Analyse und Adaptierung existierender GATSP-Lösungsverfahren für das Problem der Optimierung der Aktivitätenanordnung mit Routenbestimmung

Bachelor's Thesis von

Julind Mara

an der Fakultät für Informatik
Software-Entwurf und -Qualität (SDQ)

Erstgutachter:	Prof. Dr. Ralf Reussner
Zweitgutachter:	Prof. Dr. Andreas Oberweis
Betreuendere Mitarbeiterin:	M.Sc. Alexandra Wins

22.02.2021 – 22.06.2021

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.

PLACE, DATE

. . . . . .

(Julind Mara)

Zusammenfassung

TODO

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Motivation

Heutzutage spielt Mobilität eine entscheidende Rolle über verschiedene Alltagssituationen hinaus. Die räumliche Entfernung zwischen Wohn-, Arbeits-, Freizeit- und Kauforten führt zu einem steigendem Mobilitätsbedürfnis und folglich zu einer zunehmender Bewegung zu diesen Zielen. Mobilität stellt sich als etwas Unumgängliches. Diese Entwicklung wird in der verschiedensten angebotenen Arten der Fortbewegung gezeigt. Das breite Angebot der öffentlichen Verkehrsmittel und die privaten Mobilitätslösungen erleichtern den Verkehr und ermöglichen das Erreichen auch von den weit entfernten Zielorten. Diese breite Palette an Wahlmöglichkeiten der Transportmittel wirkt auf die andere Seite als unübersichtlich und erschwert eine genaue Planung der Aktivitäten des Alltags, die räumlich entfernt stattfinden. Die eigenen Tagesaktivitätsplänen stehen in einer ständigen Wechselbeziehung mit den Verkehrsmodalitäten, die diese ermöglichen.

Für viele Tagesabläufe wird das Auto meist als alternativlos angesehen und das entspricht einem Akzeptanzproblem in Bezug auf öffentliche bzw. alternative Mobilitätslösungen. Das liegt im Grunde, dass das Einspielen der öffentlichen Verkehrsmittel in einem manuell-erfassten Alltagsplan ein Unsicherheitsgefühl mitbringt und für viele Menschen daher als unzuverlässig gesehen wird. Darüber hinaus kommen in einer Alltagsplanung besonders zeitkritische Aufgaben mit einer hohen Wahrscheinlichkeit, was so ein Zusammenspiel noch zweifelhafter macht.

Ein automatisiertes Tagesplanungssystem für das Einordnen von alltäglichen Aktivitäten und Routen könnte wesentlich sein, um die Unsicherheit mit solchen Mobilitätslösungen beseitigen zu können. Im Allgemein ist es hoch kompliziert eine vollständige Tagesplanung zu erstellen, wo viele Aktivitäten durch die öffentlichen Verkehrsmittel erledigt werden sollen. Noch schwieriger ist es, wenn so eine Planung durch eine Kombination von privaten alternativen Mobilitätslösungen wie: Fahrrad, e-Scooter, usw. und öffentlichen Transportmitteln ermöglicht werden soll. Das liegt daran, dass der öffentliche Nah-/Fernverkehr nur in bestimmten Zeitpunkten des Tages zur Verfügung steht und somit die Flexibilität im Vergleich zu privaten Mobilitätslösungen deutlich niedriger ist. Mit einem solchen System sollte eine Tagesplanung einfacher und dynamischer für die Endnutzer gemacht werden, damit die Ungewissheit in Bezug auf öffentlichen Verkehr reduziert wird.

1.2 Zielsetzung

Um diese Verknüpfung herzustellen, soll im Rahmen dieser Bachelorarbeit ein Verfahren entwickelt werden, der die bestmögliche Anordnung von Aktivitäten sowie die Routen zwischen diesen bestimmt. Es soll dabei zwischen festen und flexiblen Aktivitäten differenziert werden.

Feste Aktivitäten entsprechen Anforderungen und Verpflichtungen der Menschen und haben fest definierte Zeiten. Im Gegensatz dazu können flexible Aktivitäten an einem beliebigen Zeitpunkt durchgeführt werden. Die letzteren sollen je nach benutzerdefinierten Kriterien bestmöglich in den Tag eingeplant werden. Das Lösen des Problems der Aktivitätenanordnung mit Routenbestimmung ist Äquivalent zu der Lösung des generisches asymmetrisches Problem des Handlungsreisenden mit Zeitfenstern (*generic asymetric traveling salesman problem with time windows* - GATSPTW). Es wurden bereits Vorarbeiten in unterschiedlichen Forschungsbereichen wie Tourismus, Gesundheitswesen und Personaleinsatzplanung geleistet.

Im Rahmen der Bachelorarbeit soll im ersten Schritt eine Literaturrecherche von existierenden Lösungsansätzen durchgeführt werden. Diese sollen im zweiten Schritt auf Erweiterbarkeit und Anwendbarkeit auf das Problem der Aktivitätenanordnung mit Routenbestimmung geprüft werden. Abschließend soll der Ansatz identifiziert und implementiert werden, der bezüglich Performance und Lösungsqualität die besten Ergebnisse liefern kann.

1.3 Aufbau der Arbeit

2 Grundlagen

2.1 Begriffserklärungen

2.1.1 Definitionen von problembezogenen Terme

In diesem Abschnitt werden etliche problembezogene Terme definiert, die für das Verständnis der weiteren Kapitel wichtig sind.

Term	Definition
Flexible Aktivität	Eine flexible Aktivität in der Alltagsplanung bezeichnet eine Aktivität, die keine Erledigungszeitfenster besitzt und somit jederzeit innerhalb 24 Stunden erfüllt werden soll.
Feste Aktivität	Eine feste Aktivität in der Alltagsplanung ist eine Aktivität, die mit einem Zeitfenster versehen wird und daher nur innerhalb dieses Fensters erledigt werden soll.
Servicezeit	Die Servicezeit ist eine temporale Angabe für die Dauer einer Aktivität. Jede Aktivität besitzt eine Servicezeit.
Nutzerpräferenzen	Nutzerpräferenzen sind jegliche Wünsche, die Nutzer des Systems für die alltägliche Planung haben und damit in der Routenberechnung einfließen sollen. Solche Präferenzen können unter der Kategorie der öffentlichen Verkehrsmittelwünsche fallen wie z. B. Vorlieben für schnelle Verbindungen, niedrige Kosten, usw. oder unter der Kategorie der privaten Mobilitätslösungsmöglichkeiten des Nutzers wie z. B., ob der Nutzer ein Fahrrad, e-Scooter, usw. besitzt oder nicht.

2.1.2 Generisches asymmetrisches Problem des Handlungsreisenden mit Zeitfenstern

Im folgenden Abschnitt wird das generische asymmetrische Problem des Handlungsreisenden mit Zeitfenstern erläutert. Es werden dabei die einzelne Teilen des Begriffes präzisiert und damit langsam auf der vollen Erklärung aufgebaut.

2.1.2.1 TSP

Das Problem des Handlungsreisenden (eng. Traveling Salesman Problem) kurz TSP ist ein kombinatorisches Optimierungsproblem der theoretischen Informatik. Es ist ein *NP* vollständiges Problem. Zusammengefasst besteht das Problem aus einer zweiteiligen Aufgabe eines Besuchers, oftmals Handlungsreisende genannt [App+06]:

1. Es soll eine passende Reihenfolge für den Besuch mehrerer Orte gewählt werden, sodass diese Tour ein zu besuchenden Ort nur einmal enthält und dort endet, wo es angefangen hat. Dieser Ort wird oft als Depot genannt.

2. Zusätzlich soll diese Route die kürzestmögliche Reisestrecke der Tour sein.

Mathematische Modellierung Für dieses Problem sind zwei mathematische Modellierungen des Problems möglich: es kann als ein ganzzahliges lineares Programm oder als ein gewichteter Graph modelliert werden. Eine Modellierung mittels Graphen lässt sich verständlicher und anschaulicher beschreiben was für die Problemerklärung wertvoller ist.

In der Graphentheorie ist ein Graph G eine Menge von Gegenständen zusammen mit der zwischen den Gegenständen bestehende Verbindungen. Ein Gegenstand wird als ein Knoten bezeichnet und die Menge aller Knoten von G wird mit V notiert. Die Verbindungen zwischen den Knoten des Graphs heißen Kanten und die Menge aller Kanten ist mit E versehen. Rein mathematisch ist ein Graph [Wil] ein geordnetes Paar $G = (V, E)$ mit

$$E \subseteq \{\{a, b\} \mid a, b \in V \text{ und } a \neq b\} \tag{2.1}$$

Solch ein Graph wird als symmetrisch bzw. ungerichtet bezeichnet.

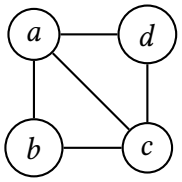


Abbildung 2.1: Beispiel eines symmetrischen Graphen

Damit werden durch die Graphenmodellierung folgende TSP Begriffe folgenderweise abgebildet:

- **Ort** Ein zu besuchender Ort wird als ein Knoten abgebildet.

- **Strecke** Eine Strecke zwischen zwei Städten, wird als eine Kante im Graph abgebildet. Zusätzlich heißen zwei Orten bzw. Knoten a, b benachbart wenn $(a, b) \in V$.

Außerdem wird für das TSP die Graphenmodellierung um eine Dreiecksfunktion

$$C : V \times V \to \mathbb{R} \text{ wo } \forall a, b, d \in V, c(a, b) + c(b, d) \geq c(a, d) \tag{2.2}$$

erweitert, die für jede Strecke eine Zahl zuordnet. Diese Funktion heißt Kostenfunktion und das davon ergebende Resultat stellt die Kosten oder Gewichte einer Kante dar. Die Kosten können

i.d.F als die Streckenlänge zwischen zwei Städten interpretiert werden. Somit bildet sich ein gewichteter Graph.

Die erste Aufforderung des TSP (1) kann somit als eine mathematisches Problem beschrieben werden. Dieses Problem wird in Bezug auf die Graphentheorie dem Hamiltonkreis Problem zugeordnet. Ein Hamiltonkreis ist ein geschlossener Pfad in einem Graphen, der jeden Knoten genau einmal enthält und die Suche, ob ein solcher Pfad mit einem festgelegten Startpunkt existiert, ist das Hamiltonkreisproblem [Wei03]. Ein Graph, der einen solchen Hamiltonkreis besitzt, nennt man auch ein Hamiltongraph. Es ist möglich, dass ein Graph mehrere solche Hamiltonkreise enthält.

