Optimaliseren van praktische 'oriëntering'-problemen
met stochastische en tijdstipafhankelijke reistijden: op naar filevrije routes

Optimizing Practical Orienteering Problems
with Stochastic Time-Dependent Travel Times:
towards Congestion Free Routes

Cédric Verbeeck

Promotoren: prof. dr. E.-H. Aghezzaf, prof. dr. ir. P. Vansteenwegen
Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen

Vakgroep Industriële Systemen en Productontwerp
Voorzitter: prof. dr. E.-H. Aghezzaf
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2015 - 2016

ISBN 978-90-8578-871-3
NUR 804
Wettelijk depot: D/2016/10.500/3

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Industriële Systemen en Productontwerp
(ISPO EA18)

Promotoren: Prof. Dr. El-Houssaine Aghezzaf
Prof. Dr. Ir. Pieter Vansteenwegen

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Industriële Systemen en Productontwerp (ISPO EA18)
Technologiepark 903, B-9052 Zwijnaarde, België
Tel.: +32-9-264.55.02
Fax.: +32-9-264.58.47

Dit werk kwam tot stand in het kader van een specialisatiebeurs van het IWT-Vlaanderen (Instituut voor de aanmoediging van Innovatie door Wetenschap en Technologie in Vlaanderen).

Proefschrift tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen
Academiejaar 2015-2016

Alice: Would you tell me, please, which way I ought to go from here?"
Cheshire Cat: "That depends a good deal on where you want to get to."
Alice: "I don't much care where "
Cheshire Cat: "Then it doesn't matter which way you go."

– Lewis Carroll, *Alice in Wonderland* (1865)

Dankwoord

Gedurende mijn master opleiding aan de Universiteit Gent vond ik het wiskundig modelleren van problemen en het implementeren van verschillende optimalisatiemethodes een stimulerende leeractiviteit. Deze interesse in samenspraak met mijn voorliefde voor programmeren gaf uiteindelijk de doorslag om een doctoraatsstudie aan te vatten.

Een doctoraat is niet zomaar het werk van één persoon, daarom wil ik ook mijn oprechte dank betuigen aan enkele mensen die geholpen hebben bij het verwezenlijken van dit onderzoeksproject:

- Mijn promotoren Pieter Vansteenwegen en El-Houssaine Aghezzaf die me deze kans en hun vertrouwen hebben gegeven en steeds bereikbaar waren voor advies, vragen en hulp.
- Daarnaast wens ik de leden van de jury te bedanken: Tom Van Woensel, Nour-Eddin El Faouzi, Sofie Verbrugge, Tarik Aouam en Hendrik Van Landeghem. Hun constructieve opmerkingen en inzichtvolle vragen hebben zowel dit onderzoeksproject als mijn inzicht aangescherpt.
- Ik ben ook het agentschap voor Innovatie door Wetenschap en Technologie (IWT) dankbaar voor de financiële ondersteuning die mij de financiële mogelijkheid heeft gegeven om dit doctoraat te voltooien.
- Verder wil ik mijn collega's Maarten Houbraken, Onne Beek en Derek Verleye in het bijzonder bedanken voor hun spontane hulp en raad doorheen de jaren.
- Ik wens Veerle en Peggy te bedanken om me met plezier bij te staan in administratieve zaken.
- Bedankt, Mario Vanlommel en Be-Mobile voor het aanleveren van de historische file data.
- Ik wil An De Wispelaere en Steven De Schrijver van Solutia bedanken voor het verschaffen van data omtrent realistische bedrijfsituaties waarop files een impact hebben.

- Mijn ouders wil ik ook bedanken want zonder hun steun gedurende dit traject was dit alles uiteraard niet mogelijk geweest.
- Uiteraard wil ik ook mijn vrienden bedanken. Dankzij jullie kon ik dit doctoraatsverhaal beter relativeren. Onne, Arne, Roos, Matthias, Daan, Sander, Mathieu, Maarten en Bram, bedankt om steeds interesse te tonen in mijn bezigheden.
- Anastasia, tenslotte, dankjewel voor je geduld, liefde en glimlach van elke dag.

Gent, december 2015

Cédric Verbeeck

Table of Contents

List of Figures	ix
List of Tables	xi
List of Acronyms	xv
Nederlandse samenvatting	xix
English summary	xxv
1 Introduction	1-1
1.1 Motivation and problem statement	1-2
1.2 The orienteering problem	1-3
1.2.1 Applications	1-4
1.2.2 Classification by travel time	1-5
1.2.3 Other classifiers	1-7
1.3 Objective and outline	1-8
2 Literature survey	2-1
2.1 The orienteering problem (with time windows)	2-2
2.2 The team orienteering problem (with time windows)	2-6
2.3 TD-OP	2-7
2.4 Orienteering problems with stochastic travel times	2-8
2.5 Other variants of the orienteering problem	2-10
2.6 TD-VRP(TW) & STD-VRP(TW)	2-12
2.7 Travel time distributions	2-14
2.8 Conclusion	2-16
3 Problems descriptions	3-1
3.1 Time-dependent orienteering problem (TD-OP)	3-1
3.1.1 A time-dependent travel speed model	3-1
3.1.2 Mathematical formulation	3-3
3.1.3 TD-OP instance generation	3-7
3.2 Time-dependent orienteering problem with time windows (TD- OPTW)	3-11
3.2.1 Mathematical formulation	3-11

3.2.2	TD-OPTW instance generation	3-13
3.3	TD-OPSWTW	3-15
3.3.1	Mathematical formulation	3-15
3.3.2	Position in the literature	3-16
3.3.3	TD-OPSWTW instance generation	3-17
3.4	Conclusion	3-19
4	Solution methods and results for the orienteering problem with time-dependent travel times	4-1
4.1	Motivation for the selected solution method	4-1
4.2	Solution method for the TD-OP	4-2
4.2.1	Insert local search move	4-2
4.2.2	Modified 2-opt	4-5
4.2.3	Ant colony system	4-6
4.3	Results for the TD-OP	4-9
4.3.1	Comparison to optimal solutions of small instances	4-9
4.3.2	Comparison to known optimal solutions of large instances	4-10
4.3.3	Sensitivity analysis	4-14
4.4	Solution method for the TD-OPTW	4-18
4.4.1	General outline	4-18
4.4.2	Pre-processing	4-18
4.4.3	Initialize parameters	4-19
4.4.4	Construct solution & local pheromone update	4-19
4.4.5	Calculation of the local evaluation metric	4-21
4.4.6	Swap local search move	4-21
4.4.7	Insert local search move	4-23
4.4.8	Replace local search move	4-24
4.4.9	Update global_best	4-24
4.4.10	Global Pheromone update	4-25
4.5	Results for the TD-OPTW	4-25
4.5.1	Comparison to optimal solutions of small instances	4-25
4.5.2	Comparison to known optimal solutions of large instances	4-26
4.5.3	Comparison to a state-of-the-art solution procedure	4-28
4.5.4	Sensitivity analysis	4-31
4.5.5	Impact of time-dependent travel times	4-31
4.6	Motivation for the local search moves	4-35
4.7	Conclusion	4-35
5	Solution method and results for the orienteering problem with time-dependent and stochastic travel times	5-1
5.1	Solution method for the TD-OPSWTW	5-1
5.1.1	Arrival time and objective function calculation	5-2
5.1.1.1	Width of departure time distribution spanning multiple time slots	5-3
5.1.1.2	Arriving early at a vertex	5-5

5.1.1.3	Arriving late at a vertex	5-7
5.1.1.4	Monte Carlo simulation	5-10
5.1.1.5	Estimation algorithm	5-11
5.1.2	Pre-processing	5-12
5.1.3	Stochastic ant colony system (SACS)	5-12
5.1.3.1	Construction phase	5-14
5.1.3.2	Using local search moves in a stochastic context	5-15
5.2	Results for the TD-OPSWTW	5-18
5.2.1	Validation of the estimation algorithm	5-18
5.2.2	Evaluation of the SACS algorithm	5-18
5.2.3	Sensitivity analysis	5-20
5.2.4	Impact of stochastic travel times	5-21
5.2.5	Practical insights	5-25
5.3	Conclusion	5-26
6	Case studies: solving a multi-constrained team orienteering problem with time-dependent travel times	6-1
6.1	Mathematical formulation	6-2
6.2	Case studies	6-6
6.2.1	Case study 1: setting and instance generation	6-6
6.2.2	Case study 2: setting and instance generation	6-8
6.3	Pre-processing	6-9
6.4	Solution method	6-9
6.4.1	Construct solution	6-12
6.4.2	Swap	6-12
6.4.3	2-opt	6-13
6.4.4	Calculation of <code>max_shift</code>	6-14
6.4.5	Insert	6-14
6.4.6	Relocate	6-15
6.4.7	Remaining procedures	6-16
6.5	Results	6-17
6.5.1	Results Case study 1	6-17
6.5.2	Results Case study 2	6-19
6.6	Conclusion	6-19
7	Conclusion	7-1
7.1	Concluding summary	7-1
7.2	Contributions	7-5
7.3	Future research	7-6
	References	R-1
A	Appendix	A-1
A.1	Width of departure time distribution spanning multiple time slots	A-1
A.2	Early arrival	A-4

