \forall sil \quad (44)$$

$$\text{unavailableStart}_{s,i,l} \geq u_{i,l,0} - p_{s,0} \quad \forall sil \quad (45)$$

$$\text{unavailableStart}_{s,i,l} \leq p_{s,1} - p_{s,0} \quad \forall sil \quad (46)$$

$$\text{unavailableStart}_{s,i,l} \geq c_{s,i} \quad \forall sil \quad (47)$$

$$\begin{aligned} b\text{UnavailableStart}_{s,i,l} = 0 & \rightarrow \text{unavailableStart}_{s,i,l} \\ & \leq u_{i,l,0} - p_{s,0} \quad \forall sil \end{aligned} \quad (48)$$

$$b\text{UnavailableStart}_{s,i,l} = 1 \rightarrow \text{unavailableStart}_{s,i,l} \leq c_{s,i} \quad \forall sil \quad (49)$$

$$\text{unavailableEnd}_{s,i,l} \leq u_{s,i,l} - p_{s,0} \quad \forall sil \quad (50)$$

$$\text{unavailableEnd}_{s,i,l} \leq c_{s,i} + \text{duration}_i \quad \forall sil \quad (51)$$

$$\begin{aligned} b\text{UnavailableEnd}_{s,i,l} & = 0 \\ & \rightarrow \text{unavailableEnd}_{s,i,l} \geq u_{i,l,1} - p_{s,0} \quad \forall sil \end{aligned} \quad (52)$$

$$\begin{aligned} b\text{UnavailableStart}_{s,i,l} & = 1 \\ & \rightarrow \text{unavailableEnd}_{s,i,l} \geq c_{s,i} + \text{duration}_i \quad \forall sil \end{aligned} \quad (53)$$

$$\begin{aligned} \text{unavailableDuration}_{s,i,l} \\ & \geq \text{unavailableEnd}_{s,i,k} - \text{unavailableStart}_{s,i,k} \quad \forall sil \end{aligned} \quad (54)$$

Die Zielfunktion kann direkt aus der Problemstellung übernommen werden (15). Um numerische Ungenauigkeiten zu vermeiden, werden die Konstanten der ursprünglichen Zielfunktion mit 360 multipliziert. Die längste Route wird durch die maximale Differenz der Start- und der Endzeit aller Routen berechnet (16). So darf ein Chlaus den Weg von Besuch  $i$  zu Besuch  $i$  nicht beschreiten (17). Des Weiteren, muss jede Familie genau einmal besucht werden (18). Dabei muss jeder Besuch über einen Hinweg gestartet und über einen Rückweg wieder verlassen werden (19). Um die Performance des Solvers zu verbessern (durch eine engere LP-Relaxation), wird diese Einschränkung auch pro Chlaus angewendet (20). Dasselbe gilt für den Startpunkt: Falls ein Chlaus den Startpunkt verlässt, so muss er auch zurückkehren (21). Falls ein Chlaus eine Familie besucht, so muss er zwingend auch den Startpunkt besuchen (22)(23). Eine weitere Einschränkung, welche zu einem Performance-Gewinn führt, besagt, dass pro Chlaus die Anzahl begangener Wege und die Anzahl der Besuche gleich sein müssen (24). Wenn ein Chlaus eine Familie besucht und eine Pause eingetragen hat, so muss er diese zwingend machen (25)(26). Zudem darf diese Pause nur von dem eingetragenen Chlaus gemacht werden (27). Anders als bei der Phase 1 wird das Verhindern von Subtouren über die Zeit Variable  $c$  sichergestellt. Die Variable  $c$  wird in ihrem Maximum auf die Dauer des Arbeitstages

beschränkt (28). Falls eine Familie nicht von dem Chlaus besucht wird, so wird auch keine Zeit eingetragen (29). Wird der Weg von Familie  $i$  zur Familie  $j$  begangen, so muss die Familie  $j$  später als die Familie  $i$  besucht werden (30). Der Mindestabstand zwischen dem Besuchsstart von Familie  $i$  und dem Besuchsstart von Familie  $j$  ergibt sich aus der Besuchszeit von Familie  $i$  sowie der Wegzeit von  $i$  nach  $j$ . Durch diese Einschränkung erhöht sich die Zeit mit jedem begangenen Weg und verhindert somit Subtouren. Abbildung 4 ist eine Visualisierung, dieser Einschränkung.

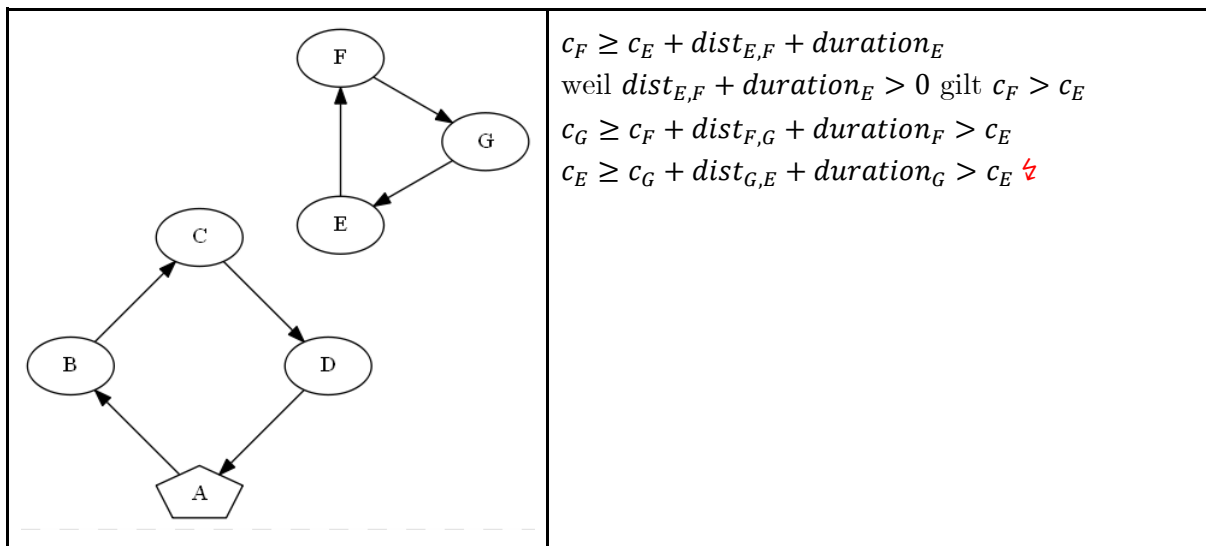


Abbildung 4: Subtour Eliminierung

Der Zeitpunkt der Rückkehr kann durch eine Maximum-Einschränkung der Zeitvariable fixiert werden. Der Zeitpunkt der Rückkehr ist später als jedes Besuchsende plus der Rückweg zum Startpunkt (31). Der Startpunkt der Route kann durch eine Minimal-Einschränkung der Zeitvariable minus dem Weg vom Startpunkt zu dem Besuch beschränkt werden. Weil die Zeitvariable für eine Familie, die der Chlaus nicht besucht, den Wert 0 annimmt, muss diese Einschränkung mit einer Big-M Formulierung (für mehr Informationen siehe [14]) gelöst werden (32). Das M wird hier auf die obere Grenze der Variable  $minRoute_s$  gesetzt, was der Tagesdauer entspricht. Der Routenstart ist in jedem Fall vor dem Routenende (33).

Damit die erfüllte Wunschzeit berechnet werden kann, muss die Überlappung zwischen der Wunschzeit des Besuches und der tatsächlichen Besuchszeit berechnet werden. Die erfüllte Wunschzeit wird maximal so lange, wie der kleinere Wert zwischen der Besuchsdauer und der Dauer der Wunschzeit (34). Der Start dieser Überlappung liegt frühestens bei dem Startpunkt der Wunschzeit (35). Er liegt spätestens beim Tagesende (36) und frühestens beim Startpunkt der Besuchszeit (37). Das Ende der Überlappung liegt spätestens am Tagesende (38) und spätestens Ende des Besuches (39). Wird die Familie nicht besucht, so wird das Ende der Überlappung auf null gesetzt. Ist das Ende der Überlappung später als der Anfang, so gibt es eine Überlappung (40) und die erfüllte Wunschzeit wird durch die Differenz der beiden berechnet (41). Ist das Ende vor dem Start (42), so wird die Wunschzeit nicht erfüllt (43). Diese Einschränkungen führen nur zu einer oberen Beschränkung der Überlappung. Denn dadurch, dass die Zielfunktion die Kosten minimiert und die Wunschzeit mit einer negativen Konstante multipliziert wird, ist die maximale mögliche Wunschzeit das Optimum, welches angestrebt wird. In Abbildung 5 werden diese Einschränkungen dargestellt.

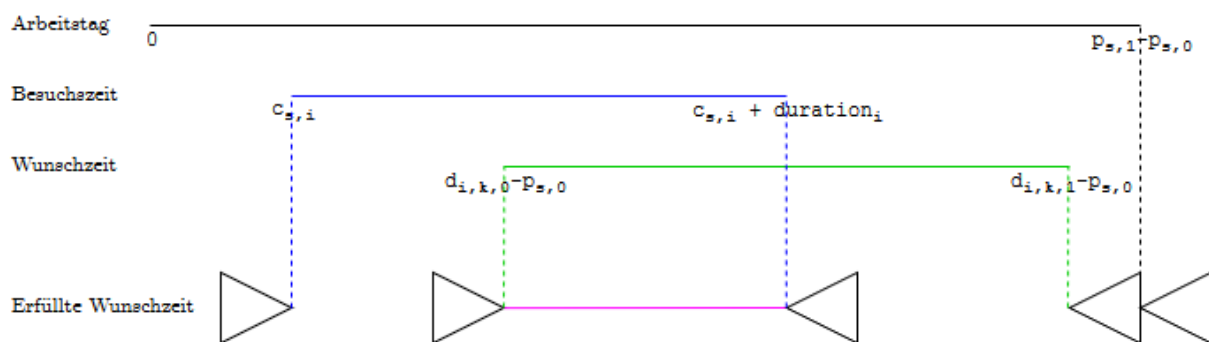


Abbildung 5: Wunschzeit Einschränkungen

Für die Zeit in der Nichtverfügbarkeit kann nicht dasselbe gemacht werden, da diese Zeit in der Zielfunktion ein positives Gewicht hat. Die maximale Nichtverfügbarkeit pro Besuch ist durch die Besuchsdauer sowie durch die Gesamtdauer der Nichtverfügbarkeit beschränkt (44). Der Start der Überlappung ist frühestens beim späteren Zeitpunkt zwischen Start der Nichtverfügbarkeit und Start des Besuches. Zudem ist der Start der Überlappung früher als das Tagesende (45)(46)(47). Der Start der Überlappung muss auf den späteren Zeitpunkt fixiert werden. Hierfür wird die binäre Entscheidungsvariable  $bUnavailableStart_{s,i,l}$  verwendet. Hat diese Entscheidungsvariable den Wert 0, so gilt die Einschränkung, dass der Start der Überlappung spätestens beim Start der Nichtverfügbarkeit ist (48). Dadurch wird der Wert fixiert. Hat  $bUnavailableStart_{s,i,l}$  den Wert 1, so gilt, dass der Start der Überlappung auf den Start des Besuches fixiert wird (49). Eine der beiden Einschränkungen führt dazu, dass das Problem nicht mehr lösbar ist, wodurch der Wert von  $bUnavailableStart_{s,i,l}$  eindeutig bestimmt wird.

Ähnlich wird das Ende der Überlappung bestimmt. Dieses Ende liegt spätestens am früheren Zeitpunkt zwischen Ende der Nichtverfügbarkeit und Ende des Besuches (50)(51). Mithilfe der binären Entscheidungsvariable  $bUnavailableEnd_{s,i,l}$  wird das Ende der Überlappung fixiert. Nimmt die  $bUnavailableEnd_{s,i,l}$  den Wert 0 an, so ist das Ende frühestens beim Ende der Nichtverfügbarkeit (52), bei 1 ist es frühestens am Ende des Besuches (53). Auch hier führt eine der beiden Einschränkungen dazu, dass das Problem nicht mehr lösbar ist und dadurch der Wert der Entscheidungsvariable bestimmt wird. In Abbildung 6 sind die Nichtverfügbarkeits-Einschränkungen dargestellt. Die Dauer der Überlappung wird aus der Differenz der End- und Startzeit berechnet (54)

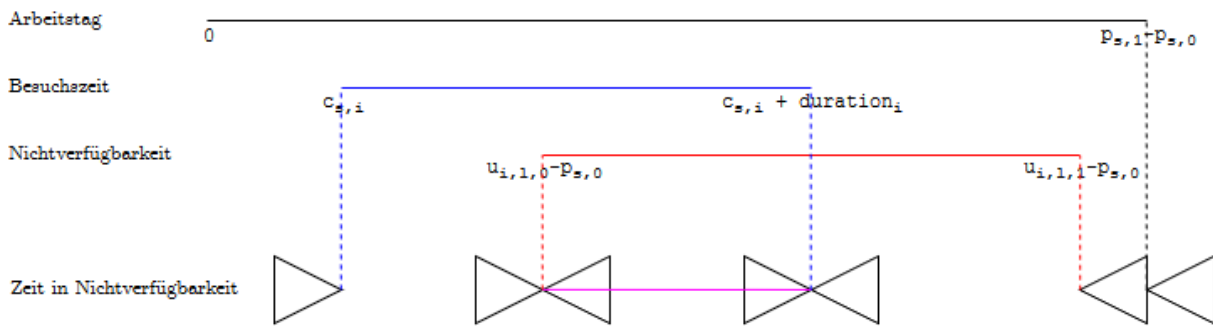


Abbildung 6: Nichtverfügbarkeit Einschränkungen

### 3.3 Genetischer Algorithmus

Bei genetischen Algorithmen (GA) handelt es sich um Suchverfahren, die nach dem Prinzip der biologischen Evolution auf Basis des Darwinismus funktionieren. Genetische Algorithmen gehören zu den Metaheuristiken. Metaheuristiken sind Optimierungsverfahren, die für theoretisch beliebige Probleme eingesetzt werden können. Sie definieren dabei lediglich abstrakte Vorgehensweisen, welche problemspezifisch implementiert werden müssen. Laut [9] eignen sich genetische Algorithmen vor allem für kombinatorische Probleme.

Um den Aufbau von genetischen Algorithmen verstehen zu können, müssen zuerst einige Begriffe eingeführt werden. Zuerst muss der Suchraum unterteilt werden. Der eigentliche Suchraum, wie er für die Chlausgesellschaft verständlich ist, wird als phänotypischer Suchraum bezeichnet. Eine einzelne Lösung ist demnach ein Phänotyp. Ein Phänotyp enthält bei unserer Problemstellung die Information, welcher Chlaus zu welchem Zeitpunkt welche Familie besucht. Demnach wird ein Phänotyp gebraucht, um die Kosten einer Planung zu berechnen. Diese Kosten geben an, wie gut die Qualität einer Lösung ist. Die Lösungsqualität wird auch Fitness genannt. Evolutionsoperatoren, welche später beschrieben werden, können in der Regel nur auf die genetischen Repräsentationen der Phänotypen angewandt werden. Diese Repräsentation wird Genotyp genannt. Mithilfe einer Dekodierungsfunktion kann ein Genotyp in einen Phänotyp umgewandelt werden.

Wie in Abbildung 7 gezeigt, sieht der grundsätzliche Ablauf von genetischen Algorithmen vor, dass als Erstes eine Startpopulation kreiert wird. Die Individuen dieser Population sind Genotypen. Diese Startpopulation enthält meist zufällig generierte Lösungen. Danach beginnt der eigentliche Evolutionsprozess, bei dem die Evolutionsoperatoren zum Einsatz kommen. Bei der Selektion werden zuerst - anhand der Fitness - Individuen bestimmt, die sich fortpflanzen sollen. Danach werden die Nachkommen mittels geeigneter Rekombination erzeugt. Als nächstes können die Individuen für die neue Generation mutiert werden. Der letzte Schritt der Evolution besteht schliesslich darin, die alte Generation durch die neue zu ersetzen. Das Ziel ist es, dass die letzte Generation eine genügend gute Lösung enthält [9].

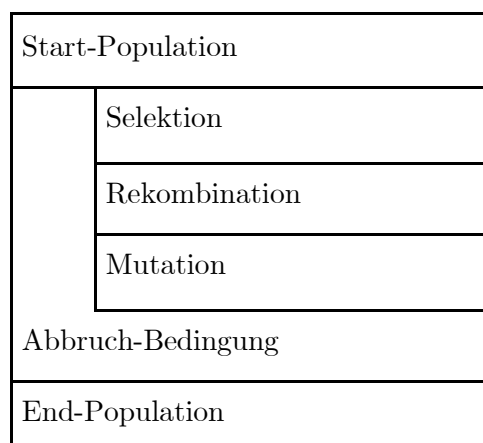


Abbildung 7: Vereinfachter Ablauf eines genetischen Algorithmus angelehnt an [9]



Im Rahmen dieser Arbeit wird ein erweitertes Modell [9] eingesetzt, welches am Institut für 4D-Technologien an der FHNW zur Optimierung unterschiedlicher Probleme eingesetzt wird. Der erste Schritt des genetischen Algorithmus ist das Erzeugen der Startpopulation. Dabei wird eine bestimmte Anzahl Genotypen zufällig erzeugt. Als nächstes beginnt die eigentliche Evolution. Wie in Abbildung 8 dargestellt, können die Individuen der nächsten Population aus verschiedenen Quellen stammen. Der erste Anteil kommt über den Elitismus. Das bedeutet, dass die besten  $q\%$  direkt in die nächste Generation übernommen werden. In dieser Implementation ist es wichtig, dass mindestens das beste Individuum übernommen wird, um die beste Lösung zu erhalten. Die nächsten  $r\%$  werden zuerst selektiert und anschliessend direkt mutiert. Die nächsten  $s\%$  werden durch Rekombination erzeugt und anschliessend mit einer bestimmten Wahrscheinlichkeit mutiert. Zum Schluss werden noch  $t\%$  an zufälligen Individuen übernommen, die analog zur Startpopulation erzeugt werden.

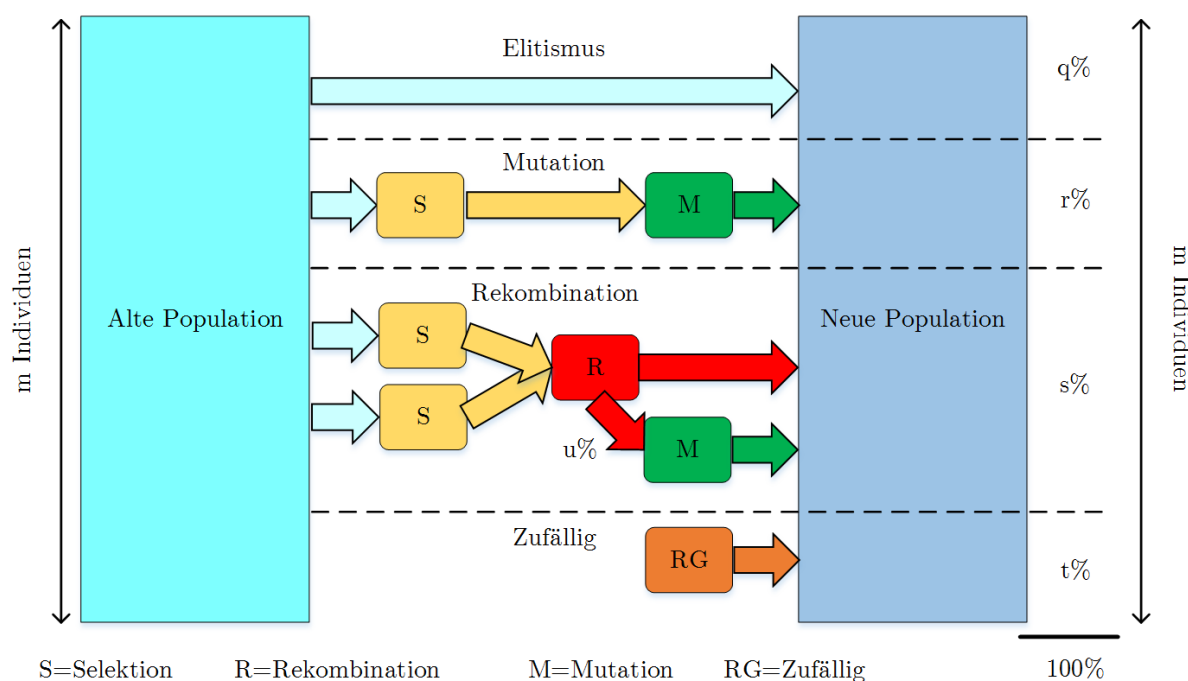


Abbildung 8: Aufbau des eingesetzten Modells des GA angelehnt an [9]

Um die genaue Funktionsweise des genetischen Algorithmus, welcher für dieses Projekt entwickelt wurde, zu beschreiben, muss zuerst die Codierung erklärt werden. Eine Codierung beschreibt, wie ein Genotyp aufgebaut ist. In diesem Modell entspricht ein Genotyp aus einer Liste ganzzahliger Werte. Diese Werte werden auch Allele genannt. In der Genetik gibt es beispielsweise ein Allel für blonde Haare. Wie ein Genotyp aussehen könnte, ist in Abbildung 9 dargestellt [9].