Die zweite Aufforderung des TSP (2) kann mathematisch als ein Minimierungsproblem des Hamiltonkreisproblems beschrieben werden. Sei H die Menge der Hamiltonkreise bzw. Hamiltonpfade eines Graphen mit einem festgelegten Startpunkt $s \in V$. Ein Pfad hat die Form

$$\pi = a \rightarrow b \rightarrow ... \rightarrow c \mid a, b, c \in V \tag{2.3}$$

und ist eine geordnete Liste von Knoten, die mit dem Richtungsanzeigeoperator "→"versehen ist.

Zu einem Pfad π sei $|\pi|$ seine Länge und π_0 der erste Knoten, π_1 der zweite, usw. Im Fall, dass $\pi \in H$ ist, ist $\pi_0 = \pi_{|\pi|-1} = s$ und mit der im 2.2 definierte Kostenfunktion C ist dann die folgende Einschränkung

$$\min_{\forall \pi \in H} \sum_{i=0}^{|\pi|-2} c(\pi_i, \pi_{i+1}) \tag{2.4}$$

genügend um die zweite Aufforderung des TSP zu erfüllen.

2.1.2.2 ATSP

Ein anderes TSP Typ stellt sich das asymmetrische TSP (ATSP) dar. Im Unterschied zu dem symmetrischen TSP ist der darin liegende Graph ein asymmetrischer Graph.

Mathematische Modellierung Ein asymmetrischer Graph ist wiederum ein geordnetes Paar $G = (V, E)$, wo

$$E \subseteq \{(a, b) \mid a, b \in V \text{ und } a \neq b\}. \tag{2.5}$$

Anders als in (2.1) enthält E geordnete Paare als Kanten. Ein solcher Graph wird als asymmetrisch bzw. gerichtet bezeichnet.

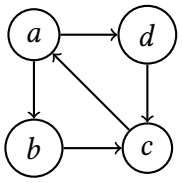


Abbildung 2.2: Beispiel eines asymmetrischen Graphen

2.1.2.3 GATSP

Die folgende GATSP Definition basiert auf [LMW93].

Eine Erweiterung des ATSPs ist der generalisierte ATSP (GATSP). Dabei handelt es sich, um den kostengünstigsten Hamiltonkreis zwischen geeigneten Partitionen des ATSP Graphen zu finden, mit einem wiederum festgelegten Startpunkt $s \in V$.

Um das GATSP vollständig mathematisch formulieren zu können, wird die Definition der Partition und der g-Tour eingeführt.

Mathematische Modellierung Sei $G = (V, E)$ wie gewöhnlich und $|V| = \kappa$. Eine Menge C wird als eine Partition bezeichnet, wenn

$$\forall c \in C \text{ gilt, dass } \sum_{c' \in C} \mathbf{1}_{\{c'=c\}} = 1. \tag{2.6}$$

Eine geeignete Partitionierung $\mathcal{C}$ von G auf $C_0, C_1, ..., C_n$ Partitionen, wo das GATSP operational ist, folgt wie unten:

$$C_0 = \{s\} \tag{2.7a}$$
$$\forall i \in \{1, ..., n\}, C_0 \cap C_i = \emptyset \tag{2.7b}$$
$$\bigcup_{i=0}^{n} C_i = V \tag{2.7c}$$

Daraus lässt sich feststellen, dass in (2.7a) der Startpunkt allein seine eigene Partition bildet und mit (2.7b), dass er in keinen anderen Partitionen sich befindet. Mit (2.7c) ist die Vereinigung aller Partitionen die Menge aller Knoten. Unter diesen Einschränkungen können überlappende oder eigenständige Partitionen die Knotenmenge V bilden.

g-Tour Ein Pfad $\gamma = v_0 \rightarrow v_1 \rightarrow ... \rightarrow v_l$ bezeichnet eine g-Tour, wenn er folgende Eigenschaften erfüllt:

$$v_0 = v_l = s \tag{2.8a}$$
$$\forall j, k \in \{1, ..., l\}, v_j \neq v_k \tag{2.8b}$$
$$n + 1 \leq |\pi| \leq \kappa + 1 \tag{2.8c}$$
$$\forall m \in \{1, ..., n\}, C_m \cap \{v_1, v_2, ..., v_{l-1}\} \neq \emptyset \tag{2.8d}$$

Unter (2.8a) wird der Startpunkt gleich dem Endpunkt der Tour gesetzt, und mit (2.8b) soll die Tour keine wiederauftretende Knoten enthalten. Die Gleichung (2.8c) in Kombination mit (2.8d) garantieren, dass die Tour mindestens einmal alle Partitionen durchgeht; genauer sichert (2.8c) die Länge der Tour, um mindestens $n + 1$ zu sein und (2.8d), dass der Pfad mindestens ein Knoten aus jeder Partition beinhaltet.

Sei Γ die Menge aller g-Touren von G. Die GATSP Optimierungsproblem umfasst die Suche nach dem kostengünstigsten Pfad $\gamma \in \Gamma$. Mathematisch, ähnlich zu (2.4), ist es durch die Einschränkung

$$\min_{\forall \gamma \in \Gamma} \sum_{i=0}^{|\gamma|-2} c(\gamma_i, \gamma_{i+1}) \tag{2.9}$$

ausgedrückt.

2.1.2.4 GATSP mit Zeitfenstern

Die folgende Definition der GATSP mit Zeitfenstern orientiert sich nach [Duc19].

Eine zusätzliche Erweiterung des GATSP stellt sich das GATSP mit Zeitfenstern (GATSPTW). Hierbei wird eine ergänzende Aufforderung dem Handlungsreisenden gestellt, wodurch, die mit Zeitfenstern versehenden Knoten, zeitliche Bedingungen bezüglich dem Besuch liegen. Der Handlungsreisende kann in einem Knoten, außer dem Startknoten, früher als die Eintrittszeit des Knotens oder später als die Austrittszeit des Knotens hineingehen, jedoch zählt so ein Knotenaufenthalt nicht als ein gültiger Besuch. Weiterhin ist das Warten bei einem Knoten innerhalb oder außerhalb seiner Zeitfenster möglich.

Zusätzlich kann ein Knoten eine Servicezeit besitzen. So eine Servicezeit stellt dem Handlungsreisenden eine erforderliche Mindestzeitdauer an dem Besuch eines Knotens.

Das GATSPTW Optimierungsproblem ist das Finden eines kostengünstigsten Hamiltonkreises zwischen den Partitionen des asymmetrisches Graphen. Dabei muss das Besuchen eines Knotens nur innerhalb ihres zugeordneten Zeitfensters erfolgen und im Fall, dass der Knoten eine Servicezeit besitzt, muss die Besuchsdauer nicht außerhalb des Zeitfensters liegen.

Mathematische Modellierung Sei $G = (V, E), \mathcal{C}$ eine geeignete Partitionierung von G und $s \in V$ der Startknoten. Hierbei wird jedem Knoten $v \in V$ ein Zeitfenster $[\mathbf{E}_v, \mathbf{A}_v]$ zugeordnet: $\mathbf{E}_v$ steht für die Knoteneintrittszeit, $\mathbf{A}_v$ für die Knotenaustrittszeit. Zusätzlich wird für jeden Knoten auch eine Servicezeit $\mathbf{S}_v$ vorgesehen. O.B.d.A wird hier angenommen, dass das Zeitfenster $\mathbf{E}_v, \mathbf{A}_v \in \mathbb{N}_0$ und Servicezeit $\mathbf{S}_v \in \mathbb{N}_0$.

Um das GATSPTW Problem mathematisch modellieren zu können, muss die Definition eines zeitgedehnten Netzwerkmodell [Duc19] $\mathcal{Z}$ eingeführt und die vorgenannte g-Tour leicht angepasst werden. Diese Anpassung ermöglicht, dass die neue g-Tour kompatibel zu dem zeitgedehnten Netzwerkmodell ist. So was wird als eine g_z-Tour definiert.

Zeitgedehntes Netzwerk Zu einem Graphen G mit einem Knoten-Zeitfenster Zuordnung wird das dazugehörige zeitgedehnte Netzwerk $\mathcal{Z}_G = (\mathcal{V}, \mathcal{E})$ definiert.

Die Knotenmenge $\mathcal{V}$ ist definiert als

$$\mathcal{V} = \{(v, t) \mid v \in V, t \in [\mathbf{E}_s, \mathbf{A}_s]\}. \tag{2.10}$$

Darin sind alle möglichen Knoten-Zeit Zustände bzw. Kombinationen enthalten, in denen der Handlungsreisende sich befinden kann, die vom Startknoten ausgehend möglich sind. Des Weiteren wird eine Fahrtdauerfunktion

$$\tau : \mathcal{V} \times \mathcal{V} \to \mathbb{N}_0 \tag{2.11}$$

definiert, was die Fahrtdauer zwischen den zwei Knoten einer Kante berechnet.

Schließlich ist die Kantenmenge $\mathcal{E}$ folgenderweise definiert:

$$\mathcal{E} = \{((a, t_a), (b, t_b)) \mid (a, t_a), (b, t_b) \in \mathcal{V}, (a, b) \in E \text{ und } P((a, t_a), (b, t_b))\} \tag{2.12}$$

wo

$$P : \mathcal{V} \times \mathcal{V} \mapsto \texttt{bool} \text{ mit } P((a, t_a), (b, t_b)) = \begin{cases} t_b = t_a + \tau(a, b) \mapsto \texttt{true} & , a \neq b \\ t_b = t_a + 1 \mapsto \texttt{true} & , a = b \\ \mapsto \texttt{false} & , sonst \end{cases} \tag{2.13}$$

ist.

Durch diese Kantenmengendefinition enthält $\mathcal{E}$ Kanten, wo der Handlungsreisende zwischen den Knoten auch außerhalb ihrer Zeitfenster fahren und ankommen kann. Die Fahrtdauern dieser Kanten respektieren die Fahrtdauerfunktion τ und können das Warteverhalten bei einem Knoten modellieren.

Zu dem Netzwerk wird eine neue geeignete Netzwerkpartitionierung C' anhand C erstellt, indem

$$C' = \{C'_0, C'_1, ..., C'_n\} \tag{2.14}$$

definiert wird, mit

$$C'_i = \{(m, t) \mid (m, t) \in \mathcal{V}, m \in C_i\}. \tag{2.15}$$

g_z-Tour Sei $\mathcal{Z}_G = (\mathcal{V}, \mathcal{E})$ ein zeitgedehntes Netzwerk eines Graphen G mit $|\mathcal{V}| = \kappa$.