A.3 Late arrival A-5

A.4 Time-independent solution solution case 1 A-6

A.5 Time-dependent solution solution case 1 A-7

A.6 Time-independent solution solution case 2 A-8

A.7 Time-dependent solution solution case 2 A-9

List of Figures

2.1	An example of the OP with feasible solution	2-3
3.1	An example of finding an efficient set of time slots for an arc . . .	3-7
3.2	Arcs randomly assigned to an arc category	3-8
3.3	Fictive example of a time-dependent orienteering problem instance	3-10
4.1	The time-dependent insert move	4-5
4.2	An example where the triangle property is not valid for the time- dependent travel times	4-6
4.3	Example of creating a time-dependent instance with a known op- timal solution	4-12
4.4	3D graph showing the impact of α and β^* on the average gap . . .	4-17
4.5	Overview of the time-dependent local search moves	4-22
5.1	Effect of departure time spanning multiple time slots on arrival time distribution	5-5
5.2	Arrival and departure time distribution in the case of an early arrival	5-6
5.3	Effect of the opening time on μ_d and σ_d in the case of an early arrival	5-7
5.4	Arrival and departure time distribution in the case of a late arrival .	5-9
5.5	Effect of the closing time on μ_d and σ_d in the case of a late arrival	5-10
6.1	An example of the multi-constrained TD-TOPTW	6-6
6.2	Legend to figures of Chapter 6	6-11
6.3	Example of the swap local search move	6-13
6.4	Example of the 2-opt local search move	6-14
6.5	Example of the insert local search move	6-15
6.6	Example of the relocate local search move	6-16

List of Tables

1.1	Overview on orienteering problems with different travel time properties	1-7
3.1	Used speed matrix	3-2
4.1	Comparison to optimal solutions of small instances	4-11
4.2	% gap (best, average, worst) and average CPU time per dataset . .	4-12
4.3	Comparison to known optimal solutions of large instances	4-12
4.4	Effect of the number of trial solutions on the average and maximum gap and average CPU time (s)	4-15
4.5	Overview of the input parameters TD-OP	4-15
4.6	Tested orthogonal combinations	4-16
4.7	Effect of α , β , N_{ni}^{max} , ρ and <code>max_ants</code> on the average % gap . .	4-16
4.8	Effect of the design components on average and maximum gap . .	4-17
4.9	Comparison to optimal solutions of small instances	4-26
4.10	Detailed results TD-OPTW	4-27
4.11	Results for the TD-OPTW per problem instance	4-29
4.12	% gap (best, average, worst) and average CPU time per dataset . .	4-30
4.13	Comparison to a state-of-the-art technique after 10,000 solutions .	4-30
4.14	Overview of the input parameters TD-OPTW	4-31
4.15	Effect of α , β , N_{ni}^{max} , ρ and <code>max_ants</code> on the average % gap . .	4-31
4.16	Effect on the % gap and average CPU time (s) of different design components	4-32
4.17	Impact of time-dependency per problem instance	4-34
4.18	Impact of time-dependency versus problem characteristics	4-34
5.1	Overview on the performance of SACS and ACS	5-20
5.2	The avg % imp for the ACS for different percentiles of the travel time distribution	5-20
5.3	Overview of the input parameters TD-OPSWTW	5-20
5.4	Effect of α , β , N_{ni}^{max} , ρ and <code>max_ants</code> on the average gap	5-21
5.5	Effect on the % gap and average CPU time (s) of different design components	5-21
5.6	Impact of stochasticity per problem instance	5-23
5.7	Impact of stochasticity versus problem characteristics	5-24

5.8	Impact on the average improvement (%) for different coefficients of variation (%) and penalty ratios (%)	5-25
6.1	Overview characteristics case 1	6-7
6.2	Overview characteristics case 2	6-9
6.3	Overview results case 1	6-18
6.4	Overview results case 2	6-19

List of Algorithms

1	Travel time calculation	3-3
2	Time slot finder algorithm	3-6
3	Calculation of <code>max_shift</code> for TD-OP	4-3
4	Ant colony system for TD-OP	4-7
5	Ant colony system for TD-OPTW	4-18
6	Calculation of <code>max_shift</code> for TD-OPTW	4-21
7	Monte Carlo simulation	5-11
8	Estimation algorithm	5-13
9	Ant colony system for TD-OPSWTW	5-14
10	SACS construction phase	5-15
11	Calculation of <code>max_start</code>	5-17
12	Ant colony system for multi-constrained TD-TOPTW	6-11

List of Acronyms

A

ACS	ant colony system
SACS	stochastic ant colony system

F

FIFO	first in first out
------	--------------------

H

HPC	high performance computing
-----	----------------------------

I

ILS	iterated local search
-----	-----------------------

M

MIP	mixed integer problem formulation
-----	-----------------------------------

O

OP	orienteering problem
OPSW	orienteering problem with stochastic weights
OPTW	orienteering problem with time time windows

S

STD-VRPTW	stochastic time-dependent vehicle routing problem with time windows
-----------	---

T

TOP	team orienteering problem
TOPTW	team orienteering problem with time windows
TD-OP	time-dependent orienteering problem
TD-TOP	time-dependent team orienteering problem
TD-OPTW	time-dependent orienteering problem with time windows
TD-TOPTW	time-dependent team orienteering problem with time windows
TD-OPSWTW	time-dependent orienteering problem with stochastic weights and time windows
TSP	travelling salesperson problem
TS	tabu search
TD-VRP	time-dependent vehicle routing problem
TD-VRPTW	time-dependent vehicle routing problem with time windows

U

UAV	unmanned aerial vehicle
-----	-------------------------

V

VNS	variable neighbourhood search
VRP	vehicle routing problem

Nederlandse samenvatting

Inleiding

Rittenplanningsproblemen (vehicle routing problem - VRP) zijn typische problemen binnen het domein van *operationeel onderzoek*, gericht op het bepalen van de efficiëntste routes om klanten te bedienen met een vloot van vrachtwagens, die elk beperkt zijn in laadcapaciteit en/of reistijd. Door toedoen van files moeten logistieke bedrijven die deze rittenplanningen opstellen voor de distributie van producten, leren leven met het feit dat ze wel weten wanneer hun vrachtwagens vertrekken aan een distributiecentrum maar niet weten wanneer die vrachtwagens zullen aankomen bij de klanten. Kortom, de reistijd tussen twee klanten kan sterk variëren doorheen de dag. Hierdoor worden bijvoorbeeld drukke plaatsen en periodes best zo veel mogelijk vermeden.

Een belangrijke variant van het VRP is het *oriëntering probleem* (ORP) (oriëntering problem - OP). Bij dit probleem is het niet langer verplicht om alle klanten te bezoeken maar in plaats daarvan wordt een selectie gemaakt van de meest winstgevende klanten. Een recente trend bij zowel het VRP als het OP is het in acht nemen van tijdstipafhankelijk reistijden (time-dependent travel times - TD). Deze nieuwe problemen worden dan respectievelijk het time-dependent vehicle routing problem en het time-dependent oriëntering problem genoemd, ofwel afgekort tot TD-VRP en TD-OP. Deze toevoeging verhoogt het realiteitsgehalte van beide modellen omdat ze de reistijdvariabiliteit, die alom aanwezig is in het echte leven, in rekening brengt. Echter, zowel het TD-VRP als het TD-OP werden op wetenschappelijk vlak nog relatief weinig onderzocht in vergelijking met het aantal wetenschappelijke publicaties omtrent andere OP en VRP uitbreidingen. Dit gebrek aan onderzoek valt te betreuren aangezien tijdstipafhankelijk reistijden niet langer genegeerd kunnen worden in tal van praktische toepassingen.

Dit onderzoeksproject heeft daarom als voornaamste doel oplossingsmethoden te ontwikkelen voor verscheidene types van TD-OP waardoor toekomstige rittenplanningssoftware efficiënter files kan vermijden.