0	8	5	4	3	-1	6	1	2	7
---	---	---	---	---	----	---	---	---	---

Abbildung 9: Beispielgenotyp

Es gibt in diesem Modell zwei Arten von Allelen. Allele mit einem Wert grösser gleich null repräsentieren Besuche oder Pausen. Allele mit negativem Wert sind Separatoren. Diese

Separatoren trennen zwei Routen voneinander. Im Beispiel aus Abbildung 9 werden bei der ersten Route die Familien 0, 8, 5, 4 und 3 besucht. Die Besuchsreihenfolge wird von der Reihenfolge innerhalb des Genotyps vorgegeben. In der zweiten Route werden die Familien 6,1,2 und 7 besucht. Bei den Besuchen gilt es noch zu beachten, dass die Pausen für jeden Tag dupliziert werden. Das heisst, wenn ein Chlaus eine Pause machen möchte und zwei Arbeitstage vorgesehen sind, gibt es die Pause danach zwei Mal. Zwei Mal heisst, dass es zwei Besuche mit unterschiedlicher Nummer gibt, die am Ende auf die originale Pause abgebildet werden. Das ist bedingt durch die Struktur des Genotyps. Routen, die von Separatoren getrennt sind, gehören zu einem Chlaus und zu einem Tag. Demnach kann der obige Genotyp auf zwei Arten interpretiert werden. Die erste Route gehört jedoch in beiden Fällen zum ersten Chlaus und zum ersten Tag. Bei der ersten Interpretation kann angenommen werden, dass es einen Chlaus und zwei Tage gibt. Somit gehört die zweite Route zum ersten Claus und zum zweiten Tag. Alternativ kann angenommen werden, dass es zwei Chläuse und einen Tag gibt. In diesem Fall gehört die zweite Route zum zweiten Chlaus und zum ersten Tag.

Andere genetische Algorithmen [15], die für das klassische VRPTW entwickelt wurden, haben oftmals keine Separatoren. Die Routen werden dort so erstellt, dass der Genotyp von links nach rechts durchlaufen wird und so viele Besuche wie möglich in die Route eingefügt werden. Erst wenn eine Route voll ist, wird eine neue erstellt. Voll bedeutet in diesem Fall, dass beim Einfügen eines weiteren Besuchs die maximale Dauer einer Route überschritten würde. Dieses Vorgehen ist jedoch für diese Problemstellung ungeeignet, da die Routen gemäss der Zielfunktion möglichst gleich lange dauern sollen.

Da für das Berechnen der Fitness ein Phänotyp benötigt wird, muss auch eine Dekodierungsfunktion definiert werden. Die Aufteilung der Routen und die Besuchsreihenfolge sind bereits im Genotyp direkt ersichtlich. Dazu gehört auch, dass jeder Route ein Chlaus und ein Tag zugewiesen ist. Was noch fehlt, sind die Startzeitpunkte der Besuche. Der Startzeitpunkt des ersten Besuchs jeder Route entspricht dem Start des dazugehörigen Tages plus der Wegzeit vom Startort zum ersten Besuch. Der Startzeitpunkt aller weiteren Besuche entspricht dem Startzeitpunkt des vorherigen Besuchs plus der Dauer des vorherigen Besuchs plus Wegzeit zwischen den beiden Besuchen. Diese Methodik hat die folgenden beiden Einschränkungen: Eine Route beginnt immer am Anfang eines Tages. Hernach können dadurch keine Wartezeiten eingeplant werden, um beispielsweise eine Familie nicht zu früh zu besuchen. Der Grund für diese Entscheidung liegt vor allem im Datensatz 34VDU aus dem Jahr 2017, in dem alle Besuche problemlos innerhalb der Wunschzeiten besucht werden konnten, ohne dass Chläuse später hätten starten müssen. Zudem wäre der Genotyp und damit der genetische Algorithmus wesentlich aufwändiger geworden, wenn Start- und Wegzeit variabel sein sollen hätten.

Als Selektionsmechanismus wurde wie in [9] [15] die binäre Wettkampfsituation implementiert. Um dabei ein Elternpaar für die Rekombination zu finden, werden im ersten Schritt zwei zufällige Individuen aus der Population ausgewählt. Danach wird von diesen beiden das Individuum mit der besseren Fitness als erster Elternteil akzeptiert. Der zweite Elternteil wird, analog aus zwei weiteren, zufälligen Individuen bestimmt.

Rekombination, häufig auch “crossover” genannt, bedeutet Informationsaustausch zwischen zwei gepaarten Individuen. Das bedeutet in diesem Fall, dass aus zwei Elternteilen ein Nachkomme kreiert werden muss. Es wurden zwei Varianten umgesetzt, welche sich gemäss [16] für das Traveling Salesman Problem (TSP) eignen sollen. Das VRPTW ist eine Erweiterung des TSP.

Die erste Variante ist der sogenannte “Order-based Crossover” (OX2). Dieses Verfahren wurde in Zusammenhang mit Zeitplanungsproblemen vorgeschlagen. Der OX2 Operator selektiert zuerst ein paar zufällige Positionen in einem Elternteil. Danach wird die Reihenfolge der selektierten Allele auf das andere Elternteil aufgezwungen. Als Beispiel werden die beiden Elternteile (1,2,3,4,5,6,7,8) und (2,4,6,8,7,5,3,1) genommen. Als nächstes würden zufällig die Positionen zwei, drei und sechs ausgewählt. Die Allele an diesen Positionen sind 4, 6 und 5. Diese Allele befinden sich im ersten Elternteil an der vierten, fünften und sechsten Position. Der Nachkomme entspricht jetzt dem ersten Elternteil, aber ohne die vierte, fünfte und sechste Position: (1,2,3,\*,\*,\*,7,8). Die fehlenden Allele werden in der Reihenfolge hinzugefügt, wie sie im zweiten Elternteil vorkommen. Das Resultat ist (1,2,3,4,6,5,7,8). Würden die Elternteile vertauscht werden, entstünde der Nachkomme (2,4,3,8,7,5,6,1).

Die zweite Variante ist der sogenannte “Edge Recombination Crossover” (ER). Der ER eignet sich gut, wenn die verwendeten Kanten der Elternpaare beibehalten werden sollen und ist so der Ausgleich zu OX2. Bei ER wird für jedes Allel die Adjazenzmenge erstellt, d.h. eine Liste mit allen benachbarten Allelen. Die Strategie ist, dass zuerst aus einem Elternteil das erste Allel genommen wird und anschliessend mithilfe der Adjazenzmenge das nächste Allel so gewählt wird, dass das nächste Allel möglichst wenig verbleibende Nachbarn hat. Dafür muss ein Allel, das zum Nachkommen hinzugefügt wird, aus allen Adjazenzmengen entfernt werden. Als Beispiel werden die beiden Elternteile (1,3,5,6,4,2,8,7) und (1,4,2,3,6,5,7,8) genommen. Die dazugehörigen Adjazenzmengen lauten wie folgt:

Allel 1 hat Nachbarn: 3, 4, 7, 8

Allel 2 hat Nachbarn: 3, 4, 8

Allel 3 hat Nachbarn: 1, 2, 5, 6

Allel 4 hat Nachbarn: 1, 2, 6

Allel 5 hat Nachbarn: 3, 6, 7

Allel 6 hat Nachbarn: 3, 4, 5

Allel 7 hat Nachbarn: 1, 5, 8

Allel 8 hat Nachbarn: 1, 2, 7

In diesem Beispiel würde der Nachkomme mit dem Allel 1 starten. Demnach muss das Allel 1 aus allen Adjazenzmengen entfernt werden. Allel 1 hat die Nachbarn 3, 4, 7 und 9. Das Allel 3 hat drei Nachbarn, da das Allel 1 bereits entfernt wurde. Die Allele 4, 7 und 8 haben alle zwei

Nachbarn, somit kann aus diesen dreien zufällig ein Allel gewählt werden. In diesem Fall wird das Allel 8 gewählt und dem Nachkommen hinzugefügt. Allel 8 hat die Nachbarn 2 und 7. Als nächstes wird das Allel 7 gewählt, da es einen Nachbarn weniger hat als das Allel 2. Allel 7 hat nur noch den Nachbarn 5. Vom Allel 5 kann wieder zufällig zwischen Allel 3 und 6 ausgewählt werden, da beide zwei Nachbarn haben. Hier wird das Allel 6 zufällig gewählt. Von Allel 6 gibt es die Nachbarn 3 und 4, mit jeweils einem Nachbarn. Es wird das Allel 4 gewählt. Das Allel 4 hat nur den Nachbarn 2, welcher einzig den Nachbarn 3 hat. Der Nachkomme ist also am Ende (1,8,7,5,6,4,2,3) [16].

Mutation bedeutet bei genetischen Algorithmen [9], dass ein Individuum beim Übergang in eine neue Generation zufällig abgeändert wird. Oftmals werden Mutationen von der Selektion wieder ausgemerzt, weil sie für die Fitness keinen Vorteil bringen. Teilweise können Mutationen jedoch überleben und zu besseren Lösungen führen. Generell sollen die Mutationen aber auch die Wahrscheinlichkeit reduzieren, dass der Algorithmus zu früh an einer lokalen Extremalstelle konvergiert und so die besten Lösungen nicht gefunden werden. Für dieses Modell wurde die Positionsmutation und die Inversion gewählt. Bei der Positionsmutation werden zwei zufällig gewählte Allele miteinander vertauscht. Bei der Inversion wird ein zusammenhängender Bereich gewählt und die Reihenfolge der Allele wird invertiert. Um lokalen Extremalstellen besser entkommen zu können, wird bei einer Mutation auch das Ausmass berechnet. Das Ausmass, oder die Schwere einer Mutation sagt aus, wie stark der Genotyp verändert wird. Um die Evolution nicht zu stark zu beeinträchtigen, sollte eine Mutation meistens ein kleines Ausmass haben. Nur mit einer geringen Wahrscheinlichkeit soll die Mutation den Genotypen stark verändern. Um diese Eigenschaft abzubilden, wird die Mutationsgrösse mittels einer Normalverteilung berechnet, wobei die Standardabweichung auf  $\frac{1}{4}$  der Anzahl Allele festgelegt wird. Für den Erwartungswert wird bei der Positionsmutation eins und bei der Inversion zwei genommen. Da eine Mutation nicht ein negatives Ausmass haben kann, wird der Betrag der Abweichung zum Erwartungswert addiert. Ein Beispiel zu den beiden Mutationen findet sich in Abbildung 10.

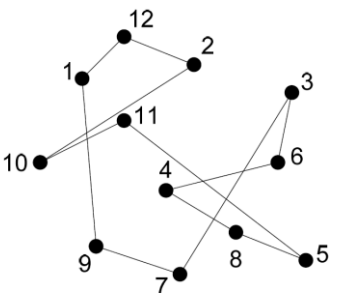
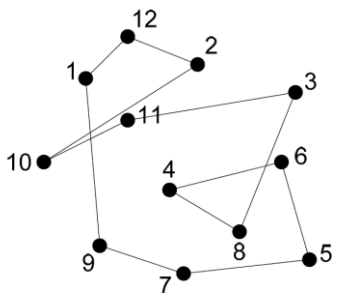
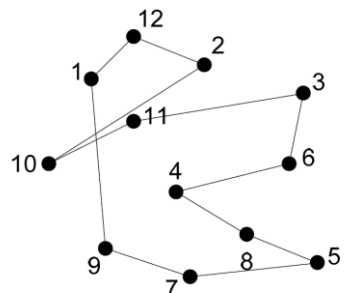
Original	Positionsmutation 3 ↔ 5 Ausmass: 1	Inversion 5,8,4,6,3 Ausmass: 5
(1,12,2,10,11,5,8,4,6,3,7,9)	(1,12,2,10,11,3,8,4,6,5,7,9)	(1,12,2,10,11,3,6,4,8,5,7,9)
		

Abbildung 10: Beispiel Positions- und Inversionsmutation angelehnt an [9]

Wenn nun der Aufbau sowie die Evolutionsoperatoren des genetischen Algorithmus bekannt sind, stellt sich die Frage nach der Konfiguration. Beispielsweise müssen die Populationsgrösse  $m$  und der prozentuale Anteil an zufällig generierten Individuen  $t$  bestimmt werden. Eine Möglichkeit, diese Parameter zu bestimmen, eröffnet sich mittels sogenannter “Meta-Optimization”. Das bedeutet, dass mithilfe eines Optimierungsverfahren ein anderes Optimierungsverfahren verbessert wird. Diese Methodik wurde bereits im Jahr 1978 von Mercer und Sampson [17] verwendet, um die optimalen Parameter für einen genetischen Algorithmus zu finden.

In diesem Projekt wird eine Partikel-Schwarm-Optimierung (PSO) verwendet, um die Parameter des genetischen Algorithmus zu optimieren. PSO ist selbst eine Metaheuristik und eignet sich gut für numerische Optimierungsprobleme. Für eine genauere Beschreibung des Verfahrens sei auf [9] [18] verwiesen.

Wichtig ist zu wissen, dass beim PSO eine Zielfunktion definiert werden muss. In diesem Fall sieht das so aus, dass in der Zielfunktion der GA aufgerufen wird. Aufgrund dessen, dass der GA ein stochastisches Suchverfahren ist, sind die Resultate nicht immer gleich gut. Um die Zielfunktion stabiler zu machen, wird der GA zehn Mal aufgerufen. Aus diesen zehn Durchläufen wird der Durchschnitt gebildet. Der Grund für die Wahl der Durchschnittsfunktion ist, dass dadurch ein möglichst genauer Wert entsteht und ein Optimum besser gefunden werden sollte. Würde beispielsweise immer der maximale Wert genommen, wäre das Resultat stärker vom Zufall abhängig als beim Durchschnitt.

Bei der Zielfunktion sollten bestenfalls alle 12 Datensätze aus Kapitel 2 berechnet werden. Jedoch würde das zu lange dauern. Aus diesem Grund wird ein neu generierter Datensatz verwendet, der die 12 Datensätze repräsentieren soll. Der Datensatz wird ebenfalls nach dem Schema aus Kapitel 2 erzeugt und hat die folgenden Eigenschaften. Er besteht aus 16 Familienbesuchen, zwei Chläusen und zwei Arbeitstagen. Beide Chläuse haben eine eingetragene Pause. Die Besuche haben zehn Wunschzeiten und zehn Nichtverfügbarkeiten, welche gleichmässig auf die beiden Tage aufgeteilt sind. Als Zeitlimit für die Berechnung wird eine halbe Sekunde genommen.

Die Konfiguration des GA besteht aus sieben Parametern, die definiert werden sollen. Zuerst sind das die Prozentwerte aus Abbildung 8. Dabei müssen nur drei Werte optimiert werden, da  $s$  so gewählt wird, dass die Summe von  $q, r, s$  und  $t$  gleich eins ist. Der nächste Parameter ist  $u$ . Dieser steht für die Wahrscheinlichkeit, dass ein Individuum, welches durch Rekombination entstanden ist, zusätzlich mutiert wird. Ferner muss definiert werden, welcher Operator für die Rekombination verwendet wird. Der Parameter  $o$  gibt an, mit welcher Wahrscheinlichkeit OX2 verwendet wird. ER wird demnach mit einer Wahrscheinlichkeit von  $1 - o$  verwendet. Ähnlich verhält es sich bei den beiden möglichen Mutationen. Mit einer Wahrscheinlichkeit von  $p$  wird die Positionsmutation verwendet, andernfalls wird eine Inversion durchgeführt. Als letztes muss die Populationsgrösse  $m$  berechnet werden.

Dieser Abschnitt befasst sich mit der genauen Konfiguration des PSO und ist für das weitere Verständnis nicht zwingend notwendig. Wie schon erwähnt, ist PSO eine Metaheuristik. Dementsprechend gibt es beim PSO auch Parameter, die festgelegt werden müssen. Da diese Parameter nicht einfach zu bestimmen sind, werden gute Parameter aus einem Bericht [18] von Magnus Erik Hvass Pedersen verwendet. Diese guten Parameter wurden mithilfe von Meta-Optimization verschiedener Problemstellungen gefunden. Um die Konfiguration des GA zu optimieren, kommen abgeleitet aus dem Bericht [18] zwei Problemgrößen für die PSO-Konfiguration in Frage: Erstens eine Problemgröße, die fünf Parameter hat und 10'000 Evaluationen der Zielfunktion durchführt. Zweitens eine Problemgröße, die zehn Parameter hat und 20'000 Evaluationen der Zielfunktion durchführt. Wie oben beschrieben, müssen für den GA sieben Parameter optimiert werden. Die Anzahl der Auswertungen der Zielfunktion wird auf 17280 gesetzt. Das entspricht einer Berechnungszeit von drei Stunden für das Auswerten der Zielfunktion. Die Zielfunktionen mit einer Berechnungszeit von  $10 \times 500$ ms pro Auswertung, sollen achtfach parallel ausgeführt werden. Wie aus dem Bericht [18] ersichtlich, gibt es für die erste Problemgröße zwei mögliche PSO-Konfigurationen. Damit die beiden PSO-Konfigurationen möglichst unterschiedlich sind, wird die Konfiguration mit  $S = 203$  genommen. Die beiden PSO-Konfigurationen werden nachfolgend mit C1 ( $S = 203, \omega = 0.5069, \phi_p = 2.5524, \phi_g = 1.0056$ ) und C2 ( $S = 53, \omega = -0.3488, \phi_p = -0.2699, \phi_g = 3.3950$ ) bezeichnet. Beim Ausführen des PSO könnte noch der Fall auftreten, dass ein Parameter ungültig wird. In diesem Fall wird der Parameter auf den nächsten gültigen Wert gesetzt. Ein Beispiel für einen ungültigen Parameter ist eine negative Populationsgröße, diese würde auf zwei korrigiert werden.

Da der PSO ebenfalls ein stochastisches Suchverfahren ist, werden pro PSO-Konfiguration drei Durchläufe gemacht. Die Tabelle 6 zeigt die Resultate der insgesamt sechs Durchgänge. Daraus lassen sich folgende Beobachtungen ableiten. Der Elitismus  $q$  macht über einen Drittel der Population aus, was relativ hoch ist. Der hohe Elitismus führt dazu, dass die Population schnell konvergiert, dafür wird der GA anfälliger für lokale Extremalstellen [9]. Der Operator OX2 ist gegenüber ER klar zu bevorzugen. Ein Grund könnte sein, dass OX2 weniger Rechenzeit braucht als ER. Bis auf eine Ausnahme schneidet die Positionsmutation besser ab als die Inversion. Bei den Populationsgrößen wird mehrfach zwei genommen, was für praktische Anwendungen ungeeignet ist. Denn mit zwei Individuen fehlt die Diversität der Population, was die Rekombination nutzlos macht. Dementsprechend kommen gemäss [19] bei den meisten Implementationen von genetischen Algorithmen Populationsgrößen von mindestens 30 zum Einsatz.

Aufgrund des besten Werts für die Kosten und der geeigneten Populationsgrösse, werden die Parameter in der dritten Spalte für den GA verwendet.

Konfiguration	C1	C1	C1	C2	C2	C2
q (Elitismus)	51.49%	35.67%	66.08%	99.82%	59.08%	60.18%
r (Mutation)	4.47%	37.8%	1.08%	0.17%	40.91%	0.00%
s (Rekombination)	21.76%	26.53%	18.36%	0.01%	0.01%	39.82%
t (Zufällig)	22.28%	0.00%	14.48%	0.00%	0.00%	0.00%
u (Mutation nach Rekombination)	34.86%	0.00%	11.01%	8.32%	0.00%	0.00%
o (OX2)	90.37%	88.37%	100%	90.39%	100%	100%
p (Positionsmutation)	39.84%	88.56%	76.94%	75.56%	68.97%	59.44%
m (Populationsgrösse)	2	102	2	2	2	2
Kosten in CHF	783.8	779.7	781.3	783.1	782.3	783.9

Tabelle 6: Resultate PSO mit Zeitlimit 0.5 s

Um zu zeigen, dass die angenommenen 0.5 Sekunden für eine Berechnung des GA nicht zu viel sind, wird der obige Test nochmal mit 0.25 Sekunden pro Berechnung wiederholt. Die Resultate des Tests mit 0.25 Sekunden sind in Tabelle 7 ersichtlich. Bei genauerer Betrachtung fällt auf, dass die numerischen Werte zwischen den Durchgängen stark voneinander abweichen oder gar zufällig aussehen. Die grossen Unterschiede deuten darauf hin, dass die Qualität der Resultate stark vom Zufall abhängt und die 0.25 Sekunden nicht ausreichen.