Es wird eine Einzigartigkeitsfunktion

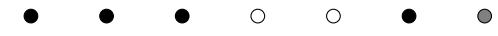
$$\delta : \gamma_z \rightarrow \{v \in V\} \tag{2.16}$$

definiert, die zu einem Pfad γ_z die Menge aller besuchten unterschiedlichen Knoten aus V außer dem Startknoten ausgibt. Z. B. zu einem $\gamma_z = (l, t_0) \rightarrow (l, t_1) \rightarrow (q, t_2) \rightarrow (l, t_3) \rightarrow (r, t_4)$ ist $\delta(\gamma_z) = \{l, q, r\}$.

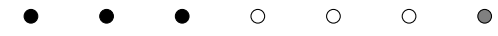
Ein Pfad $\gamma_z = (v_0, t_0) \rightarrow (v_1, t_1) \rightarrow ... \rightarrow (v_l, t_l)$ mit Pfadknoten aus $\mathcal{V}$ und Pfadkanten aus $\mathcal{E}$ bezeichnet sich als eine g_z-Tour von $\mathcal{Z}_G$, wenn es folgende Eigenschaften erfüllt:

1. $v_0 = v_l = s$
2. $\forall j, k \in \{1, ..., l\}$ mit $k - j \geq 1$ wenn $v_j = v_k$ muss $\forall q, r \in \{j, ..., k\}, v_q = v_r$ gelten
3. $n + 1 \leq |\gamma_z| \leq \kappa + 1$
4. $\forall m \in \{1, ..., n\}, C'_m \cap \{(v_1, t_1), (v_2, t_2), ..., (v_{l-1}, t_{l-1})\} \neq \emptyset$
5. $\forall u \in \delta(\gamma_z)$ gilt einer der folgenden:
   - $\mathbf{S}_u > 0 \Rightarrow \exists\{(u_i, t_i), ..., (u_s, t_s)\} \subset \gamma_z$ mit $u_i ... u_s = u$ gilt $t_i, t_s \in [\mathbf{E}_u, \mathbf{A}_u]$, $t_s = t_i + \mathbf{S}_u$ und $\forall(u_j, t_j), (u_{j+1}, t_{tj+1})$ gilt $t_{j+1} - t_j = 1$
   - $\mathbf{S}_u = 0 \Rightarrow \exists u_i \in \gamma_z$ mit $u_i = u$ gilt $t_i \in [\mathbf{E}_u, \mathbf{A}_u]$
6. $t_0 \geq \mathbf{E}_s$ und $t_l \leq \mathbf{A}_s$

Die g_z-Tour ähnelt sich der g-Tour mit dem einzigen Unterschied, dass das Warteverhalten berücksichtigt wird, indem adäquate Wiederholungen der Knoten miteinbezogen werden. Das wird in der Eigenschaft (2) gekennzeichnet. Die visuelle Bedeutung dieser Eigenschaft wird in der Abbildung 2.3 verdeutlicht. Hier werden die v_i Knoten durch Kreise abgebildet, wo unterschiedliche Farben der Kreise unterschiedliche Knoten repräsentieren. Die Eigenschaft (4) garantiert, dass der Handlungsreisende jeden Cluster besucht hat. Durch (5) wird sichergestellt, dass der Handlungsreisende jeden Knoten innerhalb ihres zugewiesenen Zeitfensters, inklusive der Mindestbesuchsdauer des Knotens, besucht hat. Letztlich besagt (6), dass er nicht früher oder später in dem Startknoten sich befindet.



(a) Eine unpassende Anordnung von Knoten.



(b) Eine passende Anordnung von Knoten.

Abbildung 2.3: Zwei Beispiele von einer unpassenden und passenden Knotenanordnung gültig für eine g_z-Tour.

Sei Γ_z die Menge aller g_z-Touren von einem mit Zeitfenstern versehenden Graph G. Die GATSPTW Optimierungsproblem besteht aus der Suche nach dem kostengünstigen Pfad $\gamma_z \in \Gamma_z$. Mathematisch formuliert ist es durch die unten gegebene Einschränkung ausgedrückt:

$$\min_{\forall \gamma_z \in \Gamma_z} \sum_{i=0}^{|\gamma_z|-2} c_z(\gamma_i, \gamma_{i+1}) \tag{2.17}$$

wo

$$C_z : \mathcal{V} \times \mathcal{V} \to \mathbb{R} \text{ mit } c_z((a, t_a), (b, t_b)) = c(a, b) \tag{2.18}$$

ist.

2.2 Stand der Technik

2.2.1 Literatur

Wie in der 1.2 Kapitel aufgelistet wurden, haben in verschiedenen Industrie-, Mathematik- und Dienstleistungsbereichen Forschungen über nahezu gleiche Probleme stattgefunden [CLQ16] [Bai+19] [AFG01] [KG12] [Hel14] [GK10], die suboptimale aber effiziente Lösungswege und Techniken zum Lösen von GATSPTW **ähnlichen** Problemen vorgestellt haben.

In Recherchen von ausschließlich kombinatorische Optimierungsprobleme der Operations Research wurde bedauerlicherweise für genau dieses Problem sehr wenig bis auf nur in einer Forschung recherchiert. Die einzige Literatur, nämlich die *A branch-and-cut algorithm for the generalized traveling salesman problem with time windows*[Yua+20] die mit dem GATSPTW Problem beschäftigt, weist auch auf diese Problematik, beziehungsweise Mangel an Recherche hin. Darüber hinaus stellen die Autoren dieser Recherche klar, dass sie die ersten sind, die exakt dieses Thema untersucht und eine Lösungsalgorithmus dafür vorgelegt haben. Andere beigebrachte Lösungsalgorithmen in den anderen Recherchen, sind für Variationen oder vereinfachten Versionen der GATSP mit Zeitfenstern vorgegeben, wie zum Beispiel für die ATSP mit Zeitfenstern[AFG01] oder für die GATSP ohne Zeitfenstern[KG12].

Weitere Untersuchungen in der Feld der Personaleinsatzplanung[CLQ16] waren auch von Interesse. Dort gibt es viele Szenarien, in denen das Personal Aufgaben an verschiedenen Standorten erledigen muss und daher ein Transport und Routenplanung braucht. Beispiele für solche Szenarien sind Krankenschwestern, die Patienten zu Hause besuchen, Techniker, die Reparaturen bei Kunden durchführen, usw. Damit ist eine zeitliche und örtliche Planung von Bedeutung. Wie oben erwähnt hat diese Betrachtung das gleiche Ergebnis geliert.

2.2.2 Familie der Lösungsalgorithmen

Aus den oben erwähnten Recherchen über die Lösungsalgorithmen zu den GATSPTW ähnlichen Problemen haben sich zwei Klassen von Lösungsalgorithmen ergeben:

1. **Reduktion auf TSP**
   Eine Lösungsklasse von solchen Problemen war die systematische Reduktion des Problem auf TSP. Dieser Lösungsweg ermöglicht, dass das Lösen des durch die Reduktion erhaltende TSPs einfacher wird. Das liegt daran, dass für das Lösen davon, bereits state of the art exakte (`Concorde`[App+]) oder sehr schnell und approximierende (`LKH`[Hel00]) TSP Löser gibt. Die allgemeine Reduktion von GATSPTW ähnlichen Problemen erfolgte wie folgt:

$$\text{GATSPTW} \xrightarrow{\text{Zeitdehnung}} \text{GATSP} \xrightarrow{\text{Konkretisierung}} \text{ATSP} \xrightarrow{\text{Symmetrisierung}} \text{TSP}$$

   a) Die Zeitdehnung sorgt dafür dass, die Knoten ihre Zeitfensterzuordnung verlieren und damit in einzelnen Knoten einer festen Zeitinstanz zerteilt werden. Dieser Prozedur ähnelt sich stark dem Zeitdehnungsverfahren in 2.1.2.4.

   b) Die Konkretisierung ermöglicht dass, die Clusters von Knoten entfernt werden und jede Knoten dann alleinstehend bleibt. Zusätzlich muss das Lösen dieser neueren Graph gleich der Lösung des GATSP Problems entsprechen. Verschiedene solche Verfahren und deren Performanzen werden in [Ben+03] dargestellt.

   c) Die Symmetrisierung des Graphen ermöglicht dass, der Graph symmetrisch wird und dabei alle entstandene Kanten symmetrisch sind. Ein solches einfaches Verfahren wird in [JV83] vorgelegt.

2. **Reduktion auf GATSP und direktes Lösen**
   Eine andere Lösungsart für solche Probleme, ist der Einsatz von Algorithmen zum Lösen des reduzierten GATSP Problems. Dieser Lösungsweg erfordert nur den Schritt der Zeitdehnung und dann direktes Einsetzen eines bestimmten Algorithmus. Dabei sind drei verschiedene Techniken von Interesse:

   ILPs Das Lösen mittels ganzzahlige lineare Optimierungsalgorithmen (engl. ILPs) war eine beherrschende Technik, was in vielen Forschungen auf GATSPTW ähnlichen und in [Yua+20] (die genau mit GATSPTW befasst) verwendet wurde, um eine approximierende Lösung zu bekommen. Hierbei wurde die Branch-and-cut Familie von Algorithmen verwendet, die zu der Familie von Branch-and-bound Algorithmen der ILPs gehört.

   Metaheuristisch Eine andere Ansatz zum Lösen von GATSP Instanzen war eine genetische Algorithmus von [GK10].

   Exakt Ein neuartiges exaktes Verfahren zur Lösung von GATSP Instanzen ist der `GLKH`[Hel14] Löser, der aus einer Erweiterung des state of the art `LKH` Lösers stammt.

2.2.3 Vergleich der Lösungsalgorithmen

Die folgende Effizienzrecherche für die Transformation von GATSP auf TSP basiert auf [KG12].

Forscher schlugen Transformationen von GATSP Instanzen in TSP Instanzen vor. Zunächst einmal scheint die Idee, ein wenig untersuchtes Problem in ein bekanntes zu transformieren sehr positiv. Dieser Ansatz hat jedoch eine sehr geringe Effizienz ergeben, da es exakte beziehungsweise optimale Lösungen der erhaltenen TSP Instanzen erfordert, weil eine suboptimale Lösung eines solchen TSP einer nicht realisierbaren GATSP Lösung entsprechen kann. Gleichzeitig weisen die erzeugten TSP Instanzen eine eher ungewöhnliche Struktur auf, die für die vorhandenen TSP Löser schwierig zum Lösen ist.

Ein effizienterer Ansatz zur genauen Lösung des GATSP ist der Einsatz von ILPs beziehungsweise der Branch-and-cut Algorithmus von [FGT97] und modifizierte Versionen davon. Mit diesen Algorithmen lösten die Forscher mehrere Instanzen der Größenordnung von bis zu 89 Clustern mit 442 Knoten. Da dieser Art von Algorithmen relativ effizient arbeitet, wurde es als Basis der Arbeit von [Yua+20] ausgewählt indem verschiedene Erweiterungen und Modifikationen eingebaut wurden, um den resultierenden Branch-and-cut Algorithmus kompatibel zum Lösen von GATSPTW Instanzen zu machen.