Probleembeschrijvingen

Het OP wordt gedefinieerd op een *graaf* (graph), een verzameling van *knopen* (vertices) die geografische locaties voorstellen en waar een beloning kan ontvangen worden. Een *boog* (arc) stelt een verbinding voor tussen twee knopen en heeft

een bepaalde reistijd. Het doel van het OP is om te bepalen welke deelverzameling van knopen bezocht moet worden en in welke volgorde zodat de winst maximaal is en de maximum toegelaten reistijd niet overschreden wordt. Daarenboven moet een toegelaten OP oplossing ook starten en eindigen op een voorafbepaalde knoop. Het OP kan worden uitgebreid tot het oriëntering probleem met *tijdsvensters* (*oriëntering problem with time windows* - OPTW) door aan elke knoop een *servicetijd* en een tijdsvenster (*service time and time window*) te koppelen. Enerzijds stelt de servicetijd de tijd voor die nodig is om een knoop te bezoeken en anderzijds staat een tijdsvenster voor de openings- en sluitingstijd van een knoop.

Files worden gemodelleerd door het maken van verscheidene veronderstellingen omtrent de benodigde reistijd om een boog af te leggen. Als men deze reistijd ziet als een constante waarde dan wordt de reistijd als deterministisch aanzien. Indien dit niet het geval is, en de reistijd onderworpen is aan variaties met een zeker kans, definiëren we de reistijd als stochastisch. Anderzijds moet er een onderscheid gemaakt worden tussen tijdsafhankelijke en tijdsafhankelijke reistijden. Als de tijd om een boog af te leggen niet afhankelijk is van het tijdstip waarop men vertrekt dan wordt de reistijd als tijdsafhankelijk gedefinieerd en kan hij voorgesteld worden als één enkele waarde. In het ander geval spreekt men van een tijdsafhankelijke reistijd. In dit laatste geval wordt de reistijd voorgesteld door een functie afhankelijk van het vertrektijdstip. Het oriëntering probleem met deterministische en tijdsafhankelijke reistijden wordt het oriëntering probleem met tijdstipafhankelijke reistijden (*time-dependent oriëntering problem* - TD-OP) genoemd. De uitbreiding met tijdsvensters heet het oriëntering probleem met tijdstipafhankelijke reistijden en tijdsvensters (*time-dependent oriëntering problem with time windows* - TD-OPTW). Beide problemen worden in dit proefstuk onderzocht. De variant van het oriëntering probleem waarbij de reistijd zowel stochastisch als tijdsafhankelijk is, definiëren we als het oriëntering probleem met tijdstipafhankelijke en stochastische gewichten (*time-dependent oriëntering problem with stochastic weights* - TD-OPSW). In dit geval bestaat de reistijd uit een unieke kansverdeling voor elk vertrektijdstip. De uitbreiding van dit probleem met tijdsvensters (*time-dependent oriëntering problem with stochastic weights and time windows* - TD-OPSWTW) wordt in dit proefstuk onderzocht.

Verder worden er ook realistische testproblemen met gekende optimale oplossingen gecreëerd voor het TD-OP. Deze zijn gebaseerd op de originele *tijdstip-onafhankelijke* OP testproblemen in combinatie met een gekend snelheidsmodel voor het TD-VRP. Voor het TD-OPTW en het TD-OPSWTW werd een set van realistische testproblemen gegenereerd op basis van het echte wegennetwerk van de Benelux met 425.479 knopen en 519.915 bogen. De reistijd voor elke boog wordt bepaald aan de hand van historische reistijdinformatie die bestaat uit reistijdmetingen per 15 minuten op een typische dinsdag.

Oplossingsmethode

Metaheuristieken worden in het onderzoeksdomein van het operationeel onderzoek algemeen aanvaard als de meest aangewezen methodes om voor optimalisatieproblemen van een realistische grootte een zeer goede oplossing te vinden binnen een relatief korte tijdspanne. De bijdrage van dit proefstuk bestaat dan ook uit het aanreiken van efficiënte metaheuristieken voor de drie hierboven vermelde problemen (TD-OP, TD-OPTW, TD-OPSWTW). Deze metaheuristieken zijn gebaseerd op een *ant colony system* (ACS) en worden aangevuld met nieuwe *local search moves*. Local search moves proberen bestaande oplossingen te verbeteren door het aanbrengen van kleine wijzigingen aan de huidige oplossing. Daarenboven wordt ook de wiskundige formulering voor de drie problemen beschreven in dit proefstuk.

Het ACS is gebaseerd op het gedrag van een mierenkolonie en is een *constructie metaheuristiek* die verschillende onafhankelijke oplossingen aanmaakt en verbetert gedurende een aantal iteraties. Deze oplossingen worden gecreëerd door een agent (een mier genaamd) die hiervoor eenvoudige beslissingsregels tot zijn beschikking heeft. Het ACS start door één voor één verschillende initiële oplossingen te creëren. Het constructieproces start bij de eerste knoop en voegt dan een nieuwe knoop toe op basis van *greedy* informatie en een geheugenstructuur (*feromonen* genaamd) verbonden aan de boog die naar die knoop in kwestie leidt. De greedy informatie is gebaseerd op de ratio van de beloning en de afstand tussen de twee knopen. Na de creatie van een geldige oplossing worden, omwille van het streven naar diverse oplossingen, de opgenomen bogen minder aantrekkelijk gemaakt voor een volgende constructieprocedure. Dit wordt verwezenlijkt door de feromoonwaarden te verlagen met een *verdampingsfactor*. Zodra alle oplossingen van één iteratie gecreëerd zijn worden ze verbeterd door een set van *local search moves*. Welke local search move wanneer gebruikt worden verschilt per variant van het OP. Het doel van een local search move is om de oplossing te verbeteren door kleine wijzigingen aan de oplossing te maken. Bij het OP is het bijvoorbeeld mogelijk om een extra knoop aan een bepaalde route toe te voegen met het doel een hogere totale beloning te verzamelen. De feromoonwaarden van de bogen die gebruikt werden in de beste oplossing van de iteratie worden verhoogd. Nadien wordt gekeken of de beste oplossing van de iteratie beter is dan de beste oplossing die tot dan toe gevonden werd. Deze procedures worden herhaald gedurende een aantal iteraties. Wanneer na een aantal iteraties geen verbetering wordt gevonden, worden de feromoonwaarden gereset naar hun startwaarden.

Voor het TD-OP bestaat de set van local search moves uit een tijdstipafhankelijke *insert move*, die onbezochte punten probeert toe te voegen aan een oplossing. Deze move kan snel worden uitgevoerd dankzij een lokaal evaluatie criterium. Hierdoor wordt een volledige evaluatie van de oplossing vermeden bij het nagaan van een mogelijke *insert* kandidaat. Om dit te verzekeren slaan we voor elke knoop in de huidige oplossing de maximale tijd op, dat een bezoek aan de knoop kan worden uitgesteld zonder dat de oplossing onhaalbaar wordt.

Voor het TD-OPTW worden de drie volgende local search moves succesvol

geïmplementeerd. De insert move en het lokaal evaluatie criterium werden aangepast om rekening te houden met toegevoegde tijdsvensters. Ten tweede wordt een *replace* move ontwikkeld die ook steunt op het lokaal evaluatie criterium. Deze move probeert een knoop vanuit de huidige oplossing te vervangen door een niet opgenomen knoop met een hogere beloning. Ten derde wordt een *swap* move gebruikt die de positie van twee opgenomen knopen probeert uit te wisselen met als doel de reistijd te verkorten.

De combinatie van tijdstipafhankelijke en stochastische reistijden in samenspraak met de tijdsvensters zorgt voor zeer gecompliceerde berekeningen van de vertrek- en aankomsttijden. Daarom worden de drie belangrijkste berekeningsmoeilijkheden en de manier waarop ze opgelost worden door een *benaderingsalgoritme* beschreven in dit proefstuk. Dit benaderingsalgoritme wordt ook gebruikt om een stochastische *swap* local search move, een insert en een replace move te ontwikkelen.

Resultaten

Het algoritme voor het TD-OP behaalt met korte berekeningstijden goede resultaten op de gegenereerde testproblemen met 20 tot 100 knopen. De optimale oplossing wordt gevonden voor 40% van de testproblemen. Verder wijst een sensitiviteitsanalyse uit dat het algoritme robuust is voor wijzigingen van de inputparameters. Gemiddeld wordt een afwijking van 1,4% met de optimale oplossing behaald, met een rekentijd van 0,5 seconden per testprobleem.