Konfiguration	C1	C1	C1	C2	C2	C2
q (Elitismus)	61.62%	66.51%	55.23%	26.33%	40.98%	21.45%
r (Mutation)	35.08%	17.03%	0.00%	0.34%	7.24%	19.50%
s (Rekombination)	0.01%	6.20%	0.01%	73.22%	51.78%	59.05%
t (Zufällig)	3.29%	10.26%	44.76%	0.11%	0.00%	0.00%
u (Mutation nach Rekombination)	99.95%	71.29%	89.81%	94.73%	100%	99.87%
o (OX2)	3.14%	10.53%	11.89%	52.18%	43.63%	0%
p (Positionsmutation)	35.48%	46.68%	48.23%	0.56%	100%	0%
m (Populationsgrösse)	2	2	2	62	51	18
Kosten in CHF	785.3	790.5	786.8	791.8	787.3	793.3

Tabelle 7: Resultate PSO mit Zeitlimit 0.25 s

Die Berechnungen vom PSO belegen, dass die Population für den GA aus 102 Individuen bestehen soll, wenn das Zeitlimit für den GA 0.5 Sekunden beträgt. Bei den Datensätzen, die für die Evaluation verwendet werden, ist das Zeitlimit jedoch wesentlich höher. Zudem kommt eine Arbeit von Gotshall und Rylander [20] zum Schluss, dass die Populationsgrösse abhängig von der Problemgrösse sein soll. Um dem Rechnung zu tragen, werden die 12 Datensätze aus Kapitel 2 innerhalb der Zeitlimits mit verschiedenen Populationsgrössen berechnet. Die Populationsgrössen werden mittels Exponentialfunktion mit Basis zwei erstellt. Die Basis zwei bietet sich an, weil bei kleinen Populationsgrössen eine gute Auflösung erzielt wird. Die kleinste getestete Populationsgrösse ist zwei, was auch der minimalen Anzahl Individuen entspricht, die der GA für eine korrekte Funktionsweise benötigt. Die grösste getestete Populationsgrösse ist 1048576, was  $2^{20}$  entspricht. Pro Populationsgrösse werden sechs Durchläufe gemacht, die parallel auf einem Lenovo W541 berechnet wurden. Dass genau sechs Instanzen parallel laufen, begründet sich dadurch, dass dies den Bedingungen der Evaluation aus Kapitel 4.1 am nächsten kommt, was auch aus dem Anhang 8.5 ersichtlich ist. Als Mass für die Performance wird die Anzahl der erzeugten Generationen verwendet. Der Anhang 8.5 zeigt auch, dass bei der Evaluation am besten zwei GA parallel ausgeführt werden sollen, um die Rechenkapazität optimal auszunutzen. Die insgesamt 1440 Messwerte sind online [21] zu finden.



Um pro Populationsgrösse auf eine einzige Zahl zu kommen, wird entsprechend der Evaluation das dritte Quartil genommen. Bei der Betrachtung der Messwerte pro Datensatz fällt auf, dass sechs Messwerte nicht genug sind, um allfällige Schwankungen des GA zu kompensieren. Als Beispiel werden in Abbildung 11 die Werte von Datensatz 100VBDU gezeigt. Um diese Schwankungen auszugleichen, werden diese Werte geglättet. Beim Glätten wird statt des originalen Messwertes der Durchschnitt aus dem originalen Messwert und den beiden benachbarten Messwerten genommen (sog. «moving average»). Wenn pro Instanzgrösse der beste, geglättete Wert genommen wird, resultiert das in der Zuweisung aus Tabelle 8. Mit Instanzgrösse sind die Anzahl Besuche gemäss Definition in Kapitel 2 gemeint. Bei Instanzgrössen, die mehrere zugehörige Datensätze haben, wird der Durchschnitt genommen.

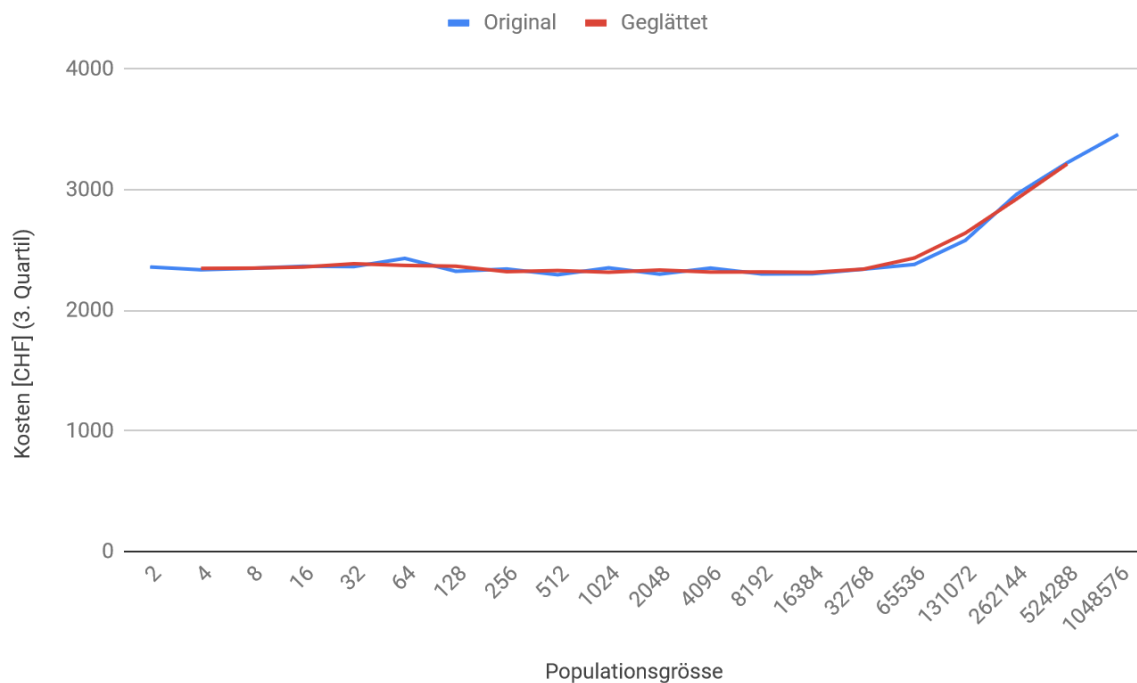


Abbildung 11: Lösungsqualität in Abhängigkeit der Populationsgrösse

Instanzgrösse	Beste Populationsgrösse
10	262144
20	262144
34	262144
50	131072
100	16384
200	16
1000	16

Tabelle 8: Beste gefundene Populationsgrösse pro Instanzgrösse

Die Werte aus Tabelle 8 mögen für dieses Projekt genügen, jedoch soll für beliebige Instanzgrößen eine Populationsgröße berechnet werden können. Mithilfe der Funktion  $f$  (55) kann die Populationsgröße  $m$  für eine beliebige Problemgröße  $n$  berechnet werden. In Abbildung 12 wird anhand der Daten aus Tabelle 8 der grafische Verlauf der Funktion dargestellt. Der Bereich zwischen 34 und 50 wird linear interpoliert. Die Punkte bei 50, 100 und 200 werden mit einer Exponentialfunktion verbunden, damit die Populationsgröße monoton fällt (55).

$$f(n) = \begin{cases} 262144 & \text{für } n \leq 34 \\ -8192n + 540672 & \text{für } 34 < n \leq 50 \\ -250.92 + 1036717e^{-0.0413231x} & \text{für } 50 < n < 200 \\ 16 & \text{sonst} \end{cases} \quad (55)$$

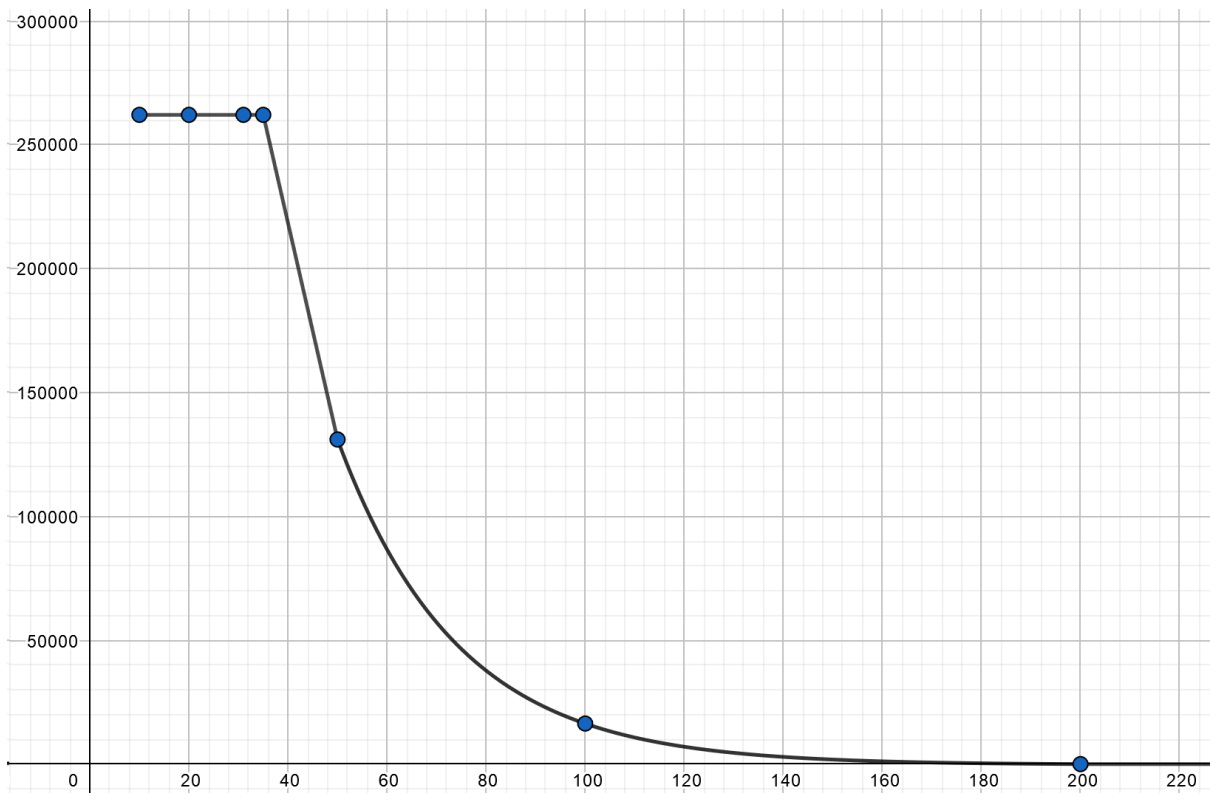


Abbildung 12: Graphischer Verlauf der Funktion  $f$  (55) In der X-Achse ist die Instanzgröße und in der Y-Achse die Populationsgröße abgebildet.

## 3.4 LocalSolver

LocalSolver ist ein Produkt, zur Lösung mathematischer Optimierungsprobleme. Im Hintergrund wird ein Local Search Algorithmus verwendet. Die Hauptprinzipien davon sind «multithreaded adaptive simulated annealing», «autonomous moves» und hoch optimierte inkrementelle Evaluation. Genauere Informationen über Merkmale und Funktionsweise sind in [22] sowie [23] beschrieben [24].

Obwohl bei LocalSolver Einschränkungen gemacht werden können, sollen harte Einschränkungen so oft wie möglich vermieden werden, da dabei die Lösbarkeit eingeschränkt wird. Im Idealfall werden harte Einschränkungen durch weiche Einschränkungen ersetzt [25].

Die Probleme werden bei LocalSolver in einer eigenen Modellierungssprache beschrieben. Es gibt jedoch auch eine objektorientierte Schnittstelle zu diversen Programmiersprachen wie beispielsweise Python, C++, .NET und Java [26].

Die Formulierung des Modells wird in drei Phasen aufgeteilt: In der ersten Phase wird nur das VRP gemäss unserer Zielfunktion gelöst. In der zweiten Phase wird das VRPTW gelöst; allerdings gilt, dass die Wegzeiten zwischen den Besuchen fest sind und nicht verlängert werden dürfen. In der dritten Phase ist eine Verlängerung der Wegzeiten erlaubt. Erst durch mögliche Verlängerungen werden alle optimalen Lösungen möglich.

Das Resultat der vorausgegangenen Phase wird jeweils als Startpunkt der nächsten Phase verwendet. Die erste Phase wird durch gleichmässiges Aufteilen der Besuche auf die Chläuse initialisiert. Das Problem wird in die verschiedenen Phasen aufgeteilt, da ein Grossteil der Kostenfunktion vom VRP abhängig ist. Das VRP ist vergleichsmässig einfach zu lösen und bietet einen guten Startpunkt für das VRPTW. Die Aufteilung der Phasen zwei und drei begründet sich dadurch, dass verlängerte Wege eine grosse Performanceauswirkung auf das Modell haben, jedoch nur in seltenen Fällen einen messbaren Nutzen bringen.

Das gesamte Zeitlimit muss auf die drei Phasen verteilt werden. Die zur Verfügung stehende Rechenzeit wird prozentual verteilt, dadurch gibt es drei Hyperparameter, welche zu optimieren sind. Es müssen jedoch nur zwei optimiert werden, da sich der dritte Parameter trivial aus  $P_3 = 1 - (P_1 + P_2)$  ergibt.

Um die Hyperparameter zu optimieren, wird eine Rastersuche verwendet. Es werden zehn Datensätze mit 15 Besuchen, zwei Chläusen und zwei Tagen generiert. Vier Besuche haben am ersten Tag Wunschzeit und vier am zweiten Tag. Zudem haben zwei Besuche am ersten Tag Nichtverfügbarkeiten und vier Besuche am zweiten Tag. Dies entspricht skaliert dem, was in den Evaluationsdatensätzen aus Kapitel 2 geprüft wird. Das Zeitlimit pro Datensatz beträgt 60 Sekunden.

Die Rastersuche wird zweimal ausgeführt. Die Kosten werden pro Datensatz durch das jeweils beste Resultat des Datensatzes dividiert. Pro Datensatz wird der Durchschnitt der zwei Läufe genommen. Danach wird der Durchschnitt aller Datensätze genommen. Zur besseren Darstellung wurden die Werte noch mit der Funktion  $f(x) = 100 \times (x - 1)$  transformiert.

		Zeitanteil Phase 1										
		0.0%	10.0%	20.0%	30.0%	40.0%	50.0%	60.0%	70.0%	80.0%	90.0%	100.0%
Zeitanteil Phase 2	0%	2.110	1.265	0.698	0.698	0.698	0.698	0.698	0.409	0.409	0.409	6.437
	10%	3.218	1.397	0.698	0.698	0.698	0.698	0.698	0.409	0.409	0.447	
	20%	2.897	0.716	0.698	0.698	0.698	0.698	0.698	0.409	0.447		
	30%	2.731	0.716	0.698	0.698	0.698	0.698	0.698	0.447			
	40%	2.470	0.716	0.698	0.698	0.698	0.698	0.717				
	50%	2.092	0.716	0.698	0.698	0.698	0.717					
	60%	1.226	0.716	0.698	0.698	0.717						
	70%	1.158	0.427	0.698	0.632							
	80%	1.158	0.378	0.379								
	90%	1.158	0.378									
100%	1.158											

Tabelle 9: LocalSolver Rastersuche für Zeitlimiten

Das Minimum der Rastersuche liegt bei den Konfigurationen (10%, 80%, 10%) sowie (10%, 90%, 0%). Die Zahlen in Klammern entsprechen dem Zeitanteil der einzelnen Phasen, wobei die erste Zahl dem Zeitanteil für die erste Phase entspricht. Die zweite Zahl entspricht dem Wert für die zweite Phase und die dritte Zahl bestimmt demnach die Dauer der dritten Phase. Das Resultat zeigt, dass es zwei mögliche Regionen für gute Konfigurationen gibt. Dies ist dadurch zu erklären, dass einige Probleme ihr Optimum nahe beim VRP haben und andere weniger. Die Wahl ist auf die Konfiguration (10%, 80%, 10%) gefallen, da mindestens 10% für die dritte Phase verwenden werden soll, um keine optimale Lösung auszuschliessen.

Aus dem Vergleich der Lösung mit den Parametern (0%, 0%, 100%) (Resultat 2.110) und (0%, 100%, 0%) (Resultat 1.158) wird ersichtlich, dass die dritte Phase viel rechenaufwändiger ist als die zweite. Falls nur die dritte Phase verwendet wird, sind die Kosten fast doppelt so hoch.

In den folgenden Modellen wird die Variable  $V$  als Menge aller Besuche verwendet. Die Anzahl der Besuche wird mit  $n_v = |V|$  bezeichnet und ein einzelner Besuch mit  $v_i \in 1, \dots, n_v$ .

Nach demselben Prinzip wird die Variable  $S$  als Menge aller Chläuse verwendet,  $n_s$  ist die Anzahl der Chläuse und  $s_i \in 1, \dots, n_s$  als einzelner Chlaus  $i$ . Um mehrere Tage abbilden zu können, werden die ursprünglichen Chläuse mit den Anzahl Tagen  $n_p$  multipliziert. So ist bei einem Beispiel mit fünf Chläusen und zwei Arbeitstagen der Chlaus  $s_6$ , der erste Chlaus am Arbeitstag zwei. Damit jedes Problem optimal gelöst werden kann, können zusätzliche Chläuse erstellt werden. Maximal wird die Differenz zwischen der Anzahl der Besuche und der Anzahl der Chläuse erstellt. Chläuse, welche zusätzlich hinzugefügt wurden, sind in der Menge  $AS \subseteq S$  enthalten. Weiter wird die Variable  $n_{as} = |AS|$  verwendet.

Pausen werden sehr ähnlich wie Besuche behandelt, weshalb sie ebenfalls in der Menge  $V$  enthalten sind. Die Pausen sind in der Menge  $B \subseteq V$  separat zusammengefasst. Ist eine Pause für einen Chlaus vorgesehen, so ist diese  $n_p$ -mal in der Menge  $B$  enthalten. Die Pause  $b_s \in B$  ist die Pause, welche für den Chlaus  $s$  vorgesehen ist. Die Distanzen zwischen den Besuchen sind in einer Matrix  $dist$  eingetragen, wobei  $dist_{0,i}$  die Distanz vom Startpunkt zum Besuch  $i$  ist. Analog dazu ist  $dist_{i,0}$ , der Weg von Besuch  $i$  zum Startpunkt.

Wenn aus dem Kontext heraus klar ist, was gemeint ist, so wird die Notation  $\forall s$  stellvertretend für  $\forall s \in S$  verwendet.

Die verwendeten Variablen der ersten Phase sind in Tabelle 10 beschrieben.

Variablendefinition		Beschreibung
$visitSeq_s \subseteq V$	$\forall s$	Menge, welche alle besuchten Familien für Chlaus $s$ enthält
$lastVisit_s = visitSeq_s$	$\forall s$	Kurzschreibweise für den letzten Besuch für Chlaus $s$ , um die Formulierung kompakter zu halten
$santaUsed_s \in 0,1$	$\forall s$	Nimmt den Wert 1 an, falls der Chlaus $s$ mindestens eine Familie besucht, ansonsten 0
$SantaWorkingTime_s \in \mathbb{R}$	$\forall s$	Die Arbeitsdauer für Chlaus $s$
$SantaWalkingTime_s \in \mathbb{R}$	$\forall s$	Laufzeit für Chlaus $s$
$SantaVisitTime_s \in \mathbb{R}$	$\forall s$	Besuchszeit für Chlaus $s$

Tabelle 10: Variablendefinition erste Phase für LocalSolver

Die Formulierung für die erste Phase lautet:

$$\begin{aligned}
 & \text{minimize} \\
 & \frac{1}{120} \max_{i \in S} \text{SantaWorkingTime}_i \\
 & + \frac{1}{90} \sum_{s=1}^{n_s} \text{SantaWorkingTime}_s \\
 & + \frac{1}{90} \sum_{a=1}^{n_{as}} \text{SantaWorkingTime}_a \\
 & + 400 \sum_{a=1}^{n_{as}} \text{SantaUsed}_a
 \end{aligned} \tag{56}$$

$$\begin{aligned}
 & \text{s.t.} \\
 & \bigcup_{s=1}^{n_s+1} \text{visitSeq}_s = V
 \end{aligned} \tag{57}$$

$$v \notin \text{visitSeq}_{n_s+1} \quad \forall v : v \notin B \tag{58}$$

$$\text{santaUsed}_s = (|\text{visitSeq}_s| > 0) \quad \forall s \tag{59}$$

$$\text{santaUsed}_s = 1 \rightarrow b_s \in \text{visitSeq}_s \quad \forall s \tag{60}$$

$$\text{santaUsed}_s = 0 \rightarrow b_s \in \text{visitSeq}_{n_s+1} \quad \forall s \in 1, \dots, n_s \tag{61}$$

$$\text{distSelector}(s, i) = \begin{cases} \text{dist}_{0, \text{visitSeq}_{s_i}} & \text{für } i = 1 \\ \text{dist}_{\text{visitSeq}_{s(i-1)}, \text{visitSeq}_{s_i}} & \text{sonst} \end{cases} \tag{62}$$

$$\text{SantaWalkingTime}_s = \text{dist}_{\text{lastVisit}_s, 0} \sum_{i=1}^{|\text{visitSeq}_s|} \text{distSelector}(s, i) \quad \forall s \tag{63}$$

$$\text{SantaVisitTime}_s = \sum_{i=1}^{|\text{visitSeq}_s|} \text{duration}_{\text{visitSeq}_{s_i}} \quad \forall s \tag{64}$$

$$\begin{aligned}
 & \text{SantaWorkingTime}_s \\
 & = \text{SantaWalkingTime}_s + \text{SantaVisitTime}_s
 \end{aligned} \quad \forall s \tag{65}$$

$$\text{SantaWorkingTime}_s \leq \text{dayDuration}_s \quad \forall s \tag{66}$$

Die Komponenten der Zielfunktion sind von der Problemstellung übernommen. Wunschzeiten und Nichtverfügbarkeiten werden in dieser Phase noch nicht implementiert und sind deshalb nicht Teil der Zielfunktion (56).