Jedoch ist der obere Algorithmus von [FGT97] und Herleitungen davon schlechter im Vergleich zu dem genetischen Algorithmus von [GK10]. Die Auswertungen davon haben gezeigt, dass dieser metaheuristischer Algorithmus deutlich besser, sowohl hinsichtlich der Lösungsqualität als auch der Laufzeit ist. Zusätzlich kann dieser Algorithmus gleichzeitig symmetrische und asymmetrische GATSP Instanzen lösen.

Abschließend ist der `GLKH` Löser auch sehr versprechend. Im Vergleich zu dem genetischen Algorithmus von [GK10] erreicht dieses heuristische Verfahren für viele Lösungen eine bessere Lösungsqualität hingegen erhöhter Laufzeit. Diese längeren Laufzeiten sind für praktikable Zwecken äußerst vernünftig, wenn Lösungsqualität wichtig ist.

3 Problemmodellierung

3.0.1 Notationserklärung

Im weiteren Verlauf sollen diese Notationen folgende Bedeutungen tragen:

Term	Definition
$_{TW}$	Eine tiefgestellter TW zeigt, dass dieses Objekt dem GATSPTW Problem gehört.
$_{T}$	Eine tiefgestellter T zeigt, dass dieses Objekt dem GATSP Problem gehört.
K	Ein K zeigt eine Menge von Knoten.
E	Ein E zeigt eine Menge von Kanten.
C	Ein C zeigt eine Menge von Clustern.

3.0.2 GATSPTW Ansatz

Zu der Aufgabe der Entwicklung eines automatisierten Tagesplanungssystem für das Einordnen von alltäglichen Aktivitäten und Routen kann größtenteils das GATSPTW Problem zugeordnet werden. Größenteils, da es einen Unterschied zu dem allgemeinen GATSPTW Problem gibt, in dem die Kosten der Kanten dieses Routingsproblem mobilitätsabhängig sind. Das liegt daran, dass dieses System Präferenzen dem Endnutzer bezüglich der gewünschten Verkehrsmitteln anbieten soll und dadurch können zwei verschieden Orten wegen der unterschiedlichen Mobilitätslösungen unterschiedliche Kosten besitzen.

Damit lässt sich ein einfaches modifiziertes GATSPTW Modell aufbauen, das passend zu diesem Problem ist. Die folgenden semantischen Regeln für einen korrekten Problemmodellaufbau sind:

Knoten → Ein Knoten repräsentiert ein Ort.

Startknoten → Ein Startknoten repräsentiert den Startpunkt der Alltagsplanung. Das ist in den meisten Fällen der Wohnort des Nutzers.

Kante → Eine Kante repräsentiert eine Anschlussmöglichkeit zwischen zwei Orten. Kantenkosten werden, wie oben erwähnt, hier etwas anders als üblich behandelt. Zu den Kanten dieses Problemmodell werden verschiedene Kosten je nach unterschiedlichen Zeitpunkten oder Transportmitteln gesetzt. So was kann zum Beispiel an der Verfügbarkeit der öffentlichen Verbindungen liegen oder an der unterschiedlichen Verbindungskombinationen, die unter verschiedenen Zeitpunkten oder Transportmitteln zustande kommen.

Cluster → Ein Cluster stellt eine Menge an Orten dar, wo eine bestimmte Aktivität ausgeführt werden kann. Die Clustergröße kann je nach Feinheitsgrad der Aktivität variieren: zum Beispiel kann der Cluster zuständig für "*Arbeiten*" nur ein Ort enthalten, nämlich nur den Arbeitsplatz des Nutzers und der Cluster für "*Lebensmittel einkaufen*" mehrere Lebensmittelgeschäftsorten enthalten.

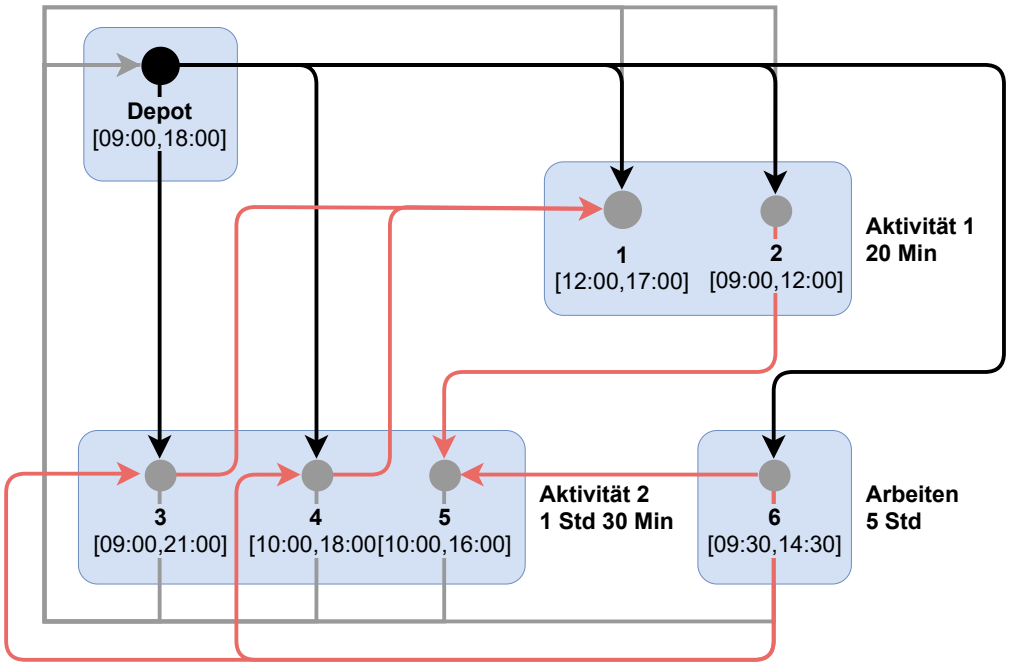


Abbildung 3.1: Beispiel eines GATSPTW Graphen. Schwarze Kanten stehen für eine Anschlussmöglichkeit zwischen der Depotknoten und ein normaler Knoten, die Roten für eine Anschlussmöglichkeit zwischen normaler Knoten und die Grauen für eine Anschlussmöglichkeit von einem normalen Knoten zurück in dem Depotknoten.

Mit dem oben beschrieben Modellaufbauregeln ist es möglich, die Mobilitätsabhängigkeit der Kantenkosten zu entfernen, wenn der Graph eine erweiterte Zeitdehnung durchläuft. Unter dieser Berücksichtigung lässt sich der GATSPTW Problem auf einem einfacheren GATSP Instanz reduzieren, wobei durch diese Zeitdehnung bzw. Reduktion, eine Lösung des GATSP Problems genau eine Lösung des GATSPTW Problems entsprechen wird.

3.0.2.1 Objektmodelle des GATSPTW Problems

Zum weiteren Verständnis der Problemmodellierung und GATSP Problemreduzierung werden hier die Teilobjekte des Modells eingeführt und modelliert. Die Hauptmodellobjekte sind Knoten, Clustern sowie Kanten und ein weiteres Mobilitätslösung-Hilfsobjekt.

Ein Zeitpunkt wird in diesen Modellen o.B.d.A. durch Werte aus der $\mathbb{N}_0$ Menge stammen.

`Cluster`:
```
id
serviceZeit
knoten : List<Knoten>
```

Ein Clusterobjekt bildet ein Cluster und enthält ein `id` Feld, um den Cluster eindeutig zu identifizieren. Zudem hat das Clusterobjekt eine `serviceZeit` Feld, was die Service Zeit für alle Knoten dieser Cluster setzt. Das wird so aufgrund der Allgemeinheit der Aufenthaltsdauer zu einem bestimmen Aktivität in jeglicher Knoten modelliert. Zum Beispiel "*Lebensmittel einkaufen*" dauert genauso lang, egal auf welchem Knoten jemand sich befinden kann. Schließlich, beinhalten das Clusterobjekt alle dazugehörige Knoten in der `knoten` Liste.

`Knoten`:
```
id
zfAnfang
zfEnde
```

Ein Knotenobjekt bildet ein Knoten. Der `id` Feld ist ein Identifikationsfeld die eine eindeutige Variable enthält, um den Knoten zwischen alle anderen Knoten der Graphen zu unterscheiden. Die `zfAnfang` und `zfEnde` bilden zusammen den Zeitfensteranfang und Zeitfensterende des Knotens.

`Kante`:
```
knotenUrsprung
knotenEnde
verbindungsmöglichkeiten : List<Verbindungsmöglichkeit>
```

Ein Kantenobjekt repräsentiert eine Verbindungsmöglichkeit zwischen zwei eindeutigen Knoten. Diese Knoten sind `knotenUrsprung` und `knotenEnde`, wo `knotenUrsprung` den Ursprungsknoten und `knotenEnde` den Zielknoten dieser Kante zeigt. Wie zu sehen ist, hat dieser Kante keine eindeutigen Kosten oder Fahrtdauer wie es üblich für die Kanten eines Graphen ist. Diese Elemente werden in den `verbindungsmöglichkeiten` Feld modelliert, wo die Liste alle möglichen Mobilitätslösungen zwischen diese zwei Knoten enthalten ist. Eine Mobilitätslösung wird durch eine Verbindungsmöglichkeit repräsentiert.

`Verbindungsmöglichkeit`:
```
dauer
kosten
transportmittel
abfahrtzeit
```

Eine Verbindungsmöglichkeit zwischen zwei Knoten zeigt in der Kontext der GATSPTW Problem eine Transportlösung zwischen zwei Orten. Der `dauer` Feld zeigt die Dauer und der `kosten` Feld die Kosten dieser Transportlösung. Der `transportmittel` Feld enthält den Namen dieser Lösung. Der `abfahrtzeit` Feld zeigt, soweit vorhanden, die Abfahrtzeit dieser Transportmittel. Wenn keine `abfahrtzeit` gesetzt wurde, bedeutet, dass diese Transportmöglichkeit jederzeit zur Verfügung steht. Solche Transportmittel sind wie z. B. das Fahrrad, e-Scooter, Fuß, usw.

3.0.3 Reduktion auf GATSP

In diesem Abschnitt wird der Weg zur Reduzierung von einer GATSPTW Probleminstanz mit Mobilitätsabhängige-Kantenkosten auf einem allgemeinen GATSP Probleminstanz gezeigt. Hierbei wird für jede GATSPW Modelleinheit die passende Umwandlung gezeigt, damit die Lösung dieses neu entwickelten GATSP Problem äquivalent zu der Lösung des ursprünglichen GATSPTW Problem ist.