Voor de uitbreiding van het probleem met tijdsvensters behaalt het aangepaste algoritme gemiddeld een afwijking van 0,2% met de optimale oplossing in een gemiddelde rekentijd van 0,4 seconden voor testproblemen die 20 tot 100 knopen bevatten. Een vergelijking tussen het ACS en een eerder voorgestelde oplossingsmethode toont aan dat de prestatie van de eerdere oplossingsmethode 6,2% lager ligt. Wanneer de optimale OPTW oplossing uitgevoerd wordt in een tijdstipafhankelijke omgeving, dan zijn gemiddeld de oplossingen tenminste 11,4% slechter dan oplossingen gegenereerd door het ACS. De gemiddelde TD-OPTW oplossing verschilt qua structuur 35,6% van de optimale OPTW oplossing.

Voor het TD-OPSWTW behaalt het SACS oplossingen, die 23,8% beter zijn dan de optimale OPTW oplossing, uitgevoerd in een tijdstipafhankelijke en stochastische omgeving. Het SACS wordt vergeleken met oplossingen die gegenereerd werden door het ACS (ontwikkeld voor het TD-OPTW) te gebruiken. Als invoerwaarde voor de reistijd van elke boog worden verschillende percentielen van de reistijd distributie uitgetest. Het SACS presteert echter steeds beter ongeacht het gekozen percentiel voor het ACS. Wanneer het *zekerheidsequivalent* opgelost wordt door het ACS voor TD-OPTW dan scoort het SACS gemiddeld 1,3% beter. Beide oplossingen verschillen qua structuur echter gemiddeld 28,3% van elkaar. Het *zekerheidsequivalent* bestaat erin het TD-OPSWTW te reduceren tot een TD-OPTW door telkens de gemiddelde waarde te nemen als waarde voor de reistijd op elke boog. Een verrassend praktisch inzicht dat verworven is, is dat

wanneer er een kans is om te laat te komen bij een knoop, dit zowel kan leiden tot een toename als tot een afname van de *standaardafwijking* van de vertrektijd van die knoop. Ter vergelijking, bij een bezoek aan een knoop dat met 100% zekerheid binnen het tijdsvenster valt, is de standaardafwijking van de vertrektijd gelijk aan die van de aankomsttijd. Bovendien veroorzaakt een kans op wachten aan een knoop altijd een daling van de standaardafwijking. De impact op de standaardafwijking bij een kans op een te late aankomst hangt onder andere af van de ratio van de standaardafwijking van de aankomsttijd en de grootte van de servicetijd. In het algemeen is in het begin van de route de servicetijd nog vrij hoog in vergelijking met de standaardafwijking van de aankomsttijd. Deze laatste bedraagt namelijk nul aan het begin van de route en verhoogt naarmate men het einde van de oplossing bereikt. Daarom zal een kans op een late aankomst bij een knoop in het begin van de route de standaardafwijking van de vertrektijd verhogen, terwijl een mogelijk te laat bezoek aan een knoop op het einde van de route de standaardafwijking van de vertrektijd zal doen dalen.

Praktische toepassing

Naast het oplossen van theoretische problemen zijn de verworven inzichten en oplossingsconcepten ook toegepast op een praktische gevalstudie. Bij deze case worden routeplanningen opgesteld voor twee logistiek bedrijven. Om aan alle eisen uit de praktijk te voldoen, zoals het gebruik van meerdere voertuigen, het inlassen van een lunchpauze en capaciteitslimieten wordt de oplossingsmethode voor het TD-OPTW drastisch aangepast omdat een oplossing nu anders geëvalueerd en aangepast wordt door de local search moves. Het voornaamste besluit uit deze gevalstudie is dat het de moeite loont om tijdstipafhankelijke reistijden in acht te nemen tijdens het optimalisatieproces. Het verder verhogen van de prestatie van de voorgestelde metaheuristiek en het beter formuleren van de doelfunctie voor deze specifieke gevalstudie behoren tot de voornaamste uitdagingen voor de toekomst.

Conclusie

Oriëntering problemen met tijdstipafhankelijke en stochastische reistijden vormen complexe problemen die efficiënte oplossingsmethoden vereisen. Deze methoden moeten namelijk in staat zijn om tijdrovende evaluaties van oplossingen zoveel mogelijk te omzeilen nadat kleine aanpassingen aan een oplossing werden aangebracht. Deze problemen kunnen worden toegepast op rittenplanningsproblemen die te maken hebben met files. Want op een realistisch wegennetwerk zijn de reistijden afhankelijk van het vertrektijdstip. Daarenboven zorgen tal van factoren er ook voor dat men nooit zeker is van het aankomsttijdstip op de bestemming. In dit proefstuk worden oplossingsmethoden voorgesteld die in staat zijn deze problemen op een efficiënte manier op te lossen. Deze oplossingsmethoden worden

uitvoerig getest op realistische test problemen en één van hen wordt ook toegepast op een realistische gevalstudie in de logistieke sector.

English summary

Problem setting

Within the domain of *operations research*, *vehicle routing problems* (VRP) focus on determining an optimal route to visit all customers using a fleet of vehicles, each constrained by a load capacity or a travel time. Managing such a fleet of vehicles is a challenging but a crucial task for companies distributing goods to customers. They need to determine the best order to visit the customers and how to assign the customers to the available vehicles. Moreover, they have to take into account congestion issues: the travel time between two customers will vary significantly during the day. This means that the drivers of the vehicles should try to avoid the busiest areas and roads during rush hour.

The *orienteeing problem* (OP) differs from this classic VRP as not all customers can be visited due to a limited time budget, instead a selection of the most profitable ones needs to be made. The OP and its variants have recently received a lot of attention in the literature. A recent trend is the incorporation of time-dependent travel times. Such problems are more realistic, because they account for travel time variability, which is overwhelmingly present in real life. However, the number of publications about the time-dependent orienteeing problem (TD-OP) but also the time-dependent VRP (TD-VRP) is tiny compared to the number of scientific publications about other extensions of both the OP and the VRP. This lack of research interest is unfortunate, since congestion and thus time-dependent travel times cannot be ignored in a lot of practical applications. The main aim of this dissertation is therefore to develop efficient solution methods to deal with various types of time-dependent orienteeing problems within a short computation time.

Problems descriptions

The orienteeing problem (OP) is defined on a *graph* in which the *vertices* represent customers where a reward can be collected. An *arc* in the graph represents a connection between two vertices and has a certain travel time. The goal of the OP is to determine which subset of vertices to visit and in which order so that the total collected reward is maximized and a given maximum total travel time is not exceeded. In addition to this, a feasible OP solution should start and end at a pre-determined vertex. The OP can be extended with a *time window* and a *service time*

attached to each vertex. The service time represents the time necessary to serve a vertex and the time window stands for the opening and closing time of a vertex.

Congestion is modelled by making various assumptions about the travel time required to traverse an arc. If the travel time is a fixed value it is defined as being deterministic. If this travel time is not a fixed value but the value is subject to variations, the travel time is said to be stochastic. On the other hand, a distinction has to be made between time-independent and time-dependent travel times. If the time to traverse an arc is not dependent on the hour of the day, it is considered to be time-independent and can be represented as a single value. When the travel time is different on different moments of the day, the travel time is said to be time-dependent. The orienteering problem with time-dependent and deterministic travel time information is called the *time-dependent orienteering problem* (TD-OP). Recall that in this case, the travel time on an arc depends on the hour of the day the arc is taken. Therefore, it can no longer be seen as a single value but as a function of values dependent on the departure time. The TD-OP and its extension with time-windows (TD-OPTW) will be discussed in this dissertation. The variant of the orienteering problem where the travel time is both stochastic and time-dependent is called the *time-dependent orienteering problem with stochastic weights* (TD-OPSW). Its extension with time windows, called the *time-dependent orienteering problem with stochastic weights and time windows* or TD-OPSWTW, will also be discussed in this dissertation. A travel time at a certain hour of the day is now represented by a probability distribution instead of by a single value. So the travel time is represented by different probability distributions depending on the departure time.

Moreover, realistic time-dependent test instances for TD-OP with known optimal solutions are developed, based on the original *time-independent* OP instances in combination with a well performing speed model for the TD-VRP. For the TD-OPTW and the TD-OPSWTW a set of realistic problem instances was developed, based on the road network of the Benelux (Belgium, The Netherlands and Luxembourg) which contains 425,479 vertices and 519,915 arcs. The historical travel time dataset, consisting of accurately recorded travel time observations every 15 minutes for a representative Tuesday, is used to calculate the travel time function and travel time distributions for each arc.

Solution approach

Since high-quality solutions are required and the computation time should be limited to only a few seconds for instances of realistic size, the literature on vehicle routing suggests the implementation of a *metaheuristic*. These solution methods try to find better solutions combining efficient *local search* moves with diversification mechanisms. Local search moves try to improve an existing solution by making small changes to the current solution. The main contribution of this dissertation, apart from a mathematical formulation for the three proposed problems (TD-OP, TD-OPTW, TD-OPSWTW), are several fast local search based meta-

heuristics inspired by an *ant colony system* (ACS).