Es gilt die Einschränkung, dass alle *visitSeq* eine Partition aller Besuche sind (57). Nicht benötigte Pausen werden in die Menge *visitSeq*<sub>n<sub>s</sub>+1</sub> verschoben, damit die Partitionsbedingung nicht verletzt wird. Familienbesuche dürfen nicht in dieser letzten Partition enthalten sein (58).

Pausen dürfen nur von dem dazugehörigen Chlaus eingelegt werden. Mit der Hilfsvariable  $santaUsed_s$  (59) lässt sich die Bedingung, wonach die Pause für Chlaus  $s$  in der Menge  $visitSequence_s$  vorhanden sein muss, falls der Chlaus benötigt wird, formulieren (60). Wird der Chlaus  $s$  nicht benötigt, so muss die Pause in der Menge  $visitSeq_{n_s+1}$  sein (61).

Um die benötigte Wegzeit eines Chlauses zu berechnen, ist die Funktion  $distSelector(s, i)$  definiert. Diese Funktion gibt die Distanz für den Chlaus  $s$  zwischen dem Besuch  $i - 1$  und dem Besuch an Stelle  $i$  zurück (62). Als Beispiel gibt  $distSelector(1, 4)$  die Distanz zwischen dem dritten und vierten Besuch für den ersten Chlaus zurück.

Mithilfe dieser Funktion kann nun die Summe  $SantaWalkingTime_s$  gebildet werden, welche die gesamte Wegzeit eines Chlauses umfasst (63). Die Arbeitszeit eines Chlauses beinhaltet auch die Besuchsdauer, welche durch die Summe aller Besuchszeiten des Chlauses gebildet werden kann (64). Die gesamte Arbeitszeit  $santaWorkingTime_s$  eines Chlauses  $s$  wird als Summe der Besuchs- und Laufzeit definiert (65). Zusätzlich gilt die Einschränkung, dass diese Arbeitszeit innerhalb des Arbeitstages geleistet werden muss (66).

Das optimierte Resultat dieses Modells wird als Startpunkt für die zweite Phase verwendet. In der zweiten Phase werden zusätzlich die Zeitfenster beachtet. Hierfür werden zusätzlich die in der Tabelle 11 aufgeführten Variablen benötigt.

Variablendefinition	Beschreibung
$waitBeforeStart_s \in \mathbb{R} \quad \forall s$	Die Zeitdauer, wie lange der Chlaus $s$ wartet bis er mit seiner Route beginnt
$overtime_s \in \mathbb{R} \quad \forall s$	Zeit, welche der Chlaus nach seiner Arbeitszeit verbringt

Tabelle 11: Variablendefinition zweite Phase für LocalSolver

Das Modell der zweiten Phase erweitert die Formulierung der ersten Phase. Dabei wird die Zielfunktion ersetzt, wogegen alle Einschränkungen übernommen werden.

minimize

$$\begin{aligned}
& \frac{1}{30} \sum_{s=1}^{n_s} \text{overtime}_s \\
& + \frac{1}{30} \sum_{s=1}^{n_s} \sum_{i=1}^{|\text{visitSeq}_s|} \text{visitUnavailableDuration}(s, i) \\
& + \frac{1}{90} \sum_{s=1}^{n_s} \text{SantaWorkingTime}_s \\
& + \frac{1}{120} \max_{i \in S} \text{SantaWorkingTime}_i \\
& - \frac{1}{180} \sum_{s=1}^{n_s} \sum_{i=1}^{|\text{visitSeq}_s|} \text{visitDesiredDuration}(s, i) \\
& + \frac{1}{90} \sum_{a=1}^{n_{as}} \text{SantaWorkingTime}_a \\
& + 400 \sum_{a=1}^{n_{as}} \text{santaUsed}_a
\end{aligned} \tag{67}$$

s. t.

$$\begin{aligned}
& \text{visitStartTime}(s, i) \\
& = \begin{cases} \text{dayStart}_s + \text{waitBeforeStart}_s + \text{distSelector}(s, i) & \text{für } i = 1 \\ \text{visitStartTime}(s, i-1) + \text{duration}_{\text{visitSeq}_{s_{i-1}}} + \text{distSelector}(s, i) & \text{für } i \neq 1 \end{cases}
\end{aligned} \tag{68}$$

$$\text{visitEndTime}(s, i) = \text{visitStartTime}(s, i) + \text{duration}_i \tag{69}$$

$$\begin{aligned}
& \text{visitDesiredDuration}(s, v) \\
& = \sum_{i=1}^{|d_v|} \begin{cases} 0 & \text{für } d_{v,i,1} < \text{visitStartTime}(s, i) \vee d_{v,s,0} > \text{visitEndTime}(s, i) \\ \min\{d_{v,s,1}, \text{visitEndTime}(s, i)\} - \max\{d_{v,s,0}, \text{visitStartTime}(s, i)\} & \text{sonst} \end{cases}
\end{aligned} \tag{70}$$

$$\begin{aligned}
& \text{visitUnavailableDuration}(s, v) \\
& = \sum_{i=1}^{|u_v|} \begin{cases} 0 & \text{für } u_{v,i,1} < \text{visitStartTime}(s, i) \vee u_{v,s,0} > \text{visitEndTime}(s, i) \\ \min\{u_{v,s,1}, \text{visitEndTime}(s, i)\} - \max\{u_{v,s,0}, \text{visitStartTime}(s, i)\} & \text{sonst} \end{cases}
\end{aligned} \tag{71}$$

$$\text{visitEndTime}(s, |\text{visitSeq}_s|) + \text{dist}_{\text{lastVisit}_s,0} - \text{overtime}_s \leq \text{dayEnd}_s \quad \forall s \tag{72}$$

Die Zielfunktion entspricht in der zweiten Phase vollständig der Problemstellung (67). Damit aus dem VRP das VRPTW wird, müssen die Zeitfenster beachtet werden. Dazu wird die rekursive Funktion  $\text{visitStartTime}(s, i)$  definiert, die den Startzeitpunkt des Chlaus  $s$  an der Stelle  $i$  zurückgibt. Die Abbruchbedingung dieser Funktion ist beim ersten Besuch. Der Chlaus kann entweder gleich zu Tagesbeginn starten oder noch warten; hierfür wird die Variable  $\text{waitBeforeStart}_s$  verwendet (68).

Mit dem Startzeitpunkt der Besuche und der Besuchsdauer kann die Wunschzeit berechnet werden. Zur Vereinfachung der Formulierung wird die Funktion  $\text{visitEndTime}(s, i)$  definiert, die den Endzeitpunkt vom  $i$ -ten Besuch von Chlaus  $s$  zurückgibt (69). Die Wunschzeit wird aus der Überschneidung der Besuchszeit des Chlaus  $s$  und den gewünschten Zeiten des



Besuches gebildet (70). Analog dazu kann die Besuchsdauer in der Nichtverfügbarkeitszeit berechnet werden (71). Dadurch, dass der Chlaus jetzt mitten im Tag starten kann, muss die Einschränkung (66) erweitert werden. Arbeitszeit ausserhalb der Tageszeit wird als Überzeit, Variable  $overtime_s$  für Chlaus  $s$ , bezeichnet (72).

Das optimierte Resultat der zweiten Phase wird direkt als Startwert der dritten Phase übernommen. In der dritten Phase sind auch verlängerte Wegzeiten zugelassen. Dafür wird die Variable  $waitBeforeVisit_{s,i} \in \mathbb{R} \forall s, i = 1..|visitSeq_s|$  benötigt, welche die Dauer angibt, wie lange nach dem Besuch  $i$  gewartet wird, bis der Chlaus  $s$  sich auf den Weg zu dem nächsten Besuch macht. Die Funktion  $visitStartTime(s, i)$  aus Formel (68) wird modifiziert, um diese Variable zu benutzen. (73).

$$\begin{aligned}
 & visitStartTime(s, i) \\
 & = \begin{cases} dayStart_s + waitBeforeStart_s + distSelector(s, i) & \text{für } i = 1 \\ visitStartTime(s, i - 1) + duration_{visitSeq_{s,i-1}} + distSelector(s, i) + waitBeforeVisit_{s,i} & \text{für } i \neq 1 \end{cases} \quad (73)
 \end{aligned}$$

Die Zielfunktion der dritten Phase sowie die Einschränkungen und Definitionen werden vollständig von der zweiten Phase übernommen.

Das optimierte Resultat der dritten Phase wird als Resultat dieses Lösungsansatzes verwendet.

## 3.5 Google OR-Tools Routing

Der Ansatz Google OR-Tools Routing (Google Routing) basiert auf dem gleichnamigen Open-Source Softwarepaket. Google OR-Tools kann für verschiedene Problemstellungen benutzt werden, dazu gehören beispielsweise Flow-Probleme oder ganzzahlige lineare Programmierung. Der Google Routing Ansatz basiert nur auf der Routing Bibliothek, die wiederum auf Constraint-Programmierung und verschiedenen Heuristiken basiert. Die Arbeit beim Google Routing besteht darin, das Problem zu formulieren, das anschliessend von Google OR-Tools optimiert wird [27].

Vor der Formulierung werden zuerst noch die verwendeten Variablen definiert. Die Anzahl der Besuche entspricht  $n_v$ . In den Besuchen  $v_i \in V$  sind auch die Pausen  $b_s \in B$  enthalten, das heisst  $B \subseteq V$ . Der Start- und Endpunkt ist  $v_0$ . Die Wegdauer, um von Besuch  $i$  zu Besuch  $j$  zu kommen, wird als  $w_{i,j}$  definiert. Der Weg vom Startpunkt zum ersten Besuch ist also  $w_{0,1}$ . Die Variable  $l_v$  bezeichnet die Dauer eines Besuchs  $v$ . Der Startpunkt hat eine Besuchsdauer von null, das heisst es gilt  $l_0 = 0$ . In der Variable  $d_v$  sind die Wunschzeiten des Besuchs  $v$  enthalten. Ein Besuch kann über mehrere Wunschzeiten verfügen. So entspricht  $d_{v,x}$  der Wunschzeit  $x$  des Besuchs  $v$ . Eine einzelne Wunschzeit besteht wiederum aus einem Start und einem Ende. Der Start der Wunschzeit  $x$  des Besuchs  $v$  entspricht  $d_{v,x,0}$ , das dazugehörige Ende ist  $d_{v,x,1}$ . Analog zu den Wunschzeiten steht  $u_{v,x}$  für die Nichtverfügbarkeit  $x$  vom Besuch  $v$ .

Bei der Routing Bibliothek [28] ist das Problem in Dimensionen aufgeteilt. Wobei jeder Besuch pro Dimension einen Wert hat. Ein Besuch kann hierbei eine besuchte Familie, eine Pause oder der Startpunkt sein. Die erste Dimension, ist die Dimension Zeit. In der Dimension Zeit ist jedem Besuch ein ganzzahliger Wert zugeordnet. In dieser Formulierung entspricht der Wert in der Zeitdimension dem Startzeitpunkt des Besuchs. Die Werte einer Dimension werden dabei aufsummiert. Aufsummieren heisst in diesem Fall, dass der Wert jeweils zum vorherigen Besuch addiert wird. Der Betrag, um den sich die Summe erhöht, wird mittels Callback berechnet. Der Callback für die Zeitdimension benötigt nur die Information, welcher Besuch  $j$  abgestattet wird und welches der letzte Besuch  $i$  war. Da sich die Zeitdimension lediglich mit Startzeitpunkten befasst, muss der Callback auch die Besuchszeiten dazurechnen. Konkret kann der Zeit-Callback als Funktion dargestellt werden (74).

$$\text{timeCallback}(i,j) = l_i + w_{i,j} \quad (74)$$

Bei einer Dimension kann eingestellt werden, ob der Wert des ersten Besuchs auf null fixiert werden soll. Der erste Besuch muss nicht am frühestmöglichen Zeitpunkt stattfinden, also muss der Wert nicht auf null gesetzt werden. Des Weiteren kann ein Schlupf definiert werden. Dieser Schlupf definiert einen maximalen Betrag, um den der Wert vergrössert werden darf. Ein Schlupf von null bedeutet also, dass die Wegzeiten exakt addiert werden müssen und nicht künstlich erhöht werden dürfen. Da die Wege zwischen den Besuchen nicht künstlich verlängert werden sollen, wird der Schlupf auf null gesetzt. Zum Schluss kann der maximale Wert einer

Dimension begrenzt werden, diese Begrenzung wird auf das Ende des letzten Tages gesetzt, da später keine Besuche mehr gemacht werden sollen.

Als Zweites wird eine Dimension Dauer erstellt, diese misst die Dauer der Routen. Bei dieser Dimension wird der Startwert fest auf null gesetzt, denn die Route beginnt beim Startpunkt mit einer Dauer von null. Die Dimension Dauer benutzt ebenfalls den Zeit-Callback (74) und verbietet verlängerte Wege.

Als Letztes werden noch Dimensionen für die Zuordnung der Pausen eingeführt. Um einem Chlaus Pausen fest zuordnen zu können, müssen diese eine Ressource verbrauchen, die nur dem dazugehörigen Chlaus zur Verfügung steht. Solche Ressourcen werden mithilfe von Dimensionen abgebildet. Als Beispiel hat der Chlaus 1 eine Pause eingetragen. Es wird also eine Chlaus1-Pause-Dimension erstellt, die misst, wie viele Pausen des ersten Chlaus in dieser Route liegen. Der Callback (75) prüft also, ob ein Besuch die Pause von Chlaus 1 ist, und gibt in diesem Fall die Zahl 1 zurück.

$$santaBreakCallback_1 = \begin{cases} 1 & \text{für } v_i = b_1 \\ 0 & \text{sonst} \end{cases} \quad (75)$$

Für jeden Chlaus, dem eine Pause zugeordnet ist, muss eine solche Pausen-Dimension erstellt werden. Die Pausen-Dimensionen haben alle einen Startwert von null und einen Schlupf von null. Der maximale Wert der Pausen-Dimension entspricht dem Wert eins, da jeder Chlaus maximal eine Pause einlegen darf. Sobald diese Pausen-Dimensionen erstellt wurden, kann für jede einzelne Pausen-Dimension und jeden Chlaus eine maximale Kapazität festgesetzt werden. Da diese maximale Kapazität pro Chlaus optional ist, kann das jedoch vereinfacht werden. Damit Chlaus 2 keine Pausen von Chlaus 1 macht, wird die maximale Kapazität von Chlaus 2 in der Pausen-Dimension 1 auf null gesetzt. Würde jetzt Chlaus 2 die Pause von Chlaus 1 machen, hätte Chlaus 2 einen Wert grösser null in der Pausen-Dimension 1, was nicht zugelassen ist. Für Chlaus 1 wird in der dazugehörigen Pausen-Dimension 1 kein Maximum gesetzt, somit darf Chlaus 1 seine eigenen Pausen machen.

Der Zeitraum, in dem die Besuche beim Google Routing gemacht werden müssen, ist grundsätzlich zusammenhängend und startet beim Zeitpunkt null. Jedoch geht die Problemstellung von mehreren, disjunkten Tagen aus. Um das umzusetzen, wird die Dauer des Zeitraums auf die Differenz zwischen dem Start des ersten Tages und dem Ende des letzten Tages gesetzt. Jedoch sind so immer noch Besuche zwischen zwei Tagen möglich. Um dieses Problem zu lösen, muss jeder Chlaus für jeden weiteren Tag einmal dupliziert werden. So kann jeder Chlaus einem Tag zugeordnet werden. Bei einem Beispiel mit 3 Chläusen und 2 Tagen, wären es insgesamt 6 Chläuse. Die ersten drei Chläuse werden dem ersten Tag zugeordnet. Die letzten drei Chläuse werden dem zweiten Tag zugeordnet. Eine solche Zuordnung geschieht über die Zeitdimension. Damit kann pro Chlaus ein Bereich festgelegt werden, der die Werte der Dimension einschränkt. Dabei gilt es zu beachten, dass die Besuche einem Chlaus zugeordnet werden und die Einschränkung nur für die Besuche des jeweiligen Chlaus gilt.

Bevor es um die Wunschzeiten und die Nichtverfügbarkeiten geht, soll zuerst das Optimierungsziel betrachtet werden. Beim Routing von Google OR-Tools kann nicht direkt eine Zielfunktion definiert werden, so wie es beispielsweise beim genetischen Algorithmus der Fall ist. Stattdessen können einzelne Kosten definiert werden. Ein erster Kostenanteil ist der längste Arbeitstag, welcher mit 30 CHF pro Stunde bestraft wird. Um diesen Kostenanteil abzubilden kommt die Dimension Dauer ins Spiel. Wie oben beschrieben, misst die Dimension die Dauer einer einzelnen Route. Es wird ein Kosten-Koeffizient von 30 CHF pro Stunde zum globalen Abstand dieser Dimension gesetzt. Dieser globale Abstand ist die maximale Differenz zwischen Start und Ende einer Route. Da eine Route immer mit einer Dauer von null startet, wird der Wert beim Endpunkt übernommen, also nach wie viel Zeit der Chlaus wieder zurück ist. Als nächstes entstehen Kosten für die Arbeitszeit der Chläuse. Diese Kosten werden über Callbacks definiert, die einem Chlaus zugewiesen sind. Da die Kosten von der Zeit abhängig sind, wird im Kosten-Callback (76) wiederum ein Zeit-Callback verwendet.

$$costCallback(i, j) = 40timeCallback(i, j) \quad (76)$$

Mithilfe dieses Kosten-Callbacks können auch die Kosten der zusätzlichen Chläuse berechnet werden. Um die einmaligen Kosten von 400 CHF für den Einsatz eines zusätzlichen Chlaus zu berechnen, wird dieser Betrag beim Startpunkt zusätzlich addiert. Die Kosten zwischen anderen Besuchen berechnen sich wie bei den vorgesehenen Chläusen, nur mit einem anderen Stundensatz. Beim Kosten-Callback müssen die 400 CHF noch mit 3600 multipliziert werden, da als Einheit überall Sekunden verwendet wird (77).

$$costAdditionalCallback(i, j) = \begin{cases} 400 \cdot 3600 + (40 + 40)timeCallback(i, j) & \text{für } i = 0 \\ (40 + 40)timeCallback(i, j) & \text{sonst} \end{cases} \quad (77)$$

Als nächstes wird noch definiert, welche Kosten entstehen, wenn ein Besuch nicht gemacht wird. Dieser Fall tritt ein, wenn das Google Routing keine Lösung findet. Dann werden nämlich keine Besuche erstattet. Die Kosten eines fehlenden Besuchs können mit einer Disjunktion gesetzt werden. Gemäss der Zielfunktion entstehen dabei Kosten von 560 CHF pro fehlendem Besuch. Dieser Betrag muss erneut mit 3600 multipliziert werden. Hier ist noch zu erwähnen, dass diese Kosten auch entstehen, wenn eine Pause nicht gemacht wird. Das heisst, wenn ein Chlaus eine Route hat, die ausschliesslich aus einer Pause besteht, wird die Pause trotzdem gemacht. Da diese Route nur unnötige Kosten verursacht, wird diese nach dem Lösungsprozess gelöscht.

Als letzter Teil der Formulierung wird die Umsetzung der Wunschzeiten und Nichtverfügbarkeiten beschrieben. Für die Zeitfenster, also Wunschzeiten und Nichtverfügbarkeiten, gibt es vier mögliche Strategien.

Die erste und einfachste Strategie (SN) besteht darin, die Zeitfenster zu ignorieren. Bei dieser Strategie wird also lediglich ein VRP mit mehreren Tagen gelöst.