Diese Umwandlung generiert dabei zwei neue Modelle: die GATSP Knoten und die GATSP Kanten. Die anderen erforderlichen Objektmodelle (das Cluster- und Mobilitätslösungobjekt) sind äquivalent zu den Objekten des GATSPTW Problems und somit ist eine Transformation davon nicht notwendig um den GATSP Problem vollständig zu modellieren.

3.0.3.1 Neue Objektmodelle des GATSP Problems

`GATSP Knoten`:

```
id
zeitInstanz
zfAnfang
zfEnde
```

Ein GATSP Knoten bildet ein Knoten, die zu dem GATSP Problem passend ist. Außer dem `zeitInstanz` Feld ist die Bedeutung aller anderen Felder identisch zu dem Knotenobjekt des GATSPTW Problem. Dieses neue `zeitInstanz` Feld besitzt dabei ein konkreter und fester Zeitpunkt.

`GATSP Kante`:

```
knotenUrsprung
knotenEnde
verbindungsmöglichkeit
```

Ein GATSP Kante stellt eine Kante zwischen zwei GATSP Knoten dar. Der einzige Unterschied zwischen diesem Modell und dem GATSPTW Modell liegt daran, dass zwischen zwei GATSP Knoten nur eine eindeutige Verbindungsmöglichkeit liegt. Das wird in dem `verbindungsmöglichkeit` Feld gesetzt. In der GATSP Kontext kann eine Verbindungsmöglichkeit nicht nur Transportlösungen enthalten, sondern auch das Warteverhalten repräsentieren. Der Grund dafür ist, dass GATSP Knoten Zeitinstanzen enthalten und das Warten eine verbindende Aktion zwischen zwei aufeinanderfolgenden Zeitpunkten ist.

3.0.3.2 Transformationsalgorithmen

In diesem Abschnitt werden die notwendige Transformationsalgorithmen für das Generieren diesen zwei neuen Objekten gegeben. Dabei wird der Algorithmus dargestellt und unten eine schrittweise Erklärung der Arbeitsweise davon.

Algorithmus 1: Transformationsfunktion für die Generierung von GATSP Knoten aus GATSPTW Knoten

Input: GATSPTW Knotenmenge K_{TW}, Startknoten s_{TW}
Output: GATSP Knotenmenge K_T

```
1  begin
2      K_T ← ∅
3      start ← s_TW.zfAnfang
4      ende ← s_TW.zfEnde
5      foreach k_TW ∈ K_TW do
6          foreach Zeitinstanz ζ ∈ [start, ende] do
7              k_T ← neue GATSP Knoten (k_TW.id, ζ, k_TW.zfAnfang, k_TW.zfEnde)
8              füge k_T in K_T
9          end
10     end
11 end
```

In diesem Algorithmus werden GATSP Knoten aus GATSPW Knoten transformiert. Es wird dabei eine Zeitexpandierung insgesamt in $O(|V_{TW}| \cdot n)$ erfolgen, wobei für jeden Knoten dem Graphen iterativ ein GATSPW Knoten genommen wird und eine neue GATSP Knoten anhand davon erstellt wird. Es werden alle Eigenschaften der alten Knoten bis auf dem Zeitinstanz Feld übernommen. Dieses Feld nimmt Werten aus dem Bereich von dem Zeitfensteranfang und dem Zeitfensterende des Startknotens.

Dieses Verfahren ermöglicht, dass das frühere Ankommen und Warten an einem Knoten modelliert werden kann. Würden die zeitgedehnten Knoten nur auf ihrem Zeitfensterbereich expandiert, wäre es dann unter Umständen nicht mehr möglich eine Verbindung zwischen zwei Knoten in verschiedenen Clustern zu finden. Das passiert, wenn es nur eine Abfahrt aus einem zeitgedehnte Knoten zu einem anderen gäbe und:

- dies später als das Zeitfensterende des Ursprungsknotens liegt
- dies früher als der Zeitfensteranfang des Zielknotens ankommt.

Algorithmus 2: Transformationsfunktion für die Generierung von GATSP Kanten aus GATSPTW Kanten

Input: GATSPTW Kantenmenge E_{TW}, GATSP Knotenmenge K_T
Output: GATSP Kantenmenge E_T

```
1  begin
2      E_T ← ∅
3      warte_D ← 1
4      warte_C ← 0
5      warte_V ← neue Verbindungsmöglichkeit (warte_D, warte_C, WARTEN)
6      foreach kU_T ∈ K_T do
7          foreach kZ_T ∈ K_T do
8              δ_UZ ← kU_T.zeitInstanz−kZ_T.zeitInstanz
9              if kU_T.id = kZ_T.id und δ_UZ = 1 then
10                 warteKante ← neue GATSP Kante (kU_T, kZ_T, warte_V)
11                 füge warteKante in E_T
12             end
13             if kU_T.id ≠ kZ_T.id then
14                 e_TW ← findeGATSPTWKante(E_TW, kU_T.id, kZ_T.id)
15                 if e_TW ≠ ∅ then
16                     V ← ∅
17                     foreach v ∈ e_TW.verbindungsmöglichkeiten do
18                         if δ_UZ = v.dauer ∧
                            v.abfahrtzeit ≠ ∅ ⇒ kU_T.zeitInstanz = v.abfahrtzeit ∧
                            V ≠ ∅ ⇒ v.kosten < V.kosten then
19                             V ← v
20                         end
21                         transportKante ← neue GATSP Kante (kU_T, kZ_T, V)
22                         füge transportKante in E_T
23                     end
24                 end
25             end
26         end
27     end
28 end
```

In diesem Algorithmus werden, die zu dem reduzierten GATSP Problem passende GATSP Kanten von den GATSPW Knoten transformiert. Durch dieses Algorithmus wird das Ziel erreicht, Kanten solcher Art zu erstellen, wo die Mobilitätsabhängigkeit der Kosten abgelöst wird und das Warteverhalten modelliert werden kann. Obwohl, dass ein Informationsverlust stattfindet in dem selektiv eine Verbindungsmöglichkeit aus dem Möglichen gewählt wird, erzielt dieses Algorithmus eine korrekte GATSP Modell.

Zunächst wird eine Warteverbindung $warte_V$ mit 0 Kosten und einer Dauer von 1 sowie einem Namen von `WARTEN` erstellt. Diese Verbindungsmöglichkeit wird dann für das Erstellen von Wartekanten zwischen zwei zeitlich aufeinanderfolgenden zeitgedehnte Knoten benötigt. Zwei Knoten gelten als zeitlich aufeinanderfolgend, wenn der zeitliche Unterschied zwischen

deren Zeitinstanzen eins beträgt. Danach werden alle mögliche Knoten in $O(|V_{TW}|^2 \cdot |E_{TW}| \cdot n)$ miteinander kombiniert und auf zwei möglichen Anschlussmöglichkeiten geprüft:

1. Wenn die beiden Knoten zeitlich aufeinanderfolgend sind und die gleiche `id` haben, dann wird eine neue Wartekante zwischen denen erstellt.

2. Wenn die Knoten unterschiedliche `ids` haben, dann wird nach einer GATSPTW Kante gesucht, die diese zwei Knoten verbindet. Falls eine gefunden wird, werden alle Verbindungsmöglichkeiten der Kante auf Gültigkeit abgecheckt. Eine Verbindung ist gültig wenn:

   a) die Verbindungsdauer gleich dem zeitlichen Unterschied zwischen den Zeitinstanzen der Knoten ist.

   b) die Abfahrtzeit der Verbindung gleich der Zeitinstanz des Ursprungsknotens ist, solange eine Abfahrtzeit für diese Verbindung existiert.

   Schließlich wird dann die kostengünstigste Verbindung aus allen gültigen Verbindungen gewählt und damit eine neue GATSP Kante erstellt. Diese neue verbindende Kante wird dann zu der Kantenmenge hinzugefügt.

3.0.4 Beweis zur Lösungsäquivalenz

In diesem Abschnitt wird ein Beweis gegeben, um zu zeigen, dass eine optimale Lösung dieses neu reduzierten GATSP Problems auch eine optimale Lösung des ursprünglichen GATSPTW Problems entspricht.

Die Lösung eines GATSPTW Problems ist mathematisch durch (2.17) definiert und es funktioniert auf einer Basis von g_z-Touren und zeitgedehntem Netzwerken. Die zeitgedehnte Netzwerken sind essenziell GATSP Graphen transformiert von GATSPTW Graphen um die Zeitabhängigkeit der Kanten zu entfernen und die g_z-Touren sind ein mathematisches Einschränkungsmittel, um nur passende Pfade aus dem Graph für die Minimierungsverfahren zu erlauben.

Unter dieser Beobachtung ist es offensichtlich, dass die GATSP Reduktion in 3.0.3 genau eine solches zeitgedehntes Netzwerk aufbaut. Ein Hauptunterschied zwischen der Kantenerstellung des Zeitdehnungsverfahren in (2.12) und der Kantenerstellungsalgorithmus (Algorithmus 2) ist, dass in dem Algorithmus ein extra Kostenminimierugnsschritt zur Verbindungsauswahl stattfindet. Es reicht daher zu zeigen, dass dieser Minimierungsschritt kein negativer Einfluss in der Lösung trägt und zu einer korrekten und optimalen Lösung führt.

Dieser Verbindungsauswahl (Algorithmus 2, Zeilen 17-20) wählt aus allen möglichen verfügbaren Verbindungen für die zeitgedehnte Knoten nur die kostengünstigsten. Somit hat jede nicht wartende Kante in dem GATSP Graphen nur die bestmöglichen Kosten. Zusätzlich bringt dieser extra Schritt keine Strukturänderungen in Bezug auf der Kantenerstellung, da es nur eine Kostenauswahl zwischen Verbindungen mit gleichem Start- und Endknoten ausführt. Damit lässt sich feststellen, dass dieser Informationsverlust wegen der Auswahl keine suboptimale Auswirkung auf der Lösung des GATSPTW Problems hat.

Insgesamt hat dann jede Pfadkomponente der möglichen g_z-Touren des reduzierten Problems garantiert die niedrigsten Kosten und eine Minimierung aller möglichen Touren zu einer optimalen Lösung des GATSPTW Problems entsprechen würde.