The ACS is based on the behaviour of a foraging ant colony, using ethereal *pheromone trails* to mark travelled arcs. The ACS starts by sequentially creating a number of initial solutions. The construction process starts from the first vertex and in succession adds a new vertex to the solution based on *greedy* information and pheromone trails of the arcs leading to that vertex. The greedy information is based on a ratio which consists of the reward of a vertex and the distance towards a vertex. After the creation of a complete solution, the included arcs are made less attractive for subsequent construction procedures, in order to increase diversification. This is done by depreciating the pheromone values with an *evaporation rate* factor (local pheromone update). Once all solutions of one iteration are created, they are improved, using a different set of local search moves for each problem. The solution with the highest total reward of this iteration is stored. Finally, the arcs that are used in this solution are made more attractive to be used in the solution construction procedure of succeeding iterations, by increasing their corresponding pheromone value. These steps are repeated during a specified number of iterations and the best solution over all iterations is stored. To prevent that only a couple of arcs dominate this solution construction procedure, the pheromone values are reset during a global pheromone update when no improvement can be found during a certain number of iterations.

For the TD-OP, the set of local search moves consists of a modified 2-opt move and a time-dependent *insert* move. The modified 2-opt move first preselects interesting arcs based on the gain in distance. The most interesting arc pair is then evaluated based on real travel times. If the new solution is feasible, it is accepted. The insert move is sped up by a *local evaluation* metric. This local evaluation metric prevents a full and time-consuming evaluation of a solution after every modification attempt and enables an efficient checking and updating mechanism. Therefore, we store for every vertex in the current solution the maximum amount of time that a visit to it can be postponed before the solution becomes infeasible. For the TD-OPTW, three time-dependent local search moves have been implemented. Firstly, the insert move and local evaluation metric were adapted to account for time windows. Secondly, a time-dependent *replace* move tries to replace a vertex from the current solution with a non included vertex with a higher reward. Thirdly, a *swap* local search move tries to exchange two solution member vertices in order to save travel time. For the TD-OPSWTW, the combination of time-dependency, time windows and stochastic travel time severely complicates the calculation of the departure and arrival time distributions. Therefore, three prominent complications were defined in this dissertation and the way they were dealt with by the proposed estimation algorithm is explained and might also be useful for other vehicle routing problems with time windows and time-dependent stochastic travel times. In addition to this stochastic ant colony system (SACS), three stochastic local search moves are used, namely a stochastic swap local search move and a stochastic insert and a replace local search move.

Results

The proposed algorithm for the TD-OP obtains high-quality results on the developed instances requiring very small computation times, even for instances with around 100 vertices. An average run obtains solutions with a reward gap with the optimal solution of only 1.4% using 0.5 seconds of computation time. For 44% of the test instances, the known optimal solution is found. A sensitivity analysis demonstrates that the performance of the algorithm is not sensitive to small changes in the parameter settings.

The TD-OPTW algorithm obtains very good results on the benchmark instances, requiring small computation times due to the combination of a well performing metaheuristic framework and the interaction effect between three efficient local search moves, tailored towards the problem characteristics. The average reward gap with the known optimal solution on these test instances is only 0.2% with an average computation time of 0.4 seconds. A comparison between the ACS and a state-of-the-art technique showed that the performance of this state-of-the-art technique is 6.2% worse. Finally, the optimal OPTW solutions, ignoring the time-dependency of the arcs in the TD-OPTW, perform on average at least 11.4% worse than the proposed ACS solutions. A TD-OPTW solution differs on average 35.6% from an optimal OPTW solution sequence.

Concerning the TD-OPSWTW, promising results were obtained in a reasonable amount of computation time. A first comparison is made by comparing the results of the SACS with optimal solutions for the regular OP with time windows, found using an exact solution method, that are evaluated in a stochastic environment. The solutions proposed by the SACS are on average 23.8% better than executing the optimal solution sequence in a time-dependent stochastic environment. Secondly, the solutions generated by the ACS for TD-OPTW, using different percentiles of the travel time distribution as input, were also evaluated in a stochastic environment and compared to the solutions of the SACS. The SACS performs better than the ACS, independent of the used travel time percentile, as it takes into account the stochastic nature of this problem. However, the SACS also requires a greater computation time. Thirdly, when solving the *certainty equivalence problem* using the ACS method for the TD-OPTW, the SACS improves solutions by 1.3% on average. When solving the *certainty equivalence problem* the mean of the stochastic travel time is taken as travel time estimate. The fourth observation is the solution sequence being on average 28.3% different, compared to the proposed solution by the state-of-the-art algorithm. Subsequently, a surprising practical insight is that a probable late arrival at a vertex can either increase or decrease the *standard deviation* of the departure time. When it is 100% certain that a vertex is visited on time, its standard deviation of the departure time is equal to the standard deviation of its arrival time. The probability of arriving early at a vertex decreases the standard deviation of the departure time. The effect of a probable late arrival depends among other things on the ratio of the standard deviation of the arrival time and the length of the service time. In general, at the beginning of the solution sequence the service time is likely to be higher than the standard deviation

of the arrival time as the service time comprises usually several minutes while the standard deviation of the arrival time is equal to zero at that point. However, the standard deviation of the arrival time tends to increase towards the end of the solution sequence. In short, a probability of a late arrival in the beginning of the solution sequence increases the standard deviation of the arrival time whereas a probable late arrival at the end of the solution sequence might decrease the standard deviation of the arrival time. The combination of these results should motivate vehicle trip planners to consider the time-dependent and stochastic nature of the travel times in the future.

Practical application

Apart from the theoretical problem instances, the acquired insights and solution concepts are also applied on a practical case. In this application, the data from two logistics companies, is used to design vehicle routing planning schedules respectively in Belgium and The Netherlands. Additional practical requirements are modelled such as different time windows per day, the use of multiple vehicles, a lunch break and capacity limitations. The solution method for TD-OPTW was adapted as some of these requirements drastically change the way a solution can be evaluated and altered by the local search moves. From this real-life application, we conclude that it pays off to take time-dependency into account during the optimization process. Increasing the performance of the algorithm and considering a multi-objective between route related costs and time-dependent travel times offers the main avenue for further research.

Conclusion

Orienteering problems with time-dependent and stochastic travel times are complex problems that require efficient solution methods to avoid a time consuming evaluation when a small change is made to a solution. These problems can be applied to vehicle routing applications that suffer from congestion. As in a realistic road network the travel time is dependent on the hour of the day due to traffic congestion. On top of that, many factors cause that one can never be sure when to arrive precisely at the destination. In this dissertation, several solution methods are presented that are able to solve these kind of problems in an efficient way. The solution methods are tested on realistic test problems and one of them is also applied to a real life logistic case study.

1

Introduction

Operations research (OR) is a discipline that deals with the application of advanced analytical methods to help make better decisions. OR employs techniques from other mathematical sciences, such as mathematical modelling, statistical analysis, and mathematical optimization to provide a solution to complex decision-making problems. Therefore, OR overlaps with other disciplines such as industrial engineering and operations management. It is often concerned with determining a maximum (profit, performance) or minimum (cost, risk). These types of problems are called respectively maximisation problems or minimisation problems. OR encompasses a wide range of problem-solving techniques and methods applied in the pursuit of improved decision-making and efficiency, such as simulation, mathematical optimization, queuing theory, Markov decision processes, economic methods, data analysis, statistics, neural networks, expert systems and decision analysis. Operations researchers faced with a new problem must determine which of these techniques are most appropriate given the nature of the system, the goals for improvement and constraints on time and computing power.

Most OR studies involve the construction of a *mathematical model*. Such a model is a collection of logical and mathematical relationships that represents aspects of the situation under study. Models describe important relationships between variables, include an objective function with which alternative solutions are evaluated and constraints that restrict solutions to feasible values. When an OR problem is mathematically modelled, it results in an optimization problem which looks for the best solution to any given *instance* of the problem. An instance of a optimization problem is the concrete particular input to the problem while the

problem itself refers to the abstract question to be solved [69].

After the modelling step, an appropriate *solution method* should be chosen to solve the optimization problem efficiently. Generally, there are two main categories of solution methods to solve combinatorial optimization problems: exact methods and (meta)heuristics. *Exact methods* guarantee an optimal solution. However, in practice, when the problem size expands, it is often not possible to obtain the optimal solution within an acceptable calculation time or memory requirements. Many combinatorial optimization problems are also *NP-hard*, which means that there is no solution method available to find an optimal solution to the problem in polynomial time. An algorithm is said to be polynomial in time if its running time is bounded by a polynomial expression (expression consisting of variables and coefficients, that involves only the operations of addition, subtraction, multiplication, and non-negative integer exponents) in the size of the instance. To solve these problems efficiently, different heuristics and metaheuristics have been used to search for the (sub-)optimal solution.