Die zweite Strategie (SU) beschränkt sich auf die Nichtverfügbarkeiten. Die Besuche werden anhand der Nichtverfügbarkeiten, ähnlich wie bei den Tagen beschränkt. Genauer gesagt, wird für jede Nichtverfügbarkeit eines Besuchs ein Bereich in der Zeitdimension des Besuchs verboten. Das sorgt dafür, dass der Besuch innerhalb der Nichtverfügbarkeit nicht durchgeführt werden darf.

Die dritte Strategie (SDH) ist eine Abwandlung der zweiten Strategie, jedoch werden Wunschzeiten berücksichtigt. Für Besuche, die keine Wunschzeit haben, wird ausschliesslich die zweite Strategie verwendet. Das heisst, Besuche innerhalb der Nichtverfügbarkeit werden verboten. Für Besuche, die eine Wunschzeit haben, wird die mögliche Besuchszeit so beschränkt, dass der Besuch in der Wunschzeit stattfinden muss. Das kann jedoch nur für eine Wunschzeit eingeschränkt werden. Deshalb wird vorgängig die beste Wunschzeit berechnet. Wobei das alleinige Mass für die Güte einer Wunschzeit deren Dauer ist. Die mögliche Besuchszeit wird also beschränkt, sodass der Besuch in der längsten Wunschzeit stattfinden muss. Diese Beschränkung hat den Effekt, dass ein Besuch, der nicht in der Wunschzeit stattfinden kann, nicht besucht wird.

Die vierte Strategie (SDS) baut auf der zweiten Strategie auf und erweitert diese um die Wunschzeiten. Die Wunschzeiten werden mittels einer weichen Einschränkung gesetzt. Auch hier kann nur eine einzige Wunschzeit verwendet werden. Die Auswahl der Wunschzeit aus mehreren Wunschzeiten erfolgt gemäss der dritten Strategie. Die eigentliche Erweiterung besteht darin, dass eine obere und eine untere Beschränkung der Zeitdimension bei allen Besuchen mit Wunschzeiten gesetzt wird. Sollte eine der beiden Beschränkungen verletzt werden, kommen Mehrkosten hinzu. Diese Kosten entsprechen dem maximalen Bonus, der durch das Einhalten der Wunschzeiten hätte erzielt werden können.

Um zu entscheiden, welche dieser vier Strategien eingesetzt werden sollen, wurden alle 12 Datensätze mit jeder Strategie optimiert. Diese Tests wurden auf dem Server gemacht, der auch für die Evaluation in Kapitel 4.1 verwendet wird. So wird sichergestellt, dass die Rechenleistung dieselbe ist, wie bei der Evaluation. Die Resultate dieses Tests sind in der Tabelle 12 ersichtlich. Dabei wurden jeweils zwei Strategien parallel berechnet, um die zwei Kerne des Servers optimal zu nutzen.

Datensatz	SN	SU	SDH	SDS
10V	560	560	560	560
10VD	527	527	480	452
10VU	878	532	532	532
20V	912	912	912	912
20VBD	851	851	685	681
20VU	1121	855	855	855
34VDU	1148	732	645	645
34VBDU	1215	690	601	601
50VBDU	2283	1723	22400	1442
100VBDU	4369	44800	44800	44800
200VBDU	7693	89600	89600	89600
1000VBDU	36820	448000	448000	448000

Tabelle 12: Kosten in CHF der Lösungen pro Strategie

Entsprechend der Tabelle 12 werden bei einem Rechner mit zwei Prozessorkernen bis Datensatz 50VBDU die Strategien SDH und SDS verwendet. Bei grösseren Probleminstanzen kommen die Strategien SN und SU zum Einsatz. Generell wird diese Unterscheidung der Strategien anhand der Anzahl Besuche gemacht. So sind grössere Probleminstanzen als Probleme mit über 50 Besuchen definiert. Sollten jedoch mehr als zwei Prozessorkerne verfügbar sein, werden bis zu vier Formulierungen gleichzeitig berechnet, die sich lediglich in der Strategie unterscheiden. Allgemein wird jedoch maximal eine Strategie pro verfügbarem Prozessorkern gleichzeitig berechnet.

## 4 Evaluation

In diesem Kapitel werden die beschriebenen Lösungsansätze auf die in Kapitel 2 definierten Datensätze angewendet. Die Resultate werden ausgewertet und diskutiert. Dabei wird auf verschiedene Eigenschaften der Ansätze eingegangen.

### 4.1 Messmethodik

Damit die Messungen reproduzierbar und realitätsnahe sind, müssen die Rahmenbedingungen definiert werden. Alle Messungen fanden auf einem HP ProLiant MicroServer Gen8 (Hersteller Nr: 712318-421) mit einem Intel Pentium G2020T Prozessor und 2GB Arbeitsspeicher statt. Dieser Server wurde vom Kunden zur Verfügung gestellt und wird später für den produktiven Betrieb genutzt.

Evaluiert werden die im Kapitel 2 eingeführten Datensätze. Pro Datensatz wird ein Zeitlimit für die maximale Rechenzeit definiert. Jeder Lösungsansatz wird pro Datensatz fünfmal ausgeführt. Sollte ein Ansatz nicht ausgeführt oder keine Lösung berechnet werden können, so wird die maximale Rechenzeit und eine Lösung ohne Besuche angenommen. Die erhaltenen Kosten werden danach durch die Anzahl Besuche geteilt, um die Datensätze in einem Verhältnis zueinander darzustellen. Die Anzahl der Besuche ist entspricht der Summe aus Familienbesuchen und der Anzahl Pausen multipliziert mit der Anzahl Tage. Weil die Resultate schwanken können, wird das dritte Quartil als Messwert genommen. Diese Messmethode soll sicherstellen, dass die Werte praxisnahe ausfallen. Danach wird ein gewichteter Mittelwert pro Lösungsansatz über alle Datensätze berechnet. Die Gewichte sind so gesetzt, dass die realen Datensätze und der nächste hochskalierte Datensatz 50% des Gewichts ausmachen, die kleineren Datensätze ein Gesamtgewicht von 25% haben und die grossen Beispiele die restlichen 25% ausmachen. Eine Übersicht der Werte findet sich in Tabelle 13.

Datensatz	Zeitlimit	Anzahl Besuche	Gewichtung
10V	10 min	10	1
10VD	10 min	10	1
10VU	10 min	10	1
20V	20 min	20	1
20VBD	20 min	20	1
20VU	20 min	20	1
34VDU	90 min	34	4
34VBDU	90 min	34	4
50VBDU	90 min	50	4
100VBDU	120 min	100	2
200VBDU	120 min	200	2
1000VBDU	120 min	1000	2

Tabelle 13: Gewichtung der Datensätze für die Evaluation

## 4.2 Übersicht

Aus Abbildung 13 geht hervor, dass LocalSolver der beste Ansatz ist, um unsere Probleminstanzen in der vorgegebenen Rechenzeit zu lösen. LocalSolver hat zusätzlich bei jedem Datensatz den besten Wert erreicht. Wie deutlich erkennbar ist, stellen der GA und das Google Routing die einzige Konkurrenz zum Gewinner dar. Alle drei ILP-Ansätze weisen grosse Probleme mit der Skalierbarkeit auf und schneiden in der Gesamtwertung dementsprechend schlecht ab.

Wie in Abbildung 14 ersichtlich, ist ILP2 der einzige Ansatz, der nicht die vollen 60 Stunden Rechenzeit braucht. Die einzelnen Werte der Ansätze pro Datensatz sind im Anhang 8.2 abgebildet. Mit den Daten im Anhang 8.2 können alle Abbildungen in diesem Kapitel generiert werden.

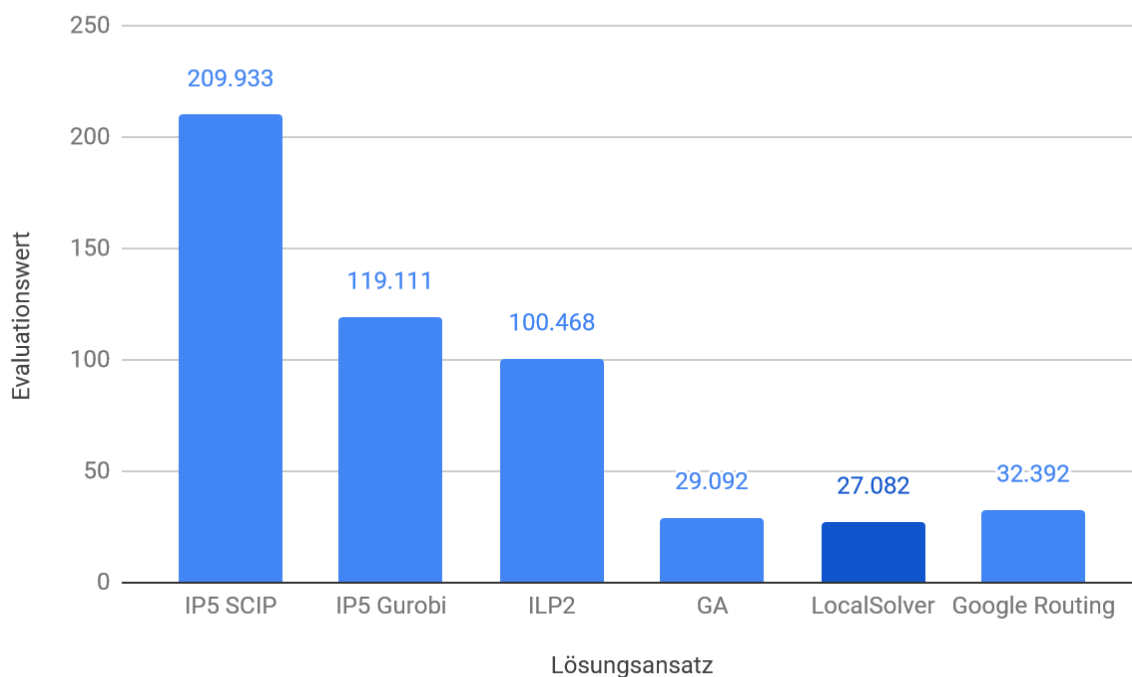


Abbildung 13: Evaluationswert pro Ansatz



## 4.3 Laufzeit

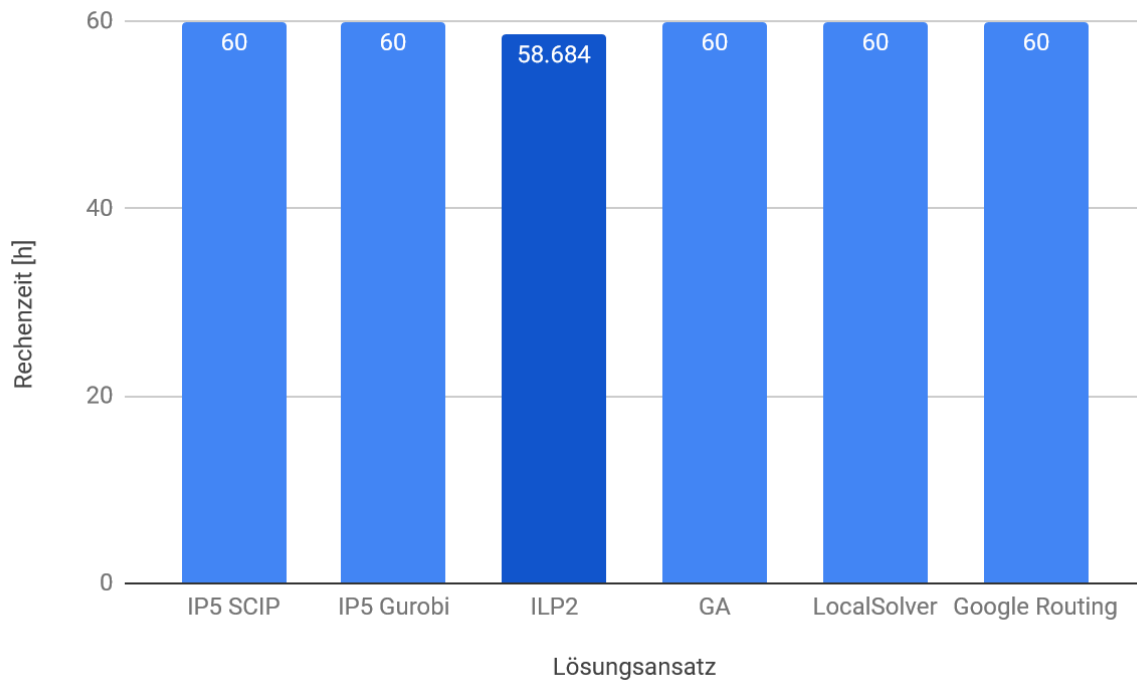


Abbildung 14: Rechenzeit pro Ansatz über alle 12 Datensätze

Wie im vorherigen Kapitel bereits erwähnt wurde, ist ILP2 der einzige Ansatz, der bei einzelnen Datensätzen nicht die maximale Rechenzeit braucht. Der Grund dafür ist, dass bei kleinen Datensätzen die gefundene Lösung der bestmöglichen Lösung entspricht und der Lösungsvorgang beendet wird. Dieser Beweis kann innerhalb der vorgegebenen Rechenzeit nur bei den Datensätzen 10V und 10VU erbracht werden. Bei den anderen Datensätzen wird die ganze Rechenzeit gebraucht, da entweder die gefundene Lösung nicht optimal ist oder die Lösung nicht bewiesen optimal ist.

Alle anderen Ansätze terminieren erst beim Erreichen des Zeitlimits. Die Gründe dafür sind entweder, dass die Lösung nicht bewiesen optimal ist oder dass abgesehen vom Zeitlimit kein anderes Abbruchkriterium definiert ist.

## 4.4 Skalierbarkeit

Wie in Abbildung 15 zu erkennen, variiert die Skalierbarkeit der Ansätze stark. Bei einem Ansatz, der auch beim grössten Datensatz noch skaliert, sollte die Linie keine Knicke nach oben ausweisen. Denn diese Knicke bedeuten, dass die Kosten überproportional zunehmen, was auf eine schlechte Skalierbarkeit hindeutet.

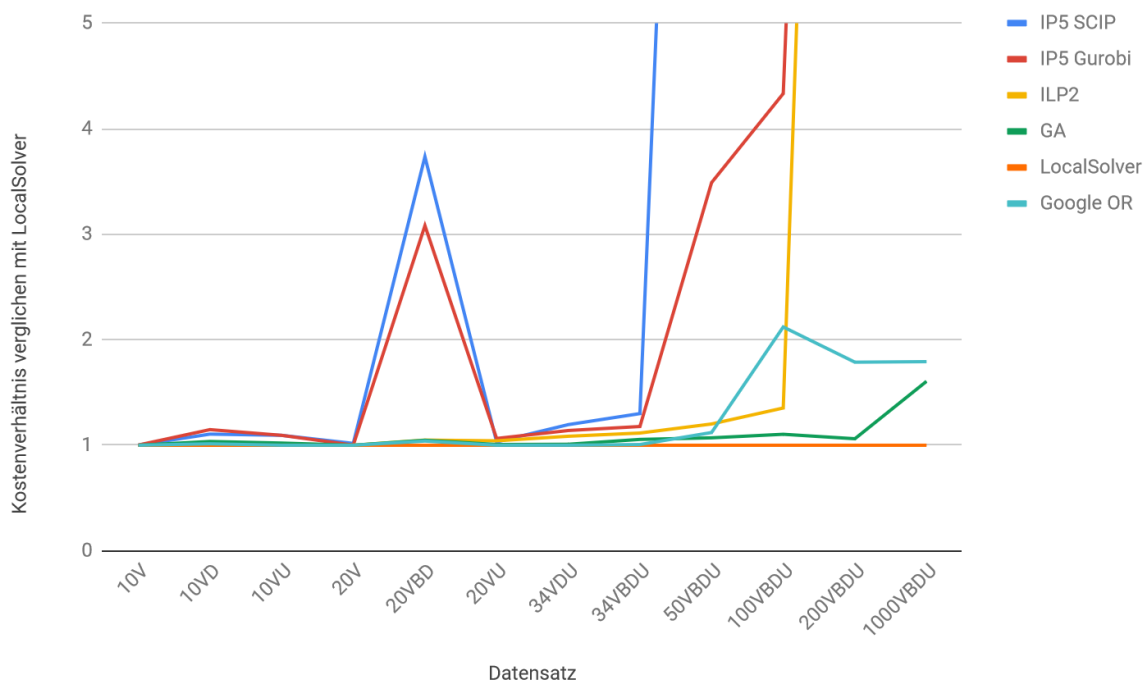


Abbildung 15: Kostenverhältnis zu den besten bekannten Kosten pro Datensatz

Die Ansätze IP5 SCIP und IP5 Gurobi skalieren allgemein am schlechtesten und bilden hinsichtlich Skalierbarkeit die erste Gruppe. Das wird bereits beim Datensatz 20VBD klar, bei dem die Lösungen im Vergleich zu den anderen Ansätzen über Faktor 3 zu teuer ist. Die maximal empfohlene Problemgrösse liegt also bei 10 Besuchen. Es muss jedoch erwähnt werden, dass teilweise auch grössere Probleminstanzen gelöst werden können.

Der optimale Anwendungsbereich für den Google Routing Lösungsansatz endet mit Datensatz 50VBUDU. Dies ist auch an den verwendeten Strategien ersichtlich, die in Kapitel 3.5 beschrieben sind. Bei Probleminstanzen, die mehr als 50 Besuche beinhalten, kommt die Strategie zum Zug, bei der die Zeitfenster ignoriert werden. Das heisst, die Leistungsfähigkeit reicht lediglich dafür aus, ein VRP mit beschränkten Tagen zu lösen. Die Kosten der Lösungen für die Datensätze mit über 100 Besuchen sind in diesem Fall um knapp Faktor zwei schlechter als bei LocalSolver. Bei den anderen Datensätzen liegen die Abweichungen zu LocalSolver unter 13%.

Bis Datensatz 100VBUDU kann ILP2 eingesetzt werden. Beim Datensatz 200VBUDU kann innerhalb des Zeitlimits keine Lösung gefunden werden. Bei Datensatz 1000VBUDU reichen die 2 GB Arbeitsspeicher der Testmaschine nicht mehr, um die Formulierung überhaupt zu laden oder gar zu lösen.

Der genetische Algorithmus kann ohne Bedenken bis Datensatz 200VBDU verwendet werden. Erst beim Datensatz 1000VBDU reicht die Rechenzeit nicht mehr und die Lösung wird gerundet 60% teurer als diejenige von LocalSolver.

Mit LocalSolver, dem besten Ansatz, kann als einziges eine gute Lösung für den Datensatz 1000VBDU berechnet werden. Aufgrund der Kosten pro Besuch, die bei LocalSolver bei den grössten drei Datensätzen alle um 21 CHF liegen, wird davon ausgegangen, dass auch noch grössere Probleminstanzen in nützlicher Frist optimiert werden könnten.

## 4.5 Lösungsqualität

In diesem Kapitel wird die Lösungsqualität der einzelnen Ansätze innerhalb des optimalen Anwendungsbereichs untersucht. Die optimalen Anwendungsbereiche sind im Kapitel 4.4 beschrieben. Die Abbildung 16 zeigt die Kosten der Lösungen, wobei die Kosten nur eingetragen sind, wenn der Datensatz im optimalen Anwendungsbereich des jeweiligen Ansatzes liegt. Zudem ist der Datensatz 1000VBDU weggelassen, da LocalSolver der einzige Ansatz ist, bei dem dieser Datensatz im optimalen Anwendungsbereich ist.



Abbildung 16: Vergleich Kosten pro Datensatz, skaliert nach best bekannter Lösung

Obwohl sich die Ansätze IP5 SCIP und IP5 Gurobi beim Datensatz 10V nur um einen Franken von der besten Lösung unterscheiden, kann bei diesen Ansätzen nicht von einer guten Lösungsqualität gesprochen werden. Der Grund dafür ist, dass die Lösungen bei den Datensätzen 10VD und 10VU mit ebenfalls 10 Besuchen über 9% schlechter ausfallen, als bei LocalSolver.

Die Lösungsqualität vom ILP2 ist auch innerhalb des optimalen Anwendungsbereichs abhängig von der Problemgrösse. Die Lösungen der Datensätze 10V, 10VD, 10VU und 20V entsprechen den besten gefundenen Lösungen. Bis zum Datensatz 34VBDO nimmt die Abweichung tendenziell bis auf ungefähr 10% zu. Bei den grösseren Datensätzen steigt die Abweichung stark an und liegt beim Datensatz 100VBDO bei gerundet 35%.

Der genetische Algorithmus hat bis zum Datensatz 34VDU eine Abweichung von unter 5% verglichen mit LocalSolver. Bis zum Datensatz 200VBDO sind die Lösungen vom GA maximal 11% schlechter als von LocalSolver.

Der Ansatz LocalSolver erreicht bei allen Datensätzen den Bestwert und geht bei der Lösungsqualität klar als Sieger hervor. Dabei sind die Lösungen der Datensätze 10V, 10VD, 10VU und 20V bewiesen die bestmöglichen, da der erhaltene Wert gemäss ILP2 optimal ist.