4 Systemmodellierung

4.1 Auswahl des Lösungsalgorithmus

Die programmatische Lösung dieses Problems ist äußerst abhängig von dem ausgewählten Lösungsverfahren, da die Daten- und interne Problemmodellierung von einem Löser zu dem anderen viel abweichen. Um so ein System zu entwickeln, ist es wichtig zuerst eine Entscheidung über den Auswahl der möglichen Verfahren zu treffen.

Auf den ersten Blick erscheint die vorgegebene Lösung von [Yua+20] als das geeignetste Verfahren zur Verfügung für das Lösen von einem GATSP Problem mit Zeitfenstern. Diese Idee fällt aber unter genaues Einsehen des Lösungsansatzes und allgemeiner Aufbau der Recherche schwer als ein sehr kompetenter Ansatz im Vergleich zu den anderen Lösungsansätzen wahrzunehmen. Genaues Betrachten der Recherche schafft den Eindruck von einem gehetztes wissenschaftlicher Arbeit, wo viele Module in großen Mengen als Textkopien von der [AFG01] Arbeit stammen. Neuheiten die diese Recherche an dem bereits von [AFG01] eingeführte Branch-and-Cut Lösungsalgorithmus bringt, ist eine schnelle Heuristik um eine obere Schranke zu finden, die zum Begrenzen des Suchraums für das Branch-and-Cut Verfahren dient und leichte Erweiterungen an der Problemmodellierung für die Realisierung von Clusters.

Anderseits wäre das Inkorporieren eines exakten Verfahrens mehr gewünscht. Allgemeine ILP Löser (wie IBMs `CPLEX` oder Quelloffen wie Coin-ORs `CBC`) sowie metaheuristisches Verfahren sind zwar nicht minderwertig, aber die Existenz von zwei schnelle state of the art exakte TSP Löser (`Concorde` und `LKH`) sollte nicht vernachlässigt werden. Als Beispiel kann man erwähnen, dass der leicht modifizierte `LKH` (die `GLKH`), die mit GATSP Probleme umgehen kann, als ein sehr kompetenter Konkurrent gegenüber dem metaheuristischen genetischen Algorithmus von [GK10] steht. Unglücklicherweise existieren keine Daten, die eine direktes Vergleich zwischen der Arbeit von [Yua+20] und eine Lösungstechnik die so ein state of the art Löser einbaut ermöglichen kann.

Ein anderer Grund warum `GLKH` die beste Lösung wäre, ist, dass `GLKH` von der ungewöhnliche Struktur des intern reduzierten TSP Problems nicht negativ beeinflusst wird [Hel14]. Wie in 2.2.3 erwähnt wurde, ist so was für die Mehrheit der state of the art Löser ein großes Problem.

Durch weitere Erwägungen (Codewartbarkeit, Modularität, Möglichkeiten an Problemmodellerweiterungen, usw.) und modellierungsbedingte Einengungen, wurde der Weg zur Reduktion des GATSPTW Problems auf einem GATSP Instanz und dann mittels `GLKH` zu lösen gewählt.

4.2 Allgemeiner Aufbau und Operationsablauf

Im Folgenden wird ein elementarer Aufbau der Module und die Funktionsweise des Systems dargestellt. Dieses System wurde im Rahmen der Bachelorarbeit **Itinerantur** genannt (lateinisch für Reise) und es wurde komplett in Java geschrieben.

Itinerantur besteht aus vier Hauptmodulen:

Transformator Ein Transformator der ein aus einem GATSPTW Graphen mit zeitabhängigen Kantenkosten ein GATSP Graph erstellt.

Vorbereiter Ein Vorbereitungsmodul der aus einem GATSP Graphen eine GTSP Datei erstellt, die zu dem GTSPLIB-Format [FST] konform ist. Dieser Modul schafft eine geeignete GTSP Datei, was der `GLKH` richtig interpretieren und lösen kann.

Löser Ein Löser der eines GATSP Problem lösen kann. In diesem Fall ist `GLKH` der Löser solche Probleme.

Interpreter Ein Interpreter Modul der aus einer TOUR Datei, eine GATSPTW Lösung lesen kann. Diese TOUR Datei wird von `GLKH` erstellt, wenn eine passende Tour in dem GATSP Graphen gefunden wurde.

Die Funktionsweise des Systems und das Zusammenspiel der oben benannten Modulen wird durch die Abbildung 4.1 klarer gemacht. Hierbei sind die Teile durch Farben markiert: Magenta steht für den Transformatormodul, Blau für den Vorbereitermodul, Grün für den Lösermodul und Gelb für den Interpretermodul.

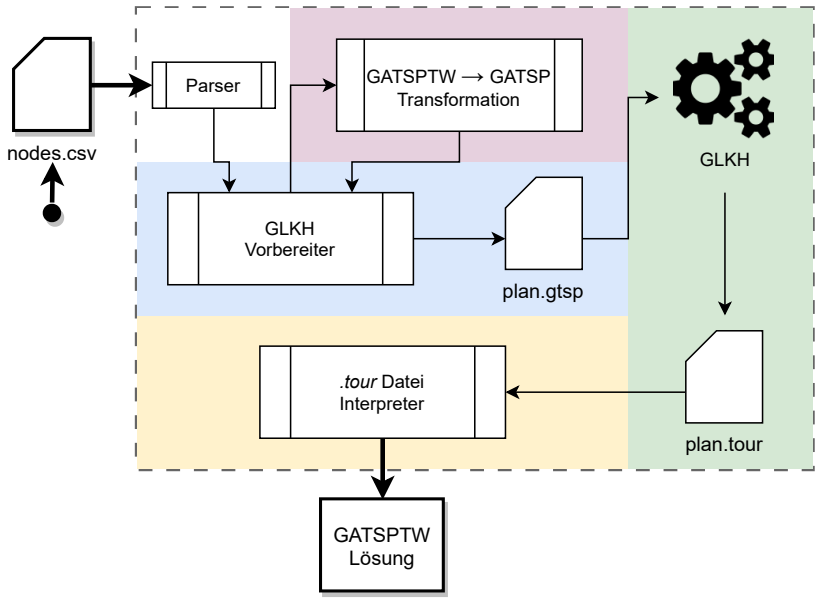


Abbildung 4.1: Funktionsweise von Itinerantur

Der Operationsablauf startet mit einer nodes CSV Datei die eine Beschreibung eines GATSPTW Graphen beinhaltet. Itinerantur liest zuerst diese Datei und parst davon eine GATPSTW Objekt. Dieses Objekt wird dann zu dem `GLKH` Vorbereiter weitergeleitet, wo notwendige Änderungen an dem GATSPTW Modell stattfinden müssen. Diese Änderungen sind für das korrekte Verhalten von `GLKH` im Umgang mit diesem Problem wichtig. Danach wird dieser manipulierter GATSPTW Graph an dem Transformator gegeben, wo ein GATSP Graph dadurch erstellt wird. Nachfolgend wird dieses GATSP Graph wieder leicht manipuliert damit `GLKH` ein korrektes

Ergebnis für unsere Zwecken liefert. Somit wird eine GTSP Datei erstellt und an `GLKH` geschickt und gelöst. Das Lösen davon erstellt eine TOUR Datei, die dann später von dem Interpreter gelesen wird und schließlich eine GATSPTW Lösung unseres ursprüngliches Problem weitergibt.

4.3 Klassenmodellierung

In diesem Abschnitt werden die UML Klassenmodelle von wichtigen Systembausteinen gezeigt. Dabei werden zu jeder Klasse die Relationen zu den Interfaces und anderen Klassen dargestellt sowie deren öffentlichen Methoden. Dieses soll im Weiteren zu dem tieferen Verstehen des Systems und die inneren Abhängigkeiten davon dienen.

4.3.1 Knoten

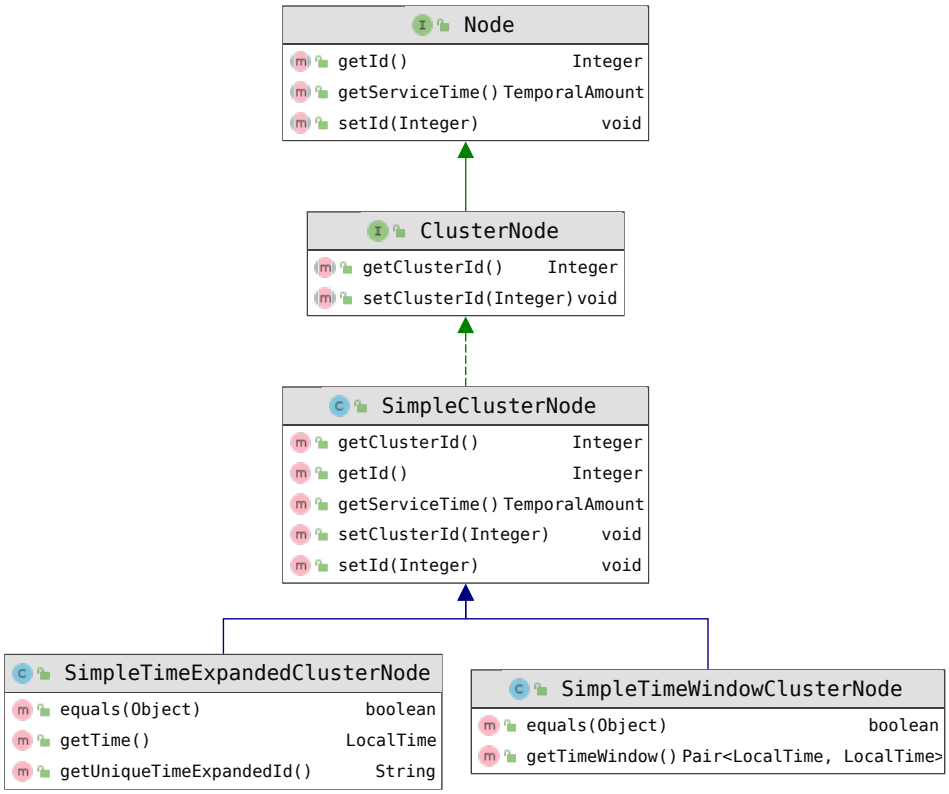


Abbildung 4.2: Klassenmodellierung der Knoten

Itinerantur besitzt die Möglichkeit mit zwei Arten von Knoten zu arbeiten. Diese Knoten sind die SimpleTimeExpandedClusterNode, die zeitgedehnte Knoten modellieren kann und die SimpleTimeWindowClusterNode, die Konten mit einem Zeitfenster modellieren kann.