Heuristics and *metaheuristics* are search methods designed based on human's intuitive and creative thinking, and aim at finding good solutions quickly in a restricted area of the solution space. Metaheuristics are higher level heuristics that control the whole process of search, so that global optimal solutions can be obtained more efficiently. They are iterative procedures that apart from other procedures use local search moves to improve a current best solution. This means that a small change is made to the solution in order to improve the objective function. Although metaheuristics cannot always guarantee to obtain the true global optimal solution, they can provide very good results for many practical problems. Usually, metaheuristics can enhance the computing power of a computer system greatly without increasing the hardware cost. So far, many metaheuristics have been proposed in the literature. One possible classification of metaheuristics uses two categories. The first one is *single-point-single-path* search, and the second one is *multi-points-multi-paths* search. Examples of the former are tabu search, simulated annealing, iterated local search, etc. Common examples for the latter are evolutionary algorithms (including genetic algorithms, genetic programming, evolutionary strategy, memetic algorithm), ant colony systems, etc.

In Section 1.1 the motivation of this dissertation will be discussed. Subsequently, the main problem that will be researched, the orienteering problem, will be introduced in Section 1.2. Finally, the outline of this dissertation will be explained in Section 1.3.

1.1 Motivation and problem statement

Within the domain of operations research, *vehicle routing problems* (VRP) focus on determining an optimal route to visit customers using a fleet of vehicles, each

constrained by load capacity or travel time. Managing a fleet of vehicles is a challenging but a crucial task for companies distributing goods to customers. They need to determine how to assign the customers to the available vehicles and the best order to visit these customers.

In the classic VRP the travel time between two customers is assumed to be a constant value such as a distance metric or an average travel time estimate. However, this is not realistic, since travel times may vary exogenously due to traffic congestion, weather conditions, moving targets or mobile obstacles, or endogenously whenever the planner can set the vehicles speeds (in order to trade-off between fuel consumption and travel time). Subsequently, drivers should try to avoid for example congested roads as much as possible. In general, taking time-dependency into account, which means that the travel time between two customers depends on the departure time at the first customer, leads to a more realistic vehicle trip planning process. In a real-life environment the travel times on an individual arc are stochastic in nature. Due to weather conditions, car accidents and traffic congestion, among others, time and spatial fluctuations of traffic flows can be observed. The planned arrival times are therefore uncertain.

The impact of congestion on the road network differs from country to country. According to INRIX, people living in Belgium (Europe), waste on average 50.9 hours per year [68]. They also rank Belgium as the most congested country in the world. A lot of other European countries such as the United kingdom, the Netherlands, Italy, Switzerland, France, Germany, Luxembourg and Austria are also in the top 10 of most congested countries in the world [68]. According to a somewhat older source, the congestion in Belgium implies a cost of 114 million Euro for the society [127]. The impact of congestion on the vehicle planning process should therefore not be underestimated and therefore forms an important factor when planning vehicle trips.

The problem of optimizing the schedule for a fleet of vehicles given time-dependent travel times is modelled by *time-dependent vehicle routing problems* (TD-VRP). When it is not obligatory to visit all customers but when a customer is visited a specific reward is obtained, the goal of the distribution company changes to selecting a subset of customers in order to maximise the collected reward given a maximum allowed travel time. This problem is called the *orienteering problem* (OP). This dissertation focuses on solving different variants of time-dependent extensions to the orienteering problem.

1.2 The orienteering problem

The *orienteering problem* (OP) is defined on a graph in which the vertices represent geographical locations where a reward can be collected. An arc in the graph represents a connection between two vertices and has a certain travel time. The

goal of the OP is to determine which subset of vertices to visit and in which order, so that the total collected reward is maximized and a given maximum total travel time is not exceeded. In addition to this, a feasible OP solution should start and end at a predetermined vertex. The OP integrates the *knapsack problem* and the *travelling salesperson problem* (TSP). In contrast to the TSP, not all vertices can be visited in an OP due to the limited travel time. However, determining the shortest route for visiting the selected vertices might decrease the total travel time and helps to visit extra vertices. Golden et al. [61] proved that the OP is NP-hard and thus no polynomial algorithm is known to find its optimal solution. The OP and its variants have recently received a lot of attention [44, 145].

In the remaining of this section, the applications of the OP are discussed as well as the classification of different OP variants based on the type of travel time. Finally, in the last part of this section, other classifiers based on other properties or extensions of the OP are mentioned.

1.2.1 Applications

The OP has many interesting applications in military surveillance, tourism and logistics. The military application of the OP considers *unmanned aerial vehicle* (UAV) mission planning to collect intelligence information about different locations in the area of operations [39, 40, 42, 101, 115, 155]. The aim of these missions is to acquire as much information as possible during the flight, while the length of the flight is limited by the available fuel capacity of the UAV.

To model personalized tourist trips, each vertex is a point of interest and the reward indicates the personal appreciation of the point of interest by the tourist. In this application, a tourist wants to maximise the enjoyment by visiting several different sightseeing locations, while the length of the tourist route is restricted by the total time the tourist can spend on sightseeing. These problems are called tourist trip design problems. Apart from our own research [152], this problem was studied by [53, 56, 118, 124, 126, 148–150, 155]. Gavalas et al. [54] present the most recent survey on algorithmic approaches for this class of problems.

When applied in logistic planning tools, which is also the main focus in this dissertation, a vertex represents a customer and the reward reflects the profit margin when the customer is served. The aim is to select the combination of customers and sequence that maximizes the total reward [61, 70, 142].

Other applications of the OP include determining which suppliers to audit to maximize recovered claims [67], the planning of scouting tours to different high schools to scout sport players [13, 20] and pharmaceutical representatives who visit doctors to build brand awareness [158]. Most of these applications are covered more in detail in the survey paper on the OP by Vansteenwegen et al. [145].

1.2.2 Classification by travel time

Routing problems in general and orienteering problems in particular may be classified with respect to a number of criteria. In this dissertation, we have chosen to make a first classification between time-independent and time-dependent travel time information. A second major classification is performed with respect to the quality of the travel time information (certain or uncertain). Other important classifiers are discussed in detail in Section 1.2.3 but might be:

- is a single route (OP) or are multiple routes (team orienteering problem and VRP) required?
- are customers associated to vertices (OP), arcs (arc orienteering problem) or both (mixed orienteering problem)?
- information evolution: is all input known beforehand (static problems) or does the input change over time? (dynamic problems)
- is the schedule completely constructed before its execution (offline scheduling) or is the schedule constructed based on real time events (online scheduling).

In this dissertation only static problems where the customers are associated to the vertices are discussed. The goal is to provide algorithms to the orienteering problem for offline scheduling that take historic congestion patterns into account. However, we developed algorithms that require low computational effort so they can also be used to assist online scheduling and dynamic versions of the orienteering problem.

Given the limited capacity of a road network, traffic congestion has become a daily phenomenon. As people are travelling, digital connected devices can send and receive traffic information on the road. While this allows users to optimize their journey by avoiding congestion they are alerted to, it also improves the quality of the traffic information as the data supplied by the users can be fed back into the system to estimate the traffic state. This source of data is called floating car data and makes it possible together with other data sources such as automatic vehicle identification and inductive loop detectors to obtain very accurate estimations of the travel time on a particular arc on the road network [95, 110, 157, 160]. Congestion causes late arrivals at customers and additional hiring costs for hiring the truck drivers. Therefore, accounting for and avoiding traffic congestion is possible as the required information is easily available and at the same time has a large potential for cost savings [78].

In general, congestion causes two undesirable effects to the transportation users: on the one hand it causes an increase of expected travel time due to a reduction of the mean speed and queuing, and on the other hand it increases the

variability of travel times. Therefore, congestion is modelled by making various assumptions about the travel time required to traverse an arc. If the travel time is a fixed value, it is defined as being deterministic. If this travel time is not a fixed value but the value is subject to variations, the travel time is said to be stochastic. On the other hand, a distinction has to be made between time-independent and time-dependent travel times. If the time to traverse an arc is not dependent on the hour of the day, it is considered to be time-independent and can be represented as a single value. When the travel time is different on different moments of the day, the travel time is said to be time-dependent. Therefore, in a realistic road network, travel times are both time-dependent and stochastic. Due to possible traffic jams, the travel time in a road network is dependent on the hour of the day. Moreover, many factors (weather conditions, congestion, accidents, etc.) cause that one can never be sure when to arrive precisely at the destination.