Der Google Routing Ansatz hat bei den Datensätzen 10V, 10VU, 20V und 20VU die beste bekannte Lösung erzeugt. Allgemein weichen die Lösungen bis Datensatz 34VBDO verglichen mit LocalSolver lediglich maximal 4% ab. Erst beim Datensatz 50VBDO beträgt der Unterschied zur besten Lösung gerundet 12%.

## 4.6 Vergleich manuell geplante Route

Kapitel 4.5 verdeutlicht, wie die Ansätze betreffend Lösungsqualität gegeneinander abschneiden. Jedoch sagt das nichts darüber aus, ob diese Qualität der Lösungen auch für die praktische Anwendung ausreicht. In diesem Kapitel wird der Frage nachgegangen, ob die Ansätze bessere Lösungen erzeugen, als eine manuelle Planung.

Um objektiv Vergleichen zu können, werden die Kennzahlen der manuellen Lösungen berechnet. Aus den Kennzahlen wird der Wert der Zielfunktion berechnet. Gegenstand dieser Betrachtung sind ausschliesslich die Datensätze 34VDU und 34VBDO, weil dies die einzigen realen Datensätze sind. Da die Chläuse bei der Planung nicht mit denselben Wegzeiten gerechnet haben, wird aus der realen Routenplanung nur die Besuchszeit der ersten Familien übernommen. Die nachfolgenden Familien werden danach der Reihe nach hinzugefügt, wobei die exakten Wegzeiten aus den Datensätzen verwendet werden. Dieses Verfahren wurde gewählt, um zu lange Wegzeiten zu verkürzen und so einen möglichst fairen Vergleich zu ermöglichen. Im Normalfall wird die Lösung durch dieses Verfahren besser. Die Lösung kann jedoch schlechter werden, wenn durch die Zeitverschiebung ein Besuch in der Nichtverfügbarkeit erstattet wird oder Besuche aus dem Wunschzeitraum verschoben werden. Durch eine Prüfung beim Erzeugen der Kennzahlen wurde sichergestellt, dass diese beiden Fälle nicht eingetreten sind.

Abbildung 17 und Abbildung 18 zeigen die Kosten der Lösungen pro Ansatz. Die Kosten sind dabei farblich unterteilt, sodass deren Aufteilung ersichtlich ist.

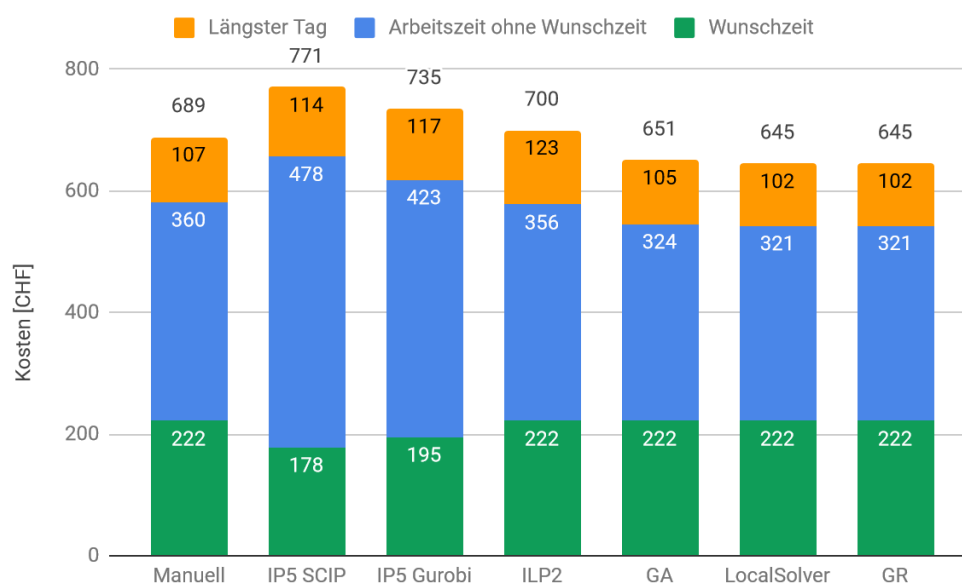


Abbildung 17: Kosten für die Routen des Datenasatz 34VDU. Manuell entspricht der händischen Planung aus dem Jahr 2017. Allfällige Rundungsdifferenzen sind beim längsten Tag addiert.

In Abbildung 17 wird das Resultat der Lösungen für den Datensatz 34VDU verglichen. Beim Datensatz 34VDU, welcher dem Jahr 2017 entspricht, können im Vergleich zur manuellen Planung lediglich die Ansätze GA, LocalSolver und Google Routing eine bessere Lösung vorweisen. Die möglichen Ersparnisse belaufen sich auf bis zu 44 CHF was gerundet 6.4% entspricht.

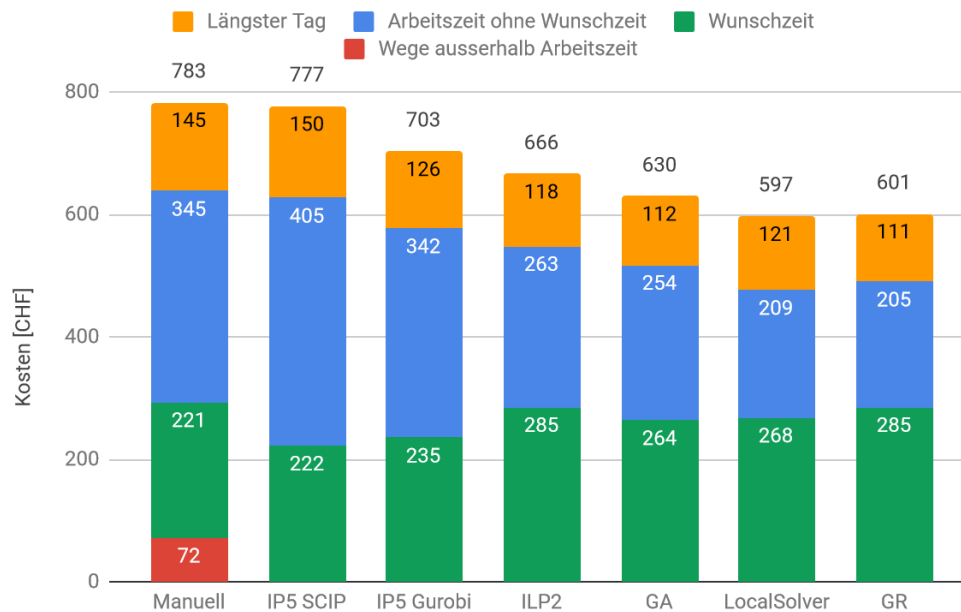


Abbildung 18: Kosten für die Routen des Datensatz 34VBDU. Manuell entspricht der händischen Planung aus dem Jahr 2018. Allfällige Rundungsdifferenzen sind beim längsten Tag addiert.

In Abbildung 18 wird das Resultat der Lösungen für den Datensatz 34VBDU verglichen. Beim Datensatz 34VBDU, welcher dem Jahr 2018 entspricht, ist die manuelle Planung die schlechteste Lösung. Ein Hauptgrund dafür ist, dass bei der manuellen Lösung Wege ausserhalb der Arbeitszeit getätigt werden. Beim Datensatz 8 betragen die möglichen Ersparnisse 182 CHF was gerundet 23% entspricht.

## 4.7 Lösungsstabilität

Wie in Kapitel 4.1 beschrieben wird, werden pro Lösungsansatz fünf Durchläufe durchgeführt, um allfällige Lösungsinstabilitäten aufzudecken. Das Whisker-Diagramm aus Abbildung 19 zeigt, dass diese Schwankungen kaum praxisrelevant sind.

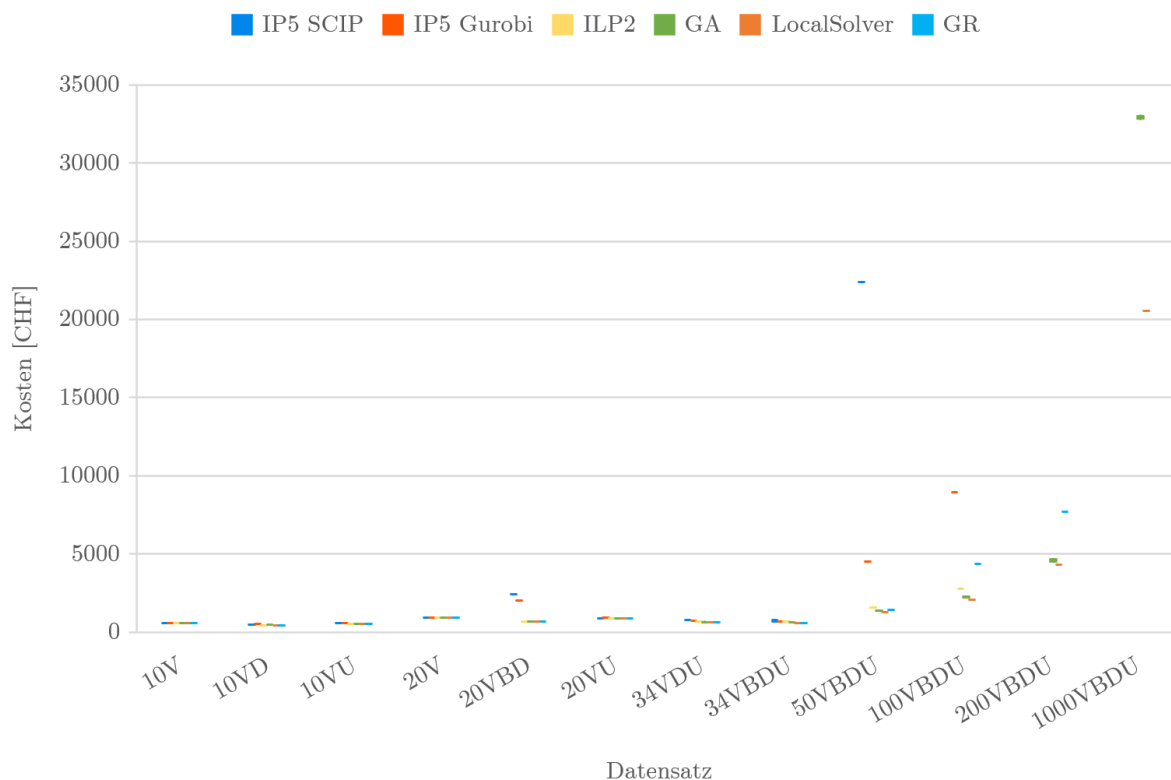


Abbildung 19: Whisker-Diagramm welches die Abweichungen innerhalb der fünf Durchgänge pro Lösungsansatz aufzeigt.

Alle drei ILP-Ansätze sind meistens stabil. Bei den Lösungen des IP5 SCIP treten nur für die Datensätzen 34VDU und 34VBDU Lösungsinstabilitäten auf. Die beste IP5 SCIP-Lösung des Datensatzes 34VBDU ist unter 14% günstiger, als diejenige des schlechtesten Durchgangs.

Für IP5 Gurobi treten beim Datensatz 100VBDU kleine Abweichungen von einem Franken auf.

Der Lösungsansatz ILP2 verhält sich abgesehen vom Datensatz 100VBDU stabil. Beim Datensatz 100VBDU lassen sich kleine Unterschiede zwischen den Lösungen feststellen, da teilweise kurz vor dem Ende noch eine bessere Lösung gefunden wird.

Beim genetischen Algorithmus treten ab Datensatz 20V Lösungsinstabilitäten auf. Diese Lösungsinstabilitäten nehmen mit der Grösse der Datensätze zu. Über alle Datensätze betrachtet erweisen sich die Kostenunterschiede durch Instabilitäten nicht gravierend. So liegt der Unterschied zwischen der schlechtesten und der besten Lösung eines Datensatzes unter 4%. Die Ansätze LocalSolver und Google Routing sind grundsätzlich sehr stabil und zeigen nur beim Datensatz 1000VBDU Abweichungen. Beim Datensatz 1000VBDU sind die Abweichungen verschwindend klein. Sie betragen weniger als ein Promille der Gesamtkosten und sind somit irrelevant.

## 4.8 Auswirkung Wunschzeit & Nichtverfügbarkeit

Um die Auswirkungen der Wunschzeiten und Nichtverfügbarkeiten zu analysieren wurden drei synthetische Testinstanzen generiert. Sie werden nachfolgend TI, TIU und TID genannt. Die Testinstanzen haben alle dieselben 20 Familien, die besucht werden müssen. Es werden jeweils 2 Chläuse an 2 Tagen eingesetzt, wobei kein Chlaus eine Pause eingetragen hat. Bei TIU ist für die ersten 10 Familien am ersten Tag und für die zweiten 10 am zweiten Tag Nichtverfügbarkeit eingetragen. Bei TID haben die ersten 10 Familien am ersten Tag Wunschzeiten und die zweiten 10 am zweiten Tag. Die Wunschzeiten und Nichtverfügbarkeiten werden analog dem Kapitel 2 generiert. Für TI werden weder Nichtverfügbarkeiten noch Wunschzeiten definiert. Die Testinstanzen werden mit jedem Lösungsansatz fünfmal berechnet. Als Zeitlimit sind 5 Minuten Rechenzeit gesetzt, um die Auswirkungen besser aufzeigen zu können. Als Messwert wird das dritte Quartil genommen.

Lösungsansatz	TI	%	TIU	%	TID	%
IP5 SCIP	855	7.68	823	2.24	761	32.35
ILP Gurobi	801	0.88	823	2.24	781	35.83
ILP2	794	0	831	3.23	609	5.91
GA	807	1.64	836	3.85	624	8.52
LocalSolver	794	0	805	0	575	0
Google Routing	794	0	805	0	581	1.04

Tabelle 14: Messwerte Datensatz TI, TIU und TID in CHF

In der Tabelle 14 sind die dritten Quartile der einzelnen Lösungsansätze sowie die prozentuale Abweichung zu der besten erreichten Lösung aufgeführt. Zu erkennen ist, dass in der kürzeren Rechenzeit nur der LocalSolver, das Google Routing und das ILP2 das Optimum für den Datensatz TI erreicht haben. Die berechnete Lösung ist gemäss ILP2 optimal (siehe Anhang 8.4). Der genetische Algorithmus wurde für Berechnungen mit 20 Besuchen innerhalb von 20 Minuten Rechenzeit parametrisiert, weshalb er hier nicht die beste bekannte Lösung erreicht. Das IP5 SCIP sowie das IP5 Gurobi konnten das Problem nicht optimal lösen.



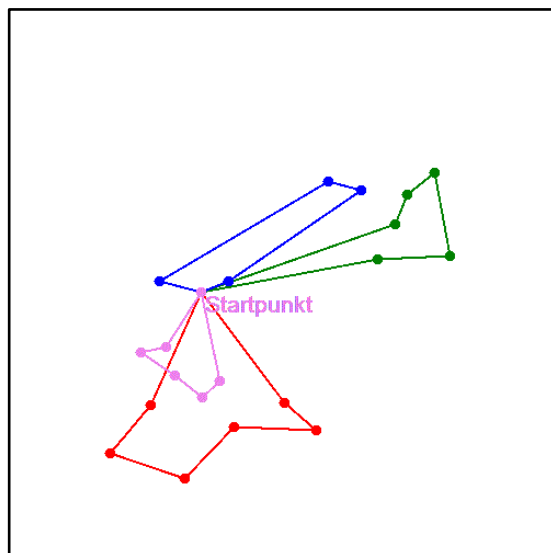


Abbildung 20: TI bewiesen optimale Lösung

Kommen Nichtverfügbarkeiten hinzu, ist beim IP5 Gurobi (siehe Anhang 8.3) sowie beim IP5 SCIP ersichtlich, dass der erste Teil, das Clustering, innerhalb weniger Sekunden gelöst werden kann. Die Nichtverfügbarkeit hilft hier das Problem schneller zu lösen, weil damit die Aufteilung der Besuche mit einer harten Einschränkung einfacher gemacht werden kann. Durch diese Aufteilung wird die Lösung jedoch stärker eingeschränkt und sie weicht um 2.24% von der besten bekannten Lösung ab. Obwohl das ILP2 ebenfalls als ganzzahliges lineares Programm formuliert wurde, erschwert die Einführung von Nichtverfügbarkeiten das Problem. So kann für das Problem ohne Nichtverfügbarkeit nur die erste Phase des Modells verwendet werden. Dieses Modell ist wesentlich weniger schwer zu Lösen als das vollständig formulierte.

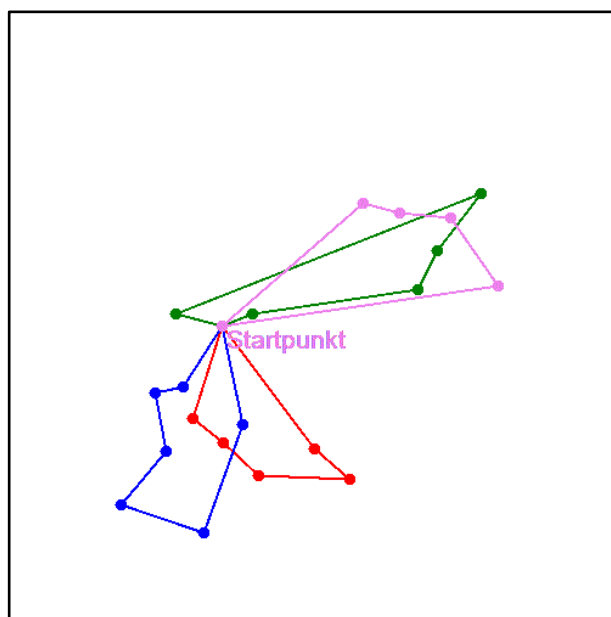


Abbildung 21: TIU beste bekannte Lösung

In der Abbildung 21 sieht man, dass die beste bekannte Lösung der Testinstanz TIU von der besten bekannten Lösung der Testinstanz TI, ersichtlich in Abbildung 20, stark abweicht. Es stimmen pro Route nie mehr als drei Punkte überein. Dies ist ein weiterer Grund, weshalb das

ILP2 eine schlechte Lösung berechnet, da 10% der Zeit für das Lösen des VRP verwendet werden. Der genetische Algorithmus verhält sich ähnlich wie bei der Testinstanz TI. Mit dem genetischen Algorithmus ist es jedoch nicht möglich, die beste bekannte Lösung zu erreichen, da die Startzeiten der Routen immer zum Tagesstart gesetzt sind und die beste bekannte Lösung erst später startet.

Das Google Routing sowie der LocalSolver haben bei dieser Problem Instanz die beste bekannte Lösung berechnet. Der Rechenaufwand wird für die beiden Lösungsansätze grösser. Beim Google Routing werden Datensätze mit über 50 Besuchen aufgrund des zu grossen Rechenaufwands ohne Nichtverfügbarkeiten gerechnet.

Eine weitaus grössere Auswirkung haben die Wunschzeiten. Mit Wunschzeiten ist eine deutliche Verschlechterung der Lösungen aller Lösungsansätze mit der Ausnahme von LocalSolver ersichtlich. Durch die vorherige Aufteilung der Routen im IP5 SCIP und IP5 Gurobi ist eine gute Lösung nicht mehr möglich. Beide Lösungsansätze weichen um mehr als 30% von der besten bekannten Lösung ab.

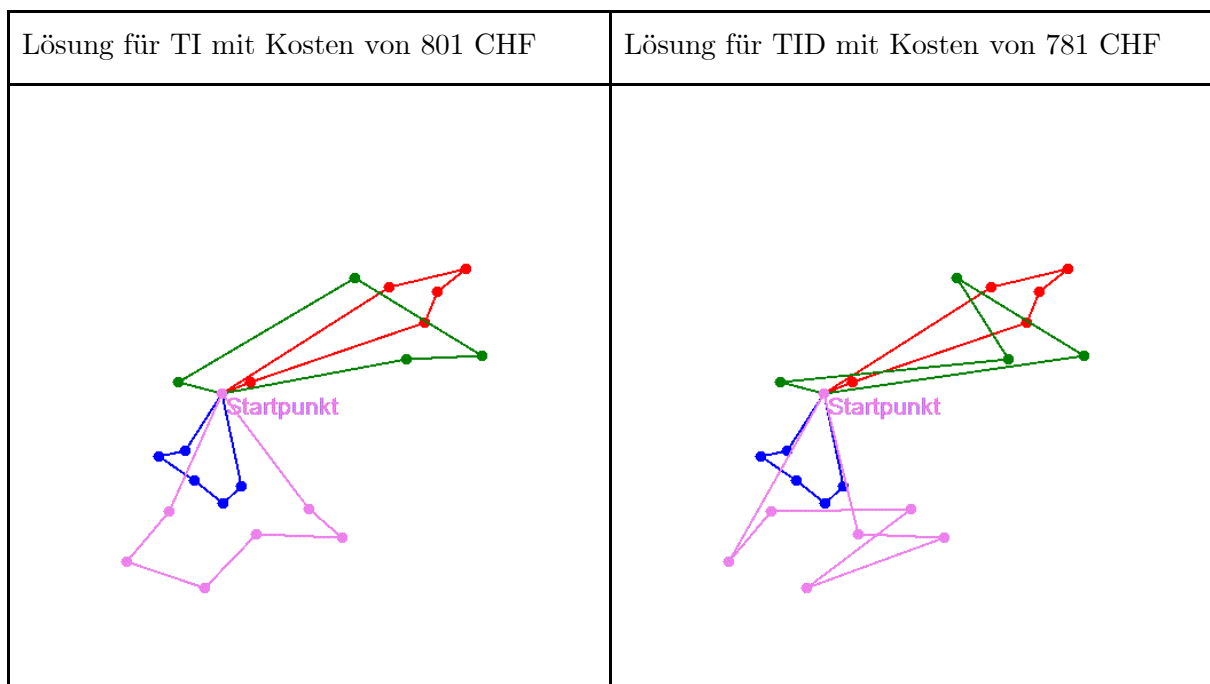


Abbildung 22: Vergleich ILP Gurobi Lösung TI und TID

In der Abbildung 22 ist ersichtlich, dass die berechneten Routen für die Testinstanz TID dieselben Besuche enthalten, wie die Lösung für die Testinstanz TI. Der Unterschied liegt lediglich in der Reihenfolge wie diese besucht werden.