4.3.2 Kanten

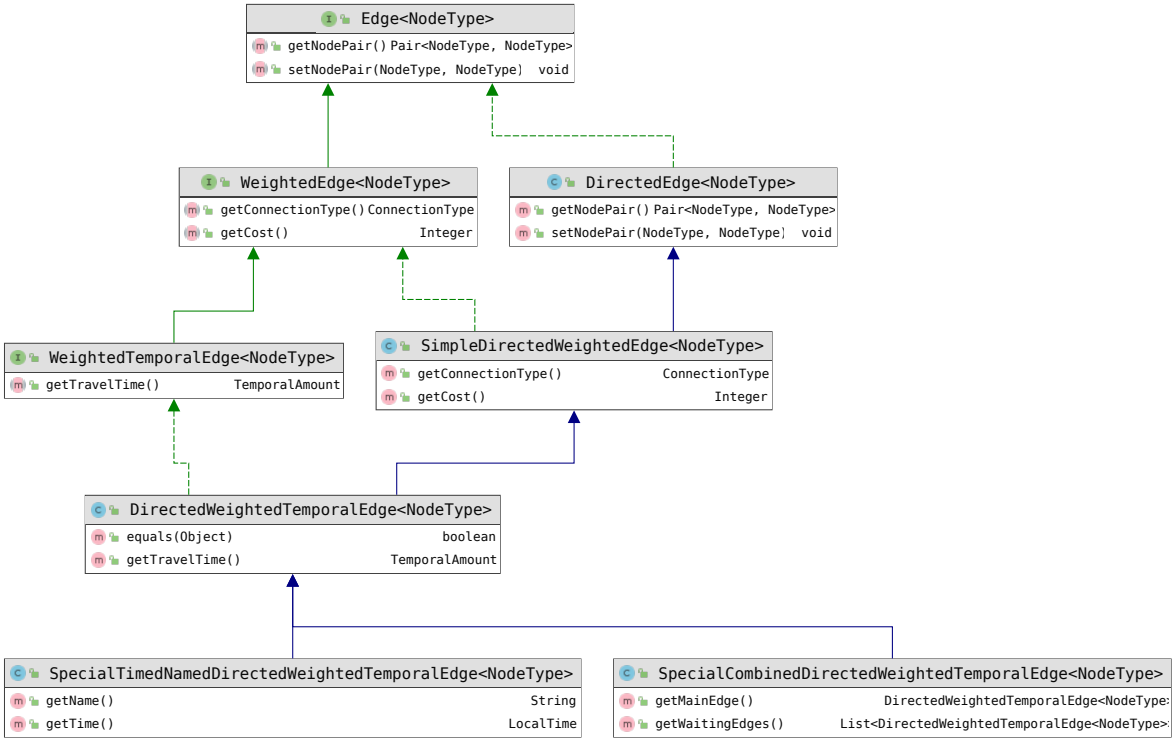


Abbildung 4.3: Klassenmodellierung der Kanten

Itinerantur ist so aufgebaut, dass Kanten kein fester Knotentyp besitzen. Die Edge Oberinterface, wo alle mögliche Kanten ihren Haupttyp bekommen, ist auf einem NodeType Typ verallgemeinert. Damit ist es möglich, dass jede Kantenklasse Verbindungen zwischen die zwei mögliche Knotentypen schaffen kann.

Itinerantur besitzt fünf Kantentypen.

1. Die DirectedEdge ist der einfachste Kantentyp. Damit kann man nur eine simple Kante zwischen zwei Knoten modellieren. Die Knoten werden als ein Paar gespeichert, wo der erste Knoten im Paar der Ursprungsknoten und der zweite der Zielknoten enthält.

2. Die SimpleDirectedWeightedEdge ist eine Erweiterung der DirectedEdge, wobei hier die Kosten und das benutzte Verkehrsmittel gesetzt werden können. Dieser Kante entspricht einem jederzeitigen Anschluss zwischen zwei Knoten, wie der Fall sein kann, wenn ein Fahrrad benutzt wird oder wenn zu Fuß hingegangen wird.

3. Die SimpleDirectedWeightedTemporalEdge ist eine erweiterte SimpleDirectedWeightedEdge die eine Fahrtdauer besitzt.

4. Die SpecialTimedNamedDirectedWeightedTemporalEdge ist ein Kantenmodell, was unter der Fahrtdauer und Kosten die es besitzen kann, auch einen Namen und eine Losfahrzeit haben kann. Mit diesem Kantentyp ist es möglich Verkehrsmodale wie Züge, Busse, usw. zu modellieren, die nur in festen Zeitpunkten des Tages zur Verfügung stehen.

5. Die SpecialCombinedDirectedWeightedTemporalEdge ist ein Sondertyp vom Kanten, die nur zum internen Systemzwecken benutzt wird. Diese Kante ist eine Kombination von Wartekanten und eine verbindende Kante. Mit diesem Kantentyp können Verhältnisse wie "*Warte 5 Minuten und dann nehme die S11 Bahn*" modelliert werden.

4.3.3 Graphen

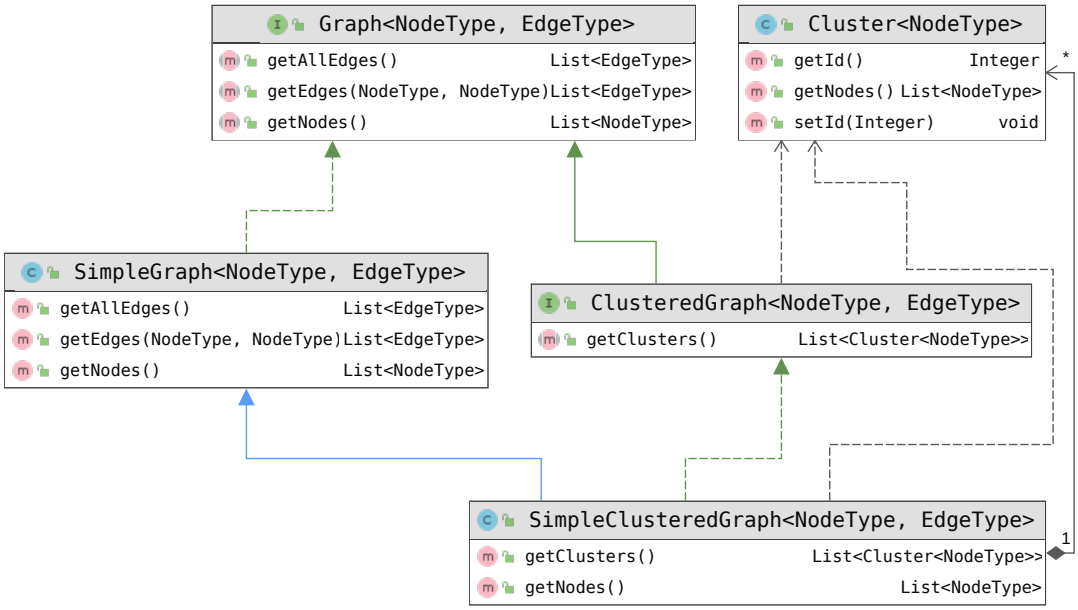


Abbildung 4.4: Klassenmodellierung der Graphen

Wie die verschiedene Kantentypen die aus dem verwendeten Knotentype möglich sind, kann Itinerantur mittels der SimpleClusteredGraph Klasse eine Reihe von verschiedene Graphen je nach den verwendete Knoten- oder Kantentyp aufbauen.

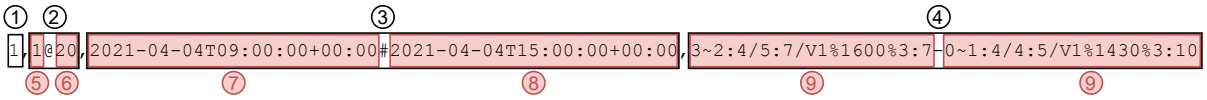
4.4 Detailliter Konstruktion und Funktionsweise des Systems

4.4.1 `nodes` CSV Datei

Als der Eingangspunkt des Systems ist die `nodes` CSV Datei eine Repräsentation eines GATSPTW Graphen mit zeitabhängigen Kantenkosten. In dieser Datei stellt jede Zeile Knotenbezogene Informationen, wie z. B. die ID des Knotens, die Clusterzugehörigkeit davon und mögliche ausgehende Kanten zu den anderen Knoten. Zum besseren Verständnis dieser CSV Datei wird ein Beispieldatensatz genutzt. Sei dieser Beispieldatensatz:

```
1,1@20,2021-04-04T09:00:00+00:00#2021-04-04T15:00:00+00:00,3~2:4/5:7/V1%1600%3:7-0~1:4/4:5/V1%1430%3:10
```

Folgende Schema zeigt die Trennung der Blöcke und die semantische Bedeutung jedes Blockes.

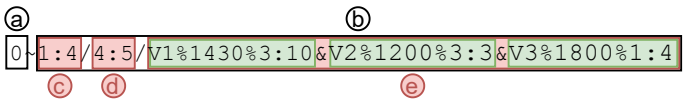


① Hier wird die ID des Knotens gesetzt. Der Depotknoten trägt immer eine ID von 0.

② Dieser Block stellt die Zugehörigkeit des Knotens in einem Cluster. Er teilt sich in 2 weitere Unterblöcke mittels des `@` Trenners. ⑤ enthält die Cluster ID und ⑥ die Servicezeit des Clusters in Minuten.

③ Hier wird das Zeitfenster des Knotens definiert. Die Eintritt- (⑦) und Austrittszeit (⑧) werden mittels der `#` Trenner getrennt. Die Zeiten sind in ISO8601 Format.

③ Hier werden die Kanten von diesem Knoten ausgehend gelistet. Eine Kante (⑨) trennt sich von der anderen mittels `-`. Da ein Kantenmodell etwas kompliziert bzgl. seinem Aufbau ist, wird der zweite ⑨ Kantenblock unten detaillierter erklärt.



ⓐ Hier wird die ID des Zielknotens gesetzt. Danach kommt eine Tilde, die den Anfang des Verbindungenblockes signalisiert.

ⓑ In diesem Block werden die mögliche Verbindungen zu diesem Knoten gelistet. Dabei gibt es drei Hauptunterblöcke getrennt mittels `/`. Die erste zwei Blöcke enthalten Verbindungen, die jederzeit möglich sind und in dem dritten Block stehen Verbindungen, die nur in bestimmten Zeitpunkten zur Verfügung sind. Wenn ein Unterblock keine mögliche Verbindungen haben kann, wird an dieser Stelle nichts geschrieben. Somit bleibt der Anzahl von dem `/` Trenner immer Zwei. (Zur Erklärungszwecken wurde der ⓔ Block um ein paar mehr zeitlich abhängige Verbindungen als in dem Beispieldatensatz erweitert.)

Genauer steht in dem Unterblock

ⓒ die Kosten und die Dauer in Minuten gegeben im Format `KOSTEN:DAUER`, wenn man zu Fuß nach 0 laufen würde.