Congestion itself is typically a local matter, it can be observed and confined within certain parts of the network and affects a limited number of arcs. However, the consequences of congestion can propagate through a much larger part of the network. Local arc travel time variation is experienced by various road network users travelling on different routes, whose reactions causes changes in the traffic patterns elsewhere. For that, even if congestion effects are observable and quantifiable at the arc level, it is important to extrapolate their effects onto the route level. This concept is more thoroughly discussed by Boyce et al. [18]. Since the decisions of road users rely on accurate and reliable travel time estimation and prediction, knowing the variability of travel time might be as important as knowing its expected value.

Research on travel time variability and route choice models for point-to-point navigation already exists and therefore we refer to Corthout et al. [28] for more information. However, the mainstream vehicle routing research incorporates as travel time estimation, the mean route travel time or some measures of travel time reliability (i.e. 10th and 90th percentiles). As a consequence, route travel time variability is rarely considered in vehicle routing applications. The research described in this dissertation aims to show that, given the growing availability and detail of travel time measurements, it is also possible and worthwhile to estimate route travel time distributions and link it directly to the likelihood of collecting a reward or penalty at the customer. In short, including time-dependent and stochastic information in the model can improve the objective value of the generated solutions.

Based on the two already mentioned travel time classifiers, four different variants of the orienteering problem can be devised and they are listed in Table 1.1. In the classic OP, the travel time is considered deterministic and time-independent. This means that in the problem formulation the travel time is represented as a single constant value. Secondly, travel time information can have an amount of uncer-

Table 1.1: Overview on orienteering problems with different travel time properties

	time-independent	time-dependent
deterministic	OP	TD-OP
stochastic	OPSW	TD-OPSW

tainty attached to it but still be independent of the departure time. The orienteering problem with stochastic travel time information is called the *orienteering problem with stochastic weights* (OPSW). The travel time in this case is represented by a single probability distribution. Thirdly, the orienteering problem with time-dependent and deterministic travel time information is called the *time-dependent orienteering problem* (TD-OP). Recall that in this case, the travel time on an arc depends on the hour of the day the arc is taken. Therefore, it can no longer be seen as a single value but as a function of the departure time. The time-dependent problem formulation allows to model congestion related issues in routing problems and multi-modal applications for logistics or tourist trip planners. The TD-OP and its extension with time-windows (TD-OPTW) will be discussed in this dissertation. In the fourth variant of the orienteering problem, the travel time is both stochastic and time-dependent. Instead of by a single value, a travel time at a certain hour of the day is now represented by a probability distribution. So the travel time is represented by different probability distributions, each dependent on the departure time. This leads to the *time-dependent orienteering problem with stochastic weights* (TD-OPSW). Its extension with time windows, called the *time-dependent orienteering problem with stochastic weights and time windows* (TD-OPSWTW), will also be discussed in this dissertation.

1.2.3 Other classifiers

The most popular extension of the orienteering problem is the addition of time windows. In this case, each vertex has a time window (opening time and closing time) and a service time. The addition of time windows to the time-dependent orienteering problem makes the problem more realistic since both tourist attractions and customers also have opening hours in real-life situations. It is therefore an obvious step in the direction of modelling and solving realistic routing problems [2]. On the other hand, by incorporating time windows, waiting times might occur when vehicles arrive at a customer before it opens. This also might postpone the departure time, which directly affects the travel time in a time-dependent problem context. In order to develop efficient solution methods, it is necessary to take time windows explicitly into account. For example, departing earlier at a vertex to avoid congestion, might turn out to be less useful if you can not serve the subsequent vertex at the scheduled arrival time because of the opening time

of the time window. This makes the problem significantly more difficult than the regular time-dependent orienteering problem. When the travel times are assumed to be stochastic, waiting and arriving late at vertices also has an influence on the variability of departure times and therefore also on the variability of the total time of the route. The addition of time windows to the time-dependent orienteering problem results in the time-dependent orienteering problem with time windows (TD-OPTW). The addition of time windows to the time-dependent orienteering problem with stochastic weights is called the time-dependent orienteering problem with stochastic weights and time windows (TD-OPSWTW).

Additional constraints such as split deliveries, the inclusion of a break, additional vehicles (team orienteering problem), limited vehicle capacities (capacitated orienteering problem) will also be added to the basic problem formulation when a practical application is considered in Chapter 6. When multiple additional constraints are added to the orienteering problem, these problems are called *multi-constrained orienteering problems*.

Apart from different kind of travel times and the addition of constraints, the way the reward is collected can also be modelled in multiple ways. In the regular OP a reward is obtained when servicing a vertex but in the *arc orienteering problem* the reward is no longer attached to a vertex but to an arc between two vertices. The orienteering problem where a reward can be collected on the vertex as well as on the arc is called the *mixed orienteering problem* but has to the best of our knowledge not been researched yet.

A last classification is made when considering the either deterministic or stochastic nature of various parameters. Three different types of stochastic orienteering problems exist: orienteering problems with stochastic weights or travel times (discussed in Section 1.2.2), *orienteering problems with stochastic rewards* and orienteering problems with both stochastic rewards and weights, also called *stochastic scouting problems*.

References to these problems will be presented in the literature survey in Chapter 2.

1.3 Objective and outline

The incorporation of time-dependent (stochastic) travel times into vehicle routing problems makes them more realistic, because they account for travel time variability and congestion which are almost always present in real life. Moreover, the travel time data required to solve these problems can relatively easy and accurately be acquired given recent data collection techniques such as floating car data. Although the number of academic publications about the time-dependent (stochastic) VRP in general and the time-dependent (stochastic) OP in particular is increasing, it remains small compared to the number of publications on the classical VRP, OP

and their extensions. This lack of research interest is unfortunate, since congestion and thus time-dependent (stochastic) travel times cannot be ignored in practical applications. Only very recently a survey paper about time-dependent vehicle routing problems is presented by Gendreau et al. [58]. However, using state-of-the-art algorithms originally developed for the time-dependent vehicle routing problem, which do not consider the rewards of the customers, returned mixed results when used as a solution method for the TD-OP.

The overall objective of this dissertation is to develop appropriate metaheuristic solution methods to solve time-dependent (stochastic) variants of the orienteering problem within a short amount of computation time.

In order to achieve this goal the following objectives are considered:

1. Review the literature concerning vehicle routing problems with time-dependent (stochastic) travel times.

Exact algorithms are time-consuming and most research is therefore focused on heuristic approaches.

2. Define and mathematically model the TD-OP, TD-OPTW and TD-OPSWTW.

The problems are formulated as discrete optimization problems and for the non-stochastic problems a mixed integer problem formulation is provided.

3. Create proper benchmark instances that allow for a realistic way of modelling congestion.

Since time-dependent orienteering problems received very little attention in the literature, no benchmark instances exist for time-dependent orienteering problems. Therefore new instances are created and made publicly available.

4. Design and implement efficient pre-processing methods and metaheuristic solution algorithms to solve the time-dependent (stochastic) variants of the orienteering problem.

The literature concerning time-dependent (stochastic) vehicle routing problems is investigated in order to design efficient solution methods. Afterwards, this method is adapted to handle different TD-OP variants but might also be useful for other time-dependent vehicle routing problems.

5. Evaluate the solution methods on the created benchmark instances.

The implemented solution methods are applied to the benchmark instances to analyse and compare the results with optimal solutions and/or results of state-of-the-art solution methods.

6. Apply the developed solution methods to real logistic use cases.

The solution methods are adapted to solve realistic problems, larger in size and requiring additional constraints.

To address these issues, this dissertation is structured as follows.

- In Chapter 2, A literature survey concerning vehicle routing problems with time-dependent travel times is presented.
- In Chapter 3, the TD-OP, TD-OPTW and the TD-OPSWTW are mathematically defined.
- In the first part of Chapter 4, a solution method for the TD-OP is discussed and tested on modified OP instances. However, these instances were based on a simplistic speed-model adopted from the TD-VRP literature. This algorithm is tested on modified OP instances that consist out of a complete graph where all arcs between vertices needed to be assigned to a road category which defined their travel time profile. In the second part of Chapter 4, this TD-OP solution method is extended to incorporate time windows (TD-OPTW) and additional local search moves are added. More importantly, realistic problem instances that originate from a large road network with available travel time profile measurements are solved effectively and efficiently by this metaheuristic solution method.
- In Chapter 5, the same logic is followed for the orienteering problem with time-dependent and stochastic travel times and time windows (TD-OPSWTW). However, the focus in this chapter lies not on the development of a new metaheuristic solution method but on an estimation algorithm that allows a previously developed metaheuristic to handle time-dependent stochastic travel times. The solution method is proposed and numerous experiments to validate the performance and practical relevance of the algorithm are presented.
- In Chapter 6, the results on a practical use case are presented using an adapted version of the proposed solution method from Chapter 4.
- Chapter 7 discusses the conclusions of this research as well as the possible future research in line with this dissertation.