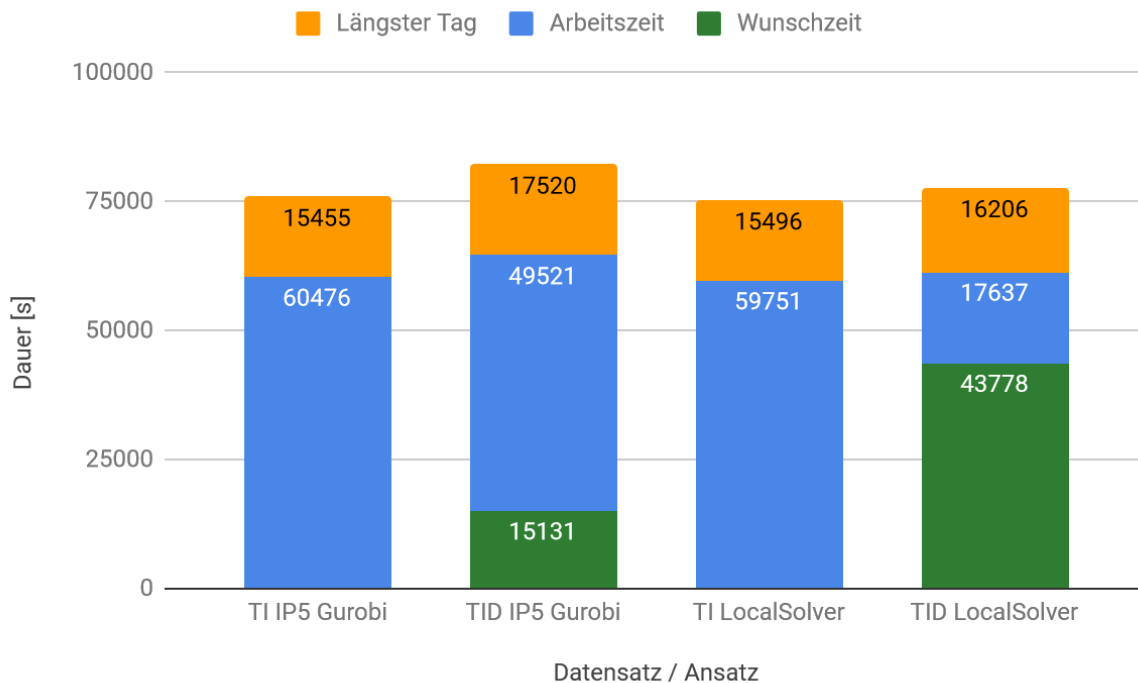


Abbildung 23: Vergleich Wunschzeit IP5 Gurobi gegenüber Localsolver

Vergleicht man die Resultate zwischen dem IP5 Gurobi und der besten bekannten Lösung, so wird der Unterschied der erfüllten Wunschzeit deutlich. Während bei der vom LocalSolver berechneten Route gerundet nur 28 Minuten mehr Arbeitszeit als bei der Lösung der Testinstanz TID anfällt, wird bei der IP5 Gurobi Lösung 70 Minuten Mehraufwand generiert. Für den ILP2 Ansatz ist die Wunschzeit ebenfalls hinderlich. Das Resultat weicht stärker von der besten bekannten Lösung ab, als bei den Berechnungen mit Nichtverfügbarkeiten oder ohne Zeitfenster. Das Google Routing erreicht mit den Wunschzeiten nicht mehr die beste bekannte Lösung. Ein Grund ist hier die Implementation des Algorithmus. Die Wunschzeiten werden nicht exakt berechnet, sondern es wird der Abstand zur Wunschzeit linear bestraft. Dies führt dazu, dass die Besuche möglichst nahe an die Wunschzeiten gelegt werden. Dadurch entsteht hier eine Abweichung von einem Prozent zur besten bekannten Lösung. Aus der Auswahl der Strategien im Kapitel 3.5 geht hervor, dass auch bei diesem Lösungsansatz der Rechenaufwand für Wunschzeiten grösser ist, als für Nichtverfügbarkeiten.

Der genetische Algorithmus verschlechtert sich ebenfalls stärker. Aufgrund des kleinen Gewichts von Wunschzeiten, werden in den ersten Generationen Lösungen bevorzugt, welche geringe Wegkosten und keine Nichtverfügbarkeit haben. Dadurch werden Individuen, die vor allem die Wunschzeiten einhalten, mit einer hohen Wahrscheinlichkeit aussortiert. Der Lösungsansatz LocalSolver berechnet erneut das beste bekannte Resultat. Der Rechenaufwand für Nichtverfügbarkeiten und Wunschzeit bleibt gleich.

## 4.9 Zusätzliche Chläuse

Die Lösungsansätze LocalSolver, genetischer Algorithmus und Google Routing haben die Möglichkeit zusätzliche Chläuse einzuführen. Diese zusätzlichen Chläuse kosten das doppelte an Stundenlohn sowie eine Pauschale von 400 CHF pro Chlaus. In diesem Kapitel soll aufgezeigt werden, wie sich das Einschalten dieser Funktionalität auf die Performance und Lösungsqualität der einzelnen Lösungsansätze auswirkt.

Die maximale Anzahl an Chläusen wird pro Datensatz auf die Anzahl der Besuche, das heisst Familienbesuche plus Pausen, multipliziert mit der Anzahl Arbeitstage, festgelegt. So ist sichergestellt, dass die optimale Lösung nicht ausgeschlossen wird.

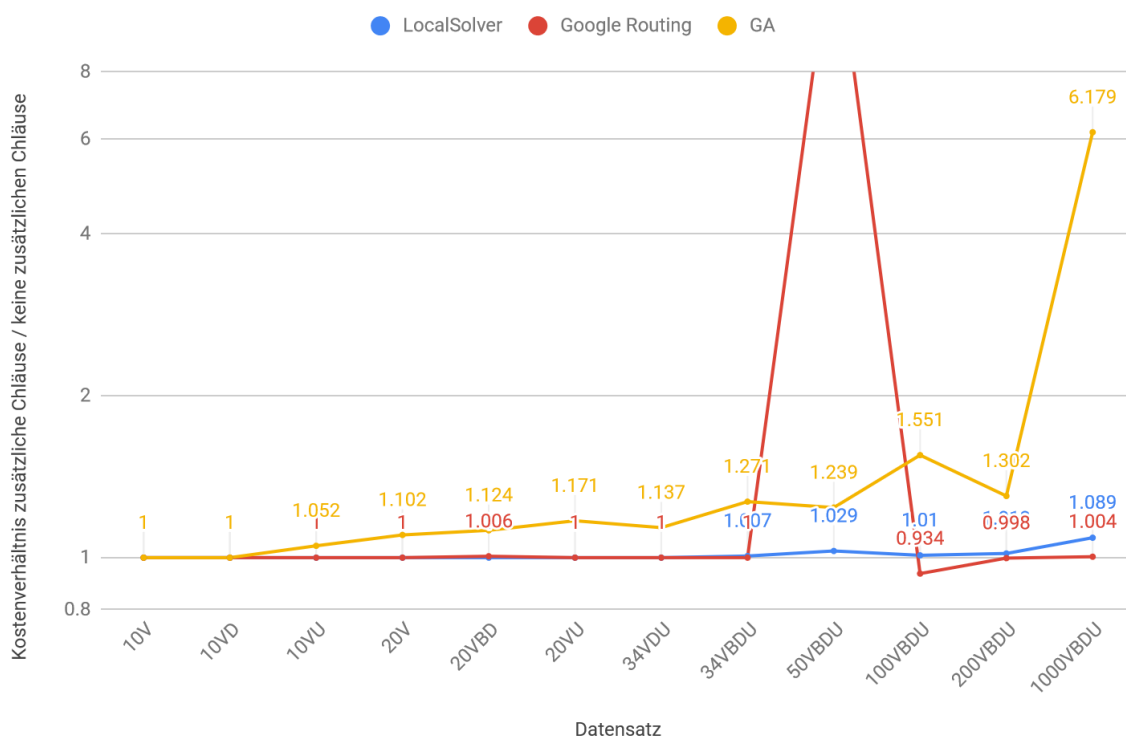


Abbildung 24: Kostenverhältnis zwischen der Lösung mit zusätzlichen Chläusen und ohne zusätzliche Chläuse

In der Abbildung 24 wird abgebildet, um welchen Faktor die Lösung mit zusätzlichen Chläusen von der Lösung ohne zusätzliche Chläusen abweicht. Deutlich zu erkennen ist der Ausschlag beim Datensatz 50VBDU vom Google Routing. Dieser Ausschlag ist dadurch zu erklären, weil bei dieser Berechnung keine valide Lösung innerhalb des Zeitlimits gefunden wird. Erstaunlich ist die Verbesserung der Resultate für 100 sowie 200 Besuche. Eine Begründung für diese Verbesserung lässt sich nicht eindeutig bestimmen. Die Verbesserung könnte jedoch mit dem Constraint-Solver zu tun haben, der schneller eine Lösung findet, da mehr Chläuse zur Verfügung stehen. So bleibt mehr Zeit für die “Large neighbourhood Search”, welche schlussendlich eine bessere Lösung findet.

Die erzeugten Lösungen des genetischen Algorithmus sind bis Datensatz 34VDU ungefähr 15% schlechter. Die Abweichung steigt bei zunehmender Instanzgrösse weiter an. Beim Datensatz 100VBDU beträgt die Abweichung bereits 55%.

Mit dem Lösungsansatz LocalSolver wird bis zum Datensatz 34VDU dieselbe Lösung berechnet. Bis zum Datensatz 200VBUDU beträgt die Abweichung zwischen 1% und 3%. Erst beim Datensatz 1000VBUDU verschlechtern sich die Lösungen um knapp 9%. Obwohl dies nach viel klingt, ist das berechnete Resultat immer noch besser, als die Resultate der anderen Lösungsansätze ohne zusätzliche Chläuse.

## 4.10 Lizenzkosten

Lösungsansatz	Akademisch	Nicht kommerziell	Kommerziell
IP5 SCIP [29] [30]	Gratis	Gratis	Auf Anfrage
IP5 Gurobi [31]	Gratis	Auf Anfrage	Auf Anfrage
ILP [31]	Gratis	Auf Anfrage	Auf Anfrage
GA	Gratis	Gratis	Gratis
LocalSolver [32]	Gratis	Auf Anfrage	Auf Anfrage
Google Routing [30]	Gratis	Gratis	Gratis

Tabelle 15: Übersicht Lizenzkosten für die Lösungsansätze

Der Ansatz IP5 SCIP basiert auf Google OR-Tools [30] sowie SCIP. SCIP ist unter der “ZIB Academic License” lizenziert und darf von akademischen und nicht kommerziellen Institutionen kostenfrei genutzt werden [29].

Die Ansätze IP5 Gurobi und ILP2 benötigen beide Gurobi und sind somit nur für akademische Nutzung kostenfrei. Andere Lizenzen müssen beim Gurobi Vertrieb angefragt werden [31].

Der genetische Algorithmus wurde im Rahmen dieses Projekts von Grund auf entwickelt und darf somit kostenfrei verwendet werden.

LocalSolver bietet ebenfalls kostenlose Lizenzen für akademische Nutzer. Diese sind jedoch auf ein physikalisches Gerät beschränkt und müssen jeden Monat erneuert werden [32].

Google Routing, welches auf den Google OR-Tools basiert, steht unter der “Apache License 2.0” und darf kostenlos verwendet werden [30].

## 5 Schlussfolgerung / Empfehlung

Aus der Evaluation geht hervor, dass LocalSolver der beste Ansatz ist. So ist die Empfehlung für die Jungwacht Niederwil, diesen Ansatz zu benutzen. Dabei spielt jedoch der Preis eine erhebliche Rolle. LocalSolver ist nur für akademische Anwendungszwecke kostenfrei. Möchte die Jungwacht LocalSolver benutzen, so muss LocalSolver betreffend einer Lizenz angefragt werden. Dabei ist wichtig, dass sich eine kostenpflichtige Lizenz praktisch nicht lohnt. Angesichts der Datensätze 34VDU und 34VBDO, ergeben sich mit LocalSolver mögliche Einsparungen von maximal 4 CHF verglichen mit Google Routing.

Das führt auch gleich zur alternativen Empfehlung, Google Routing zu benutzen, falls die Lizenz von LocalSolver etwas kostet. Wie im Kapitel 4 beschrieben, sind die Resultate vom Google Routing bei den Datensätzen mit maximal 34 Besuchen ähnlich gut, wie diejenigen von LocalSolver. Jedoch fallen für das Google Routing keinerlei Lizenzgebühren an.

Falls die Anmeldungen bei der Chlausgesellschaft in Zukunft deutlich steigen, kommt eine kostenpflichtige Lizenz bei LocalSolver wieder ins Spiel. Beispielsweise beim Datensatz 50VBDO ist die Lösung von LocalSolver 156 CHF günstiger, als diejenige vom Google Routing. Sollten die Lizenzkosten für LocalSolver deutlich höher sein, als diese 156 CHF, lohnt sich der Einsatz bei Datensatz 50VBDO nicht mehr. In dem Fall, dass mehr als 34 Besuche geplant werden sollen, lohnt sich der Einsatz des genetischen Algorithmus. Eine andere Möglichkeit ist auch, grössere Datensätze zuerst von Hand in mehrere kleinere Datensätze zu unterteilen, sodass diese kleineren Datensätze vom Google Routing effizient gelöst werden können. Eine solche Aufteilung birgt jedoch die Gefahr, dass die besten Lösungen aufgrund einer ungünstigen Aufteilung, ausgeschlossen werden.

Um das Produkt für den Kunden möglichst robust zu machen, lautet die Empfehlung wie folgt. Sofern preislich vertretbar, soll LocalSolver benutzt werden. Ob sich eine Lizenz lohnt, muss anhand der Ergebnisse der Evaluation abgeschätzt werden.

Sollte LocalSolver nicht in Frage kommen, wird empfohlen die Lösungen mit Google Routing und dem genetischen Algorithmus zu berechnen. Bestenfalls werden dabei beim Google Routing alle vier Strategien ausgeführt, was maximal vier Stunden dauert. Um den Instabilitäten des GA entgegenzuwirken, könnte dieser Ansatz dreimal mit einer Berechnungszeit von zwei Stunden ausgeführt werden. Mit diesem Vorgehen würde die Berechnung maximal zehn Stunden dauern, was es dem Kunden ermöglicht, die Routen innerhalb einer Nacht berechnen zu lassen.

Unabhängig davon, ob am Schluss LocalSolver oder eine Kombination aus GA und Google Routing eingesetzt wird, sind wir überzeugt, dass die Ansätze für die Chlausgesellschaft einen Mehrwert bieten und das Ziel erreicht ist. Durch die automatische Routenplanung werden die Routen nicht nur besser, sondern es entfällt auch ein grosser Teil der Handarbeit für die Planung. So können doppelt Kosten gespart werden. Das war beim Vorgängerprojekt, dem IP5, nicht der Fall, da die von Hand geplanten Routen tendenziell besser waren, als die berechneten.

Trotz des Umfangs dieser Arbeit, sind die Resultate stark auf die Problemstellung der Chlausgesellschaft Niederwil-Nesselnbach bezogen. Der Hauptgrund dafür ist, neben der Zielfunktion, dass die 12 Datensätze der Evaluation ähnliche Eigenschaften aufweisen, wie die originalen Daten des Kunden. So ist es möglich, dass die Ansätze ein anderes Lösungsverhalten zeigen, wenn beispielsweise die Besuche auf einem grösseren Gebiet verteilt sind oder die Besuche auf mehr als zwei Tage verteilt werden. Wie beim Solomon Benchmark [7] erkennbar, gibt es mehrere Eigenschaften, die eine VRPTW Problemstellung ausmachen können. Beispiele sind Datensätze, bei denen die Routen durch Nichtverfügbarkeiten oder Wunschzeiten faktisch vorgegeben sind, Datensätze mit ausschliesslich kurzen oder langen Wegen, Datensätze mit zufällig verteilten Besuchsorten oder lokalen Ansammlungen, Datensätze mit langen, oder solche mit kurzen Tagen. Sollten die entwickelten Ansätze allgemeingültig verglichen werden, müssten solche Aspekte genauer untersucht werden.

## 6 Verzeichnisse

### 6.1 Abbildungsverzeichnis

Abbildung 1: Datensatz mit 1000 Besuchen.....	8
Abbildung 2: Ablauf IP5.....	11
Abbildung 3: Subtour Problem, Startpunkt bei A.....	16
Abbildung 4: Subtour Eliminierung .....	20
Abbildung 5: Wunschzeit Einschränkungen .....	21
Abbildung 6: Nichtverfügbarkeit Einschränkungen .....	22
Abbildung 7: Vereinfachter Ablauf eines genetischen Algorithmus angelehnt an [9].....	23
Abbildung 8: Aufbau des eingesetzten Modells des GA angelehnt an [9].....	24
Abbildung 9: Beispielgenotyp.....	24
Abbildung 10: Beispiel Positions- und Inversionsmutation angelehnt an [9].....	27
Abbildung 11: Lösungsqualität in Abhängigkeit der Populationsgrösse.....	32
Abbildung 12: Graphischer Verlauf der Funktion $f$ (55) In der X-Achse ist die Instanzgrösse und in der Y-Achse die Populationsgrösse abgebildet.....	33
Abbildung 13: Evaluationswert pro Ansatz.....	47
Abbildung 14: Rechenzeit pro Ansatz über alle 12 Datensätze.....	48
Abbildung 15: Kostenverhältnis zu den besten bekannten Kosten pro Datensatz.....	49
Abbildung 16: Vergleich Kosten pro Datensatz, skaliert nach best bekannter Lösung.....	50
Abbildung 17: Kosten für die Routen des Datenasatz 34VDU. Manuell entspricht der händischen Planung aus dem Jahr 2017. Allfällige Rundungsdifferenzen sind beim längsten Tag addiert.....	52
Abbildung 18: Kosten für die Routen des Datenasatz 34VBDU. Manuell entspricht der händischen Planung aus dem Jahr 2018. Allfällige Rundungsdifferenzen sind beim längsten Tag addiert.....	53
Abbildung 19: Whisker-Diagramm welches die Abweichungen innerhalb der fünf Durchgänge pro Lösungsansatz aufzeigt.....	54
Abbildung 20: TI bewiesen optimale Lösung.....	56
Abbildung 21: TIU beste bekannte Lösung.....	56
Abbildung 22: Vergleich ILP Gurobi Lösung TI und TID.....	57
Abbildung 23: Vergleich Wunschzeit IP5 Gurobi gegenüber Localsolver.....	58
Abbildung 24: Kostenverhältnis zwischen der Lösung mit zusätzlichen Chläusen und ohne zusätzliche Chläuse.....	59



## 6.2 Tabellenverzeichnis

Tabelle 1: Zielfunktion der Problemstellung.....	7
Tabelle 2: Kennzahlen der Datensätze .....	9
Tabelle 3: Scheduling Planungsdarstellung .....	12
Tabelle 4: ILP2 Rastersuche Resultat 1 .....	13
Tabelle 5: ILP2 Rastersuche Resultat 2.....	13
Tabelle 6: Resultate PSO mit Zeitlimit 0.5 s .....	30
Tabelle 7: Resultate PSO mit Zeitlimit 0.25 s.....	31
Tabelle 8: Beste gefundene Populationsgrösse pro Instanzgrösse.....	32
Tabelle 9: LocalSolver Rastersuche für Zeitlimiten .....	35
Tabelle 10: Variablendefinition erste Phase für LocalSolver .....	36
Tabelle 11: Variablendefinition zweite Phase für LocalSolver .....	38
Tabelle 12: Kosten in CHF der Lösungen pro Strategie .....	45
Tabelle 13: Gewichtung der Datensätze für die Evaluation.....	46
Tabelle 14: Messwerte Datensatz TI, TIU und TID in CHF .....	55
Tabelle 15: Übersicht Lizenzkosten für die Lösungsansätze.....	60

## 6.3 Literaturverzeichnis

- [1] A. E.-S. Nasser, «Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods,» Journal of King Saud University, Cairo, 2010.
- [2] G. Dantzig und J. H. Ramser, «The truck dispatching problem,» INFORMS, 1959.
- [3] N. Christofides und S. Eilon, «An Algorithm for the Vehicledispatching Problem,» Palgrave Macmillan UK, London, 1969.
- [4] Quintiq, «Weltrekorde VRPTW,» Quintiq, [Online]. Available: <https://www.quintiq.de/optimierung/weltrekorde-vrptw.html>. [Zugriff am 19 März 2019].
- [5] IBM, «IBM ILOG CPLEX CP Optimizer,» IBM, [Online]. Available: <https://www.ibm.com/analytics/cplex-cp-optimizer>. [Zugriff am 18 März 2019].
- [6] SAP, «Effective Optimization: run-time,» SAP, 21 März 2014. [Online]. Available: <https://blogs.sap.com/2014/03/21/effective-optimization-run-time/>. [Zugriff am 18 März 2019].
- [7] sintef, «Solomon benchmark,» 18 April 2008. [Online]. Available: <https://www.sintef.no/projectweb/top/vrptw/solomon-benchmark/>. [Zugriff am 18 März 2019].
- [8] sintef, «Gehrig & Homberger benchmark,» 18 April 2008. [Online]. Available: <https://www.sintef.no/projectweb/top/vrptw/homberger-benchmark/>. [Zugriff am 18 März 2019].