ⓓ die Kosten und die Dauer in Minuten im gleichen Format wie bei ⓒ, wenn man mit dem Fahrrad nach 0 fahren würde.

ⓔ eine Liste von Zeitabhängige Verbindungen getrennt mittels `&`. Jeder Verbindungsblock (repräsentiert in Grün) ist im Format `ID%HHMM%KOSTEN:DAUER`, wo `ID` die ID dieser Verbindung ist, `HHMM` die Uhrzeit wann diese Verbindung losfährt und `KOSTEN:DAUER`, wie in ⓒ und ⓓ, die Kosten und die Dauer in Minuten dieser Verbindung sind.

4.4.2 Der Vorbereitermodul

Der Parser von Itinerantur verarbeitet zuerst eine `nodes` CSV Datei und erstellt im Anschluss eine SimpleClusteredGraph mit SimpleTimeWindowClusterNode Knoten und SimpleDirected-WeightedTemporalEdge Kanten. Dieses Graph wird, wie auf der Abbildung 4.1 gezeigt wurde, eine Vorbereitungsphase für `GLKH` bevor und nach der GATSP Transformation durchlaufen, damit `GLKH` korrekt funktioniert und die Lösung optimal ist.

Bevor der Vorbereitermodul im Detail geklärt wird, wird zuerst das GTSP Dateiformat und die Arbeitsweise von `GLKH` dargestellt. Damit soll der Sinn über die Notwendigkeit solcher der Graphmanipulationen später nachvollziehbarer gemacht werden.

4.4.2.1 GTSP Format und allgemeiner `GLKH` Arbeitsweise

GTSP Format Die von Fischetti et al. entwicklete GTSP [FST] Format, ist ein Versuch um die Familie der GATSP Probleminstanzen in einem Dateiformat zu standardisieren. Dieser Standard ist jedoch etwas streng, wenn es um die Nummerierung der Knoten- oder Cluster ID-s angeht.

Hierbei kann es keiner ID Sprung bei der Nummerierung der Knoten geben und die Nummerierung soll von 1 anfangen. Das bedeutet, dass wenn die Knoten ID-s zwischen 1 und 100 besitzen, muss es für jede Zahl zwischen 1 und 100 ein korrespondierte Knoten geben. Das liegt daran, dass die Kostenermittlung einer Kante bzw. einer Zelle in der Kostenmatrix anhand der Spalten- und Zeilenposition in der Matrix berechnet wird. Dies bedeutet, dass die Anzahl von Spalten oder Zeilen in der Matrix genau die Anzahl von Knoten entspricht.

GLKH `GLKH` ist zwar eine GATSP Löser, aber es ist insbesondere eine E-GATSP Problemlöser. Das heißt, dass `GLKH` wird nie ein Cluster zweimal besuchen, außer dem Startknoten und folglich dem Cluster in dem dieser Startknoten gehört.

4.4.2.2 ID Verschiebung

Da die Knoten und Cluster ID-s in den `nodes` GTSP Datei von 1 Starten müssen, wird bevor der Graph expandiert wurde eine ID Verschiebung stattfinden. Dabei werden alle Cluster ID-s um zwei verschoben. Selbstverständlich denkt man es sollte nur um eins verschoben werden, da die Clusternummerierung in der `nodes` Datei von 0 anfängt, aber diese ID Verschiebung steht im Zusammenspiel mit 4.4.2.3, wobei da eine extra Dummy Startcluster mit der ID 1 nach dem GATSP Transformation hinzugefügt werden muss, um die E-GATSP Problematik von `GLKH` umzugehen.

4.4.2.3 Depotclustertrennung und hinzufügen eines Dummyclusters

Da `GLKH` ein Cluster nur einmal besucht und in dem **identischen** Knoten, wo es losgegangen ist, endet, stellt so ein Verhalten ein Problem für ein nicht modifiziertes zeitgedehntes Graph dar. Wegen der Zeitdehnung des Depotknotens werden mehrere zeitgedehnte Knoten erstellt und sie werden alle in den Depotcluster wiederum gepackt. Aus der Sicht von `GLKH` haben alle diese Knoten unterschiedliche ID-s und somit wird es nie wieder in dem Depotcluster auftreten, weil es dort vom Anfang an begonnen hat.

Dieses Problem lässt sich auf eine einfache Weise lösen, in dem der Depotcluster kopiert wird, bzw. in zwei Stück getrennt wird und ein Dummy Startknoten und Startcluster hinzugefügt wird.

Bevor das Graph eine Zeitdehnung durchlauft, wird zuerst eine Kopie von dem Depotcluster mit einer neuen ID gleich der Anzahl aller Clustern plus eins erstellt. Wir werden dieser neue Cluster “Enddepotcluster” nennen. Danach werden die Kanten, die ein Depotknoten als Zielknoten haben, so verarbeitet, dass der Zielknoten mit dem neuen Depotknoten aus dem Enddepotcluster ersetzt wird.

Damit lässt sich diese Problematik umgehen, in dem auch unter eine Zeitdehnung des Depotclusters und dessen Kopie, der Depotcluster im Form des Enddepotcluster erneut besucht wird.

Jedoch ist diese Lösung noch nicht funktional. Da wieder wegen der Zeitdehnungsverfahren und die Knoten ID-Interpretation von `GLKH` kann es nie möglich sein, dass eine Tour genau dort endet, wo es angefangen hat. In dem momentanen Zustand des Modells würde dies bedeuten, dass der Reise in dem gleichen Zeitpunkt angefangen und beendet hat.

Um dieses Problem zu lösen, fügen wir in dem Graphen nach der Zeitdehnung ein Dummy Startcluster mit der ID 1. Diese Cluster enthält ein einziger Dummy Startknoten. Danach werden in den Graphen ausgehende Kanten aus diesem Dummy Knoten zu den zeitgedehnten Depotknoten und eingehende Kanten aus dem zeitgedehnten Enddepotknoten zu diesem Dummy Startknoten hinzugefügt.

Mit dieser zwei Manipulationen, die die E-GATSP Problematik ablösen, ist jetzt ein korrekter Graphenstruktur für `GLKH` vorberietet, die unsere transformierte GATSP richtig in Bezug auf dem GATSPTW Problem lösen kann.

Literatur

[AFG01] Norbert Ascheuer, Matteo Fischetti und Martin Grötschel. „Solving the Asymmetric Travelling Salesman Problem with Time Windows by branch-and-cut“. In: *Mathematical Programming* 90 (Jan. 2001), S. 475–506. DOI: 10.1007/PL00011432.

[App+] David Applegate u. a. „Concorde TSP Solver“. In: (). URL: http://www.math.uwaterloo.ca/tsp/concorde.

[App+06] David L Applegate u. a. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[Bai+19] Abdurahman Baizal u. a. „Travel route scheduling based on user's preferences using simulated annealing“. In: *International Journal of Electrical and Computer Engineering (IJECE)* 9 (Apr. 2019), S. 1275. DOI: 10.11591/ijece.v9i2.pp1275-1287.

[Ben+03] D. Ben-Arieh u. a. „Transformations of generalized ATSP into ATSP“. In: *Operations Research Letters* 31.5 (2003), S. 357–365. ISSN: 0167-6377. DOI: https://doi.org/10.1016/S0167-6377(03)00031-2. URL: https://www.sciencedirect.com/science/article/pii/S0167637703000312.

[CLQ16] J. Castillo-Salazar, Dario Landa-Silva und Rong Qu. „Workforce scheduling and routing problems: literature survey and computational study“. In: *Annals of Operations Research* 239 (Apr. 2016).

[Duc19] Vu Duc Minh. „Solving Time-Dependent Traveling Salesman Problem with Time Windows under Generic Time-Dependent Travel Cost“. In: (Nov. 2019). DOI: 10.13140/RG.2.2.32983.21927.

[FGT97] Matteo Fischetti, Juan José Salazar González und Paolo Toth. „A Branch-And-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem“. In: *Operations Research* 45.3 (1997), S. 378–394. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/172016.

[FST] Fischetti, Salazar und Toth. *GTSP File Format*. URL: http://www.cs.rhul.ac.uk/home/zvero/GTSPLIB.

[GK10] Gregory Gutin und Daniel Karapetyan. „A memetic algorithm for the generalized traveling salesman problem“. In: *Natural Computing* 9.1 (2010), S. 47–60. DOI: 10.1007/s11047-009-9111-6.

[Hel00] Keld Helsgaun. „An effective implementation of the Lin–Kernighan traveling salesman heuristic“. In: *European Journal of Operational Research* 126.1 (2000), S. 106–130. ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(99)00284-2. URL: https://www.sciencedirect.com/science/article/pii/S0377221799002842.

[Hel14] Keld Helsgaun. „Solving the Equality Generalized Traveling Salesman Problem Using the Lin-Kernighan-Helsgaun Algorithm“. In: *Mathematical Programming Computation* 7 (Mai 2014). DOI: 10.1007/s12532-015-0080-8.

[JV83] Roy Jonker und Ton Volgenant. „Transforming asymmetric into symmetric traveling salesman problems“. In: *Operations Research Letters* 2.4 (1983), S. 161–163. ISSN: 0167-6377. DOI: https://doi.org/10.1016/0167-6377(83)90048-2. URL: https://www.sciencedirect.com/science/article/pii/0167637783900482.

[KG12] D. Karapetyan und G. Gutin. „Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem“. In: *European Journal of Operational Research* 219.2 (2012), S. 234–251. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2012.01.011. URL: https://www.sciencedirect.com/science/article/pii/S0377221712000288.

[LMW93] Yao-Nan Lien, Eva Ma und Benjamin W.-S. Wah. „Transformation of the generalized traveling-salesman problem into the standard traveling-salesman problem“. In: *Information Sciences* 74.1 (1993), S. 177–189. ISSN: 0020-0255. DOI: https://doi.org/10.1016/0020-0255(93)90133-7. URL: https://www.sciencedirect.com/science/article/pii/0020025593901337.

[Wei03] Eric W Weisstein. „Hamiltonian cycle“. In: *https://mathworld. wolfram. com/* (2003).

[Wil] E.A.B.S.G. Williamson. *Lists, Decisions and Graphs*. S. Gill Williamson. URL: https://books.google.de/books?id=vaXv%5C_yhefG8C.

[Yua+20] Yuan Yuan u. a. „A branch-and-cut algorithm for the generalized traveling salesman problem with time windows“. In: *European Journal of Operational Research* 286.3 (2020), S. 849–866. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2020.04.024. URL: https://www.sciencedirect.com/science/article/pii/S0377221720303581.