- [9] M. Vogel, «Meta-Heuristiken,» Windisch, 2017.
- [10] LocalSolver, «LocalSolver wins on some of the hardest MIPLIB instances,» LocalSolver, 15 Oktober 2013. [Online]. Available: <https://www.localsolver.com/news.html?id=32>. [Zugriff am 18 März 2019].
- [11] P. Lüscher und J. Meyer, «iRuettä,» Windisch, 2018.
- [12] Gurobi, «Gurobi General Constraints,» Gurobi, [Online]. Available: <http://www.gurobi.com/documentation/8.1/refman/constraints.html#subsubsection:GeneralConstraints>. [Zugriff am 18 März 2019].
- [13] Gurobi, «The Traveling Salesman Problem with integer programming and Gurobi,» Gurobi, [Online]. Available: <http://examples.gurobi.com/traveling-salesman-problem/>. [Zugriff am 18 März 2019].
- [14] I. Griva, S. Nash und A. Sofer, Linear and Nonlinear optimization, Virginia: Society for Industrial Mathematics, 2009.
- [15] K. Q. Zhu, «A New Genetic Algorithm for VRPTW,» 2000.
- [16] A. Umbarkar und P. Sheth, «Crossover operators in genetic algorithms: a review,» ICTACT journals, 2015.
- [17] R. Mercer und J. Sampson, «Adaptive search using a reproductive meta-plan,» MCB UP Ltd, Alberta, 1978.
- [18] M. E. H. Pedersen, «Good Parameters for Particle Swarm Optimization,» 2010.
- [19] C. R. Reeves, «Using Genetic Algorithms With Small Populations,» 1998.
- [20] S. Gotshall und B. Rylander, «Optimal Population Size and the Genetic Algorithm,» 2000.
- [21] P. Lüscher und J. Meyer, «GA Test Population Size,» 2019. [Online]. Available: <https://github.com/Isitar/ip6-vehicle-routing-problem-with-time-windows/blob/master/Results/GA/Test%20Populations%20Size.csv>. [Zugriff am 21 März 2018].
- [22] T. Benoist und B. Estellon, «LocalSolver 1.x,» Springer-Verlag, 2011.
- [23] T. Benoist, B. Estellon, F. Gardi und K. Nouioua, «Toward Local Search Programming: LocalSolver 1.0,» 2010.
- [24] F. Gardi und K. Erwin, *Yet Another Math Programming Consultant*, 2012.
- [25] LocalSolver, «LocalSolver Mathematical modeling features - Constraints,» LocalSolver, [Online]. Available: <https://www.localsolver.com/docs/last/quickstart/mathematicalmodelingfeatures.html#constraints>. [Zugriff am 18 März 2019].
- [26] LocalSolver, «LocalSolver: Overview,» LocalSolver, [Online]. Available: <https://www.localsolver.com/product.html>. [Zugriff am 18 März 2019].

- [27] Google, «Google OR-Tools: About Combinatorial Optimization,» [Online]. Available: <https://developers.google.com/optimization/introduction/overview>. [Zugriff am 21 März 2019].
- [28] Google, «Google OR-Tools: Routing,» [Online]. Available: <https://developers.google.com/optimization/routing/>. [Zugriff am 21 März 2019].
- [29] SCIP, «SCIP: License,» [Online]. Available: <https://scip.zib.de/index.php#license>. [Zugriff am 21 März 2019].
- [30] Google, «Google OR-Tools: License,» [Online]. Available: <https://github.com/google/or-tools#license>. [Zugriff am 21 März 2019].
- [31] Gurobi, «Gurobi: Lizenz Center,» [Online]. Available: <http://www.gurobi.com/downloads/licenses/license-center>. [Zugriff am 21 März 2019].
- [32] LocalSolver, «LocalSolver: Licensing,» [Online]. Available: <https://www.localsolver.com/pricing.html>. [Zugriff am 21 März 2019].

## 7 Ehrlichkeitserklärung

Hiermit bestätigen wir, Pascal Lüscher und Janik Meyer, die vorliegende Arbeit selbständig, ohne Hilfe Dritter und nur unter Benutzung der angegebenen Quellen verfasst zu haben.

---

Pascal Lüscher

Janik Meyer

## 8 Anhang

### 8.1 Source Code

Der Source Code dieser Projektarbeit kann auf GitHub unter folgender Adresse eingesehen werden: <https://github.com/Isitar/ip6-vehicle-routing-problem-with-time-windows>

Darin enthalten sind unter anderem alle vorgestellten Lösungsansätze, sowie eine Webapplikation für die praktische Routenplanung.

### 8.2 Resultate der Evaluationsdurchläufe

IP5 SCIP

Datensatz	Avg	Min	Max	3. Quartil	Messwerte
10V	561	561	561	561	561,561,561,561,561
10VD	493	493	493	493	493,493,493,493,493
10VU	582	582	582	582	582,582,582,582,582
20V	928	928	928	928	928,928,928,928,928
20VBD	2446	2446	2446	2446	2446,2446,2446,2446,2446
20VU	882	882	882	882	882,882,882,882,882
34VDU	769.2	764	771	771	770,771,770,764,771
34VBDU	732	669	777	776	777,751,687,776,669
50VBDU	22400	22400	22400	22400	22400,22400,22400,22400,22400
100VBDU	44800	44800	44800	44800	44800,44800,44800,44800
200VBDU	89600	89600	89600	89600	-
1000VBDU	448000	448000	448000	448000	-

IP5 Gurobi

Datensatz	Avg	Min	Max	3. Quartil	Messwerte
10V	561	561	561	561	561,561,561,561,561,561
10VD	512	512	512	512	512,512,512,512,512,512
10VU	582	582	582	582	582,582,582,582,582,582
20V	913	913	913	913	913,913,913,913,913,913
20VBD	2018	2018	2018	2018	2018,2018,2018,2018,2018,2018
20VU	910	910	910	910	910,910,910,910,910,910
34VDU	735	735	735	735	735,735,735,735,735,735
34VBDU	703	703	703	703	703,703,703,703,703,703
50VBDU	4518	4518	4518	4518	4518,4518,4518,4518,4518,4518
100VBDU	8924.3	8924	8925	8925	8925,8924,8924,8925,8924,8924
200VBDU	89600	89600	89600	89600	-
1000VBDU	448000	448000	448000	448000	-

ILP2

Datensatz	Avg	Min	Max	3. Quartil	Messwerte
10V	560	560	560	560	560,560,560,560,560
10VD	446	446	446	446	446,446,446,446,446
10VU	532	532	532	532	532,532,532,532,532
20V	912	912	912	912	912,912,912,912,912
20VBD	687	687	687	687	687,687,687,687,687
20VU	891	891	891	891	891,891,891,891,891
34VDU	700	700	700	700	700,700,700,700,700
34VBDU	666	666	666	666	666,666,666,666,666
50VBDU	1556	1556	1556	1556	1556,1556,1556,1556,1556
100VBDU	2774.4	2756	2798	2787	2763,2798,2756,2787,2768
200VBDU	89600	89600	89600	89600	89600
1000VBDU	448000	448000	448000	448000	-

Genetischer Algorithmus

Datensatz	Avg	Min	Max	3. Quartil	Messwerte
10V	560	560	560	560	560,560,560,560,560
10VD	462	462	462	462	462,462,462,462,462
10VU	543	543	543	543	543,543,543,543,543
20V	912.4	912	914	912	912,914,912,912,912
20VBD	685.2	681	689	686	685,681,689,685,686
20VU	861.2	861	862	861	861,861,861,862,861
34VDU	650.8	648	658	651	648,649,658,651,648
34VBDU	624.8	610	631	630	630,631,610,625,628
50VBDU	1373.2	1347	1397	1386	1386,1384,1347,1397,1352
100VBDU	1906.3	2246	2293	2274	2273,2293,2246,2274,2265
200VBDU	1912.6	4451	4585	4570	4451,4568,4570,4538,4585
1000VBDU	32931.2	32802	33021	33000	33021,32851,32982,33000,32802

LocalSolver

Datensatz	Avg	Min	Max	3. Quartil	Messwerte
10V	560	560	560	560	560,560,560,560,560
10VD	446	446	446	446	446,446,446,446,446
10VU	532	532	532	532	532,532,532,532,532
20V	912	912	912	912	912,912,912,912,912
20VBD	655	655	655	655	655,655,655,655,655
20VU	855	855	855	855	855,855,855,855,855
34VDU	645	645	645	645	645,645,645,645,645
34VBDU	597	597	597	597	597,597,597,597,597
50VBDU	1295	1295	1295	1295	1295,1295,1295,1295,1295
100VBDU	2060	2060	2060	2060	2060,2060,2060,2060,2060
200VBDU	4305	4305	4305	4305	4305,4305,4305,4305,4305
1000VBDU	20547.2	20547	20548	20547	20547,20547,20547,20547,20548

Google Routing

Datensatz	Avg	Min	Max	3. Quartil	Messwerte
10V	560	560	560	560	560,560,560,560,560
10VD	452	452	452	452	452,452,452,452,452
10VU	532	532	532	532	532,532,532,532,532
20V	912	912	912	912	912,912,912,912,912
20VBD	681	681	681	681	681,681,681,681,681
20VU	855	855	855	855	855,855,855,855,855
34VDU	645	645	645	645	645,645,645,645,645
34VBDU	601	601	601	601	601,601,601,601,601
50VBDU	1451	1451	1451	1451	1451,1451,1451,1451,1451
100VBDU	4369	4369	4369	4369	4369,4369,4369,4369,4369
200VBDU	7693	7693	7693	7693	7693,7693,7693,7693,7693
1000VBDU	36817	36805	36820	36820	36805,36820,36820,36820,36820



## 8.3 IP5 Gurobi Logfile Clustering TIU

Gurobi 8.0.1 (win64, .NET) logging started 02/17/19 19:42:35

Academic license - for non-commercial use only

Optimize a model with 14902 rows, 5473 columns and 78549 nonzeros

Variable types: 1772 continuous, 3701 integer (3700 binary)

Coefficient statistics:

Matrix range [1e+00, 4e+03]

Objective range [8e-03, 1e-02]

Bounds range [1e+00, 2e+09]

RHS range [1e+00, 3e+04]

Warning: Model contains large bounds

Consider reformulating model or setting NumericFocus parameter to avoid numerical issues.

Presolve removed 13486 rows and 4682 columns

Presolve time: 0.16s

Presolved: 1416 rows, 791 columns, 8139 nonzeros

Variable types: 342 continuous, 449 integer (442 binary)

Root relaxation: objective 7.971517e+02, 589 iterations, 0.02 seconds

Nodes		Current Node			Objective Bounds			Work	
Expl	Unexpl	Obj	Depth	IntInf	Incumbent	BestBd	Gap	It/Node	Time
	0	0	797.15166	0	65	- 797.15166	-	-	0s
H	0	0			909.4611111	797.15166	12.3%	-	0s
	0	0	797.15452	0	77	909.46111	797.15452	12.3%	0s
	0	0	799.44311	0	76	909.46111	799.44311	12.1%	0s
	0	0	800.23029	0	90	909.46111	800.23029	12.0%	0s
	0	0	800.23029	0	94	909.46111	800.23029	12.0%	0s
	0	0	800.54935	0	97	909.46111	800.54935	12.0%	0s
	0	0	800.62137	0	105	909.46111	800.62137	12.0%	0s
	0	0	800.92070	0	99	909.46111	800.92070	11.9%	0s
	0	0	800.93194	0	102	909.46111	800.93194	11.9%	0s
	0	0	800.93839	0	100	909.46111	800.93839	11.9%	0s
	0	0	800.93839	0	100	909.46111	800.93839	11.9%	0s
	0	0	801.05834	0	113	909.46111	801.05834	11.9%	0s
	0	0	801.05849	0	116	909.46111	801.05849	11.9%	0s
	0	0	801.09127	0	111	909.46111	801.09127	11.9%	0s
	0	0	801.34165	0	105	909.46111	801.34165	11.9%	0s
	0	0	801.35272	0	105	909.46111	801.35272	11.9%	0s
	0	0	801.35272	0	105	909.46111	801.35272	11.9%	0s

	0	0	801.35272	0	105	909.46111	801.35272	11.9%	-	0s
	0	0	801.35272	0	105	909.46111	801.35272	11.9%	-	0s
	0	0	801.35272	0	78	909.46111	801.35272	11.9%	-	0s
	0	2	801.35272	0	78	909.46111	801.35272	11.9%	-	0s
H	29	30				856.7194444	802.44154	6.34%		25.7 0s
H	32	31				851.3861111	802.44154	5.75%		23.8 0s
*	60	44		25		848.0527778	804.22339	5.17%		20.5 0s
H	523	207				847.5361111	814.07806	3.95%		16.9 1s
*	769	319		23		846.7194444	816.10304	3.62%		15.9 1s
*	771	307		21		845.3861111	816.10304	3.46%		15.9 1s
*	1339	409		18		844.0527778	819.87817	2.86%		16.2 2s
*	1340	369		17		840.0527778	819.87817	2.40%		16.2 2s
*	1657	410		21		839.5361111	821.14220	2.19%		16.1 2s

Cutting planes:

Gomory: 6

Cover: 31

Clique: 6

MIR: 11

StrongCG: 1

Flow cover: 29

Inf proof: 9

Zero half: 7

Explored 4371 nodes (64880 simplex iterations) in 4.30 seconds

Thread count was 2 (of 2 available processors)

Solution count 10: 839.536 840.053 844.053 ... 909.461

Optimal solution found (tolerance 0.00e+00)

Best objective 8.3953611111111e+02, best bound 8.3953611111111e+02, gap 0.0000%

## 8.4 ILP2 TI Logfile Beweis Optimum

Gurobi 8.0.0 (win64, .NET) logging started 02/19/19 00:20:12

Academic license - for non-commercial use only

Optimize a model with 424 rows, 1853 columns and 16528 nonzeros

Variable types: 1 continuous, 1852 integer (1852 binary)

Coefficient statistics:

Matrix range [1e+00, 4e+03]

Objective range [8e-03, 4e+01]

Bounds range [1e+00, 3e+04]  
RHS range [1e+00, 1e+00]

MIP start produced solution with objective 964.706 (0.02s)  
Loaded MIP start with objective 964.706

Presolve removed 40 rows and 88 columns  
Presolve time: 0.03s  
Presolved: 384 rows, 1765 columns, 11256 nonzeros  
Variable types: 1 continuous, 1764 integer (1764 binary)

Root relaxation: objective 6.907556e+02, 383 iterations, 0.01 seconds

Nodes		Current Node			Objective Bounds			Work	
Expl	Unexpl	Obj	Depth	IntInf	Incumbent	BestBd	Gap	It/Node	Time
0	0	690.75556	0	168	964.70556	690.75556	28.4%	-	0s
0	0	690.96829	0	166	964.70556	690.96829	28.4%	-	0s
0	0	690.98798	0	157	964.70556	690.98798	28.4%	-	0s
0	0	692.36390	0	172	964.70556	692.36390	28.2%	-	0s
0	0	693.89350	0	185	964.70556	693.89350	28.1%	-	0s
0	0	698.33487	0	171	964.70556	698.33487	27.6%	-	0s
0	0	698.42739	0	168	964.70556	698.42739	27.6%	-	0s
0	0	698.50559	0	174	964.70556	698.50559	27.6%	-	0s
0	0	698.50682	0	164	964.70556	698.50682	27.6%	-	0s
0	0	698.50682	0	163	964.70556	698.50682	27.6%	-	0s
0	2	698.50682	0	163	964.70556	698.50682	27.6%	-	0s
*	406	391	77		908.5305556	706.77031	22.2%	27.1	1s
*	409	391	78		905.0833333	706.77031	21.9%	27.0	1s
*	523	453	70		893.2722222	706.77031	20.9%	25.5	1s
...									
2411516	56756	infeasible	43		793.03327	787.64769	0.68%	24.5	2025s
2417487	54700	infeasible	41		793.03327	787.75771	0.67%	24.5	2030s
2423749	52273	791.32002	42	61	793.03327	787.88871	0.65%	24.5	2035s
2431087	49493	789.34532	47	67	793.03327	788.04068	0.63%	24.5	2040s
2437662	47016	cutoff	39		793.03327	788.18347	0.61%	24.5	2045s
2444518	44233	cutoff	40		793.03327	788.34921	0.59%	24.5	2050s
2451771	41217	cutoff	42		793.03327	788.52935	0.57%	24.5	2055s
2458979	38015	cutoff	46		793.03327	788.73540	0.54%	24.5	2060s
2466260	34742	cutoff	46		793.03327	788.95485	0.51%	24.4	2065s
2473589	31098	cutoff	33		793.03327	789.22517	0.48%	24.4	2070s
2481710	27111	cutoff	39		793.03327	789.52778	0.44%	24.4	2075s

2489722	22790	cutoff	30	793.03327	789.88973	0.40%	24.4	2080s
2498399	17535	cutoff	29	793.03327	790.39028	0.33%	24.4	2085s
2505731	12390	792.38333	43 44	793.03327	790.98653	0.26%	24.3	2090s
2515669	3824	cutoff	44	793.03327	792.27222	0.10%	24.3	2095s

Cutting planes:

Gomory: 3

Clique: 15

MIR: 9

Flow cover: 1

Zero half: 8

Lazy constraints: 491

Explored 2519351 nodes (61113883 simplex iterations) in 2096.95 seconds

Thread count was 4 (of 4 available processors)

Solution count 10: 793.033 793.033 793.033 ... 798.503

Optimal solution found (tolerance 1.00e-04)

Best objective 7.930332686883e+02, best bound 7.929544059760e+02, gap 0.0099%

## 8.5 Test GA Parallel

Datensatz	Gerät	Anzahl parallele Instanzen	Anzahl Generationen
20VBD	W541	1	990476
20VBD	W541	1	998481
20VBD	W541	3	783694
20VBD	W541	3	797742
20VBD	W541	3	808555
20VBD	W541	4	693247
20VBD	W541	4	693239
20VBD	W541	4	687536
20VBD	W541	4	708781
20VBD	W541	5	612416
20VBD	W541	5	631978
20VBD	W541	5	630451
20VBD	W541	5	600960
20VBD	W541	5	634764
20VBD	W541	6	551188
20VBD	W541	6	538390
20VBD	W541	6	543666
20VBD	W541	6	539619
20VBD	W541	6	544066
20VBD	W541	6	509520
20VBD	W541	7	472738
20VBD	W541	7	473637
20VBD	W541	7	474337
20VBD	W541	7	470861
20VBD	W541	7	474749
20VBD	W541	7	482850
20VBD	W541	7	483127
20VBD	W541	8	420640
20VBD	W541	8	421648
20VBD	W541	8	421163
20VBD	W541	8	418552
20VBD	W541	8	421164
20VBD	W541	8	419403
20VBD	W541	8	418289
20VBD	W541	8	420895
20VBD	Server	1	528091
20VBD	Server	2	577208
20VBD	Server	2	576160
20VBD	Server	3	403368
20VBD	Server	3	323515
20VBD	Server	3	429989