2

Literature survey

The name of the orienteering problem derives from the sport game of orienteering [24, 142]. In this game, individual competitors start at a specified control point and try to maximize their reward by visiting checkpoints and returning to the control point within a given time frame. Each checkpoint has a known reward and the objective is to maximize the total collected reward. The OP is also known as the selective travelling salesperson problem [59, 85, 139], the maximum collection problem [20, 73] and the bank robber problem [11]. Since the OP is NP-hard [61], exact algorithms are time-consuming and most research is therefore focused on heuristic approaches such as the ones found in [24, 60, 61, 111, 126, 142].

Well-known extensions to the OP are the *team orienteering problem* (TOP) and the orienteering problem with time windows (OPTW). The first extension allows to plan trips for multiple days or to make use of multiple vehicles. In the second extension, a time period is defined for each vertex in which it can be visited. The *team orienteering problem with time windows* (TOPTW) combines both extensions. A recent survey on the OP and its extensions was written by Vansteenwegen et al. [145]. In this survey paper a mathematical formulation, benchmark instances and the results of various solution methods on these instances are presented for the OP, OPTW, TOP and TOPTW. Subsequently, other orienteering variants are briefly mentioned.

In the next subsections (2.1 and 2.2) the more recent additions to these four problem classes concerning solution methods and results will be briefly discussed. The orienteering problem variants where the travel time on the arcs is either time-dependent and/or stochastic will be discussed more in detail in two separate sub-

sections (2.3 and 2.4). Thirdly, in a separate subsection other variants on the orienteering problem that are not related to the addition of time-dependent or stochastic travel times are briefly presented (2.5). We decided not to focus on these solution methods that much, since the addition of time-dependency requires significantly different local search moves and diversification procedures as will be explained later on. Fourthly, the modelling of time-dependent and stochastic travel times for the VRP will also be reviewed in a final subsection (2.6). Finally, the modelling of travel time distribution will also be discussed (2.7).

2.1 The orienteering problem (with time windows)

In the regular OP, a set of vertices $V_c = 1, \dots, v$ is given, each with a reward r_i . In the literature this reward is also often called a score or a profit. The first vertex 1 is the start depot and the last vertex v the end depot. The known travel time t_{ij} is required to travel from vertex i to j . The goal is to find a single route with a maximum reward, respecting the time budget t^{max} . Each vertex can be visited at most once.

An example of an orienteering problem with 6 regular vertices and two depots is shown in Figure 2.1. In this example, t_{ij} is calculated using the euclidean distance between the vertices. Regular vertices are represented by circles and the start and end vertex are visualised with a triangle with respectively 1 and v depicted on the inside. The reward of each regular vertex is displayed inside the circle, its service time is assumed to be equal to zero. Subsequently, a solution with a total length equal to 4 and an objective function equal to 18 is visualised by following the arrows from the start vertex to the end vertex. In this dissertation solution concepts will be explained using small examples. In these examples we will make abstraction of the geographical position of the vertices and a solution to the OP and its extensions will be graphically presented as shown at the bottom of Figure 2.1.

The orienteering problem can also be modelled as a *mixed integer problem formulation* (MIP) using the binary decision variable $x_{i,j}$ which equals 1, only if a visit to vertex j happens immediately after visiting vertex i , and 0, otherwise. Secondly, the decision variable u_i equals the order of vertex i in the route.

Figure 2.1: An example of the OP with feasible solution

$$\text{Max} \sum_{i=2}^{v-1} \sum_{j=2}^v r_i x_{i,j} \quad (2.1a)$$

$$\sum_{j=2}^v x_{1j} = \sum_{i=1}^{v-1} x_{iv} = 1 \quad (2.1b)$$

$$\sum_{i=1}^{v-1} x_{ik} = \sum_{j=2}^v x_{kj} \leq 1 \quad \forall k = 2, \dots, v-1 \quad (2.1c)$$

$$\sum_{i=1}^{v-1} \sum_{j=2}^v t_{ij} x_{ij} \leq t^{max} \quad (2.1d)$$

$$u_i - u_j + 1 \leq (v-1) * (1 - x_{ij}) \quad \forall i, j = 2, \dots, v \quad (2.1e)$$

$$2 \leq u_i \leq v \quad \forall i = 2, \dots, v \quad (2.1f)$$

$$x_{ij} \in (0, 1) \quad \forall i, j = 1, \dots, v \quad (2.1g)$$

In this formulation proposed by Vansteenwegen et al. [145], the objective function (2.1a) maximizes the total collected reward. Constraints (2.1b) ensure that the OP route starts in vertex 1 and ends in vertex v . the connectivity of the route is guaranteed by constraints (2.1c). These constraints also make sure that each vertex is visited at most once. Constraints (2.1d) impose the time budget on the route. Equations (2.1e) and (2.1f) are the sub-tour elimination constraints originally formulated for the TSP by Miller et al. [99].

Given the mathematical formulation of the OP, if each vertex i is also assigned a time window $[o_i, c_i]$, the problem is called the orienteering problem with time windows (OPTW). This time window is defined such that a visit (of duration s_i) to vertex i can only start during this time window (it must end at the latest at $c_i + s_i$). The OPTW can also be formulated as a MIP. The opening and closing time of the start and end depot are in most cases set equal to 0 and t^{max} respectively. In this case, the decision variable x_{ij} remains the same but w_i is now used to model the start of the service at vertex i . M represents a large constant.

$$\text{Max} \sum_{i=2}^{v-1} \sum_{j=2}^v r_i x_{ij} \quad (2.2a)$$

$$\sum_{j=2}^v x_{1j} = \sum_{i=1}^{v-1} x_{iv} = 1 \quad (2.2b)$$

$$\sum_{i=1}^{v-1} x_{ik} = \sum_{j=2}^v x_{kj} \leq 1 \quad \forall k = 2, \dots, v-1 \quad (2.2c)$$

$$(w_i + t_{ij} + s_j) - w_j \leq M(1 - x_{ij}) \quad \forall i, j = 1, \dots, v \quad (2.2d)$$

$$w_v \leq \min(c_v, t^{max}) \quad (2.2e)$$

$$o_i \leq w_i \quad \forall i = 1, \dots, v \quad (2.2f)$$

$$w_i \leq c_i \quad \forall i = 1, \dots, v \quad (2.2g)$$

$$x_{ij} \in (0, 1) \quad \forall i, j = 1, \dots, v \quad (2.2h)$$

Equations (2.2a), (2.2b) and (2.2c) are similar to equations (2.1a), (2.1b) and (2.1c). Constraints (2.2d) guarantee the departure time connectivity of the route. Constraint (2.2e) prohibits that the length of the route becomes longer than the time budget and that the time window at the end vertex is respected. Finally, Constraints (2.2f) and (2.2g) ensure that the service to each vertex starts in its corresponding time window.

As mentioned in the survey paper of Vansteenwegen et al. [145], exact and heuristic methods for the OP can be found in [24, 44, 60, 61, 92, 111, 118, 137, 142, 153]. Similarly, methods for the OPTW by Kantor and Rosenwein [70], Montemanni and Gambardella [100] and Righini and Salani [112, 113, 114] were also covered in this survey paper.

More recently, Sevkli and Sevilgen [120] propose a *variable neighbourhood search* (VNS) method for the OP. Blum et al. [17] propose an approximation algorithm for the OP and Kek et al. [75] discuss a polynomial time approximation scheme for the OP. Nagarajan and Ravi [102] propose the first poly-logarithmic approximation for the *directed orienteering problem*. The directed orienteering problem is a variant where the goal is to visit a maximum number of vertices (without taking different rewards into account). Chekuri et al. [25] propose several approximations for the OP an OPTW. Thomas and Manni [140] developed a variable neighbourhood metaheuristic that includes dynamic constraint penalization for the OPTW. The time window constraint is relaxed and added as a penalty to the objective function. By updating the penalty parameter high quality feasible solutions are found by the algorithm. Karbowska-Chilinska and Zabielski [72] propose a hybrid *genetic algorithm* with *path relinking* for the OPTW. Good quality solutions using short computing times together with new best solutions for

some test instances were obtained. Campos et al. [22] propose a heuristic method based on the *greedy randomized adaptive search procedure* (GRASP) and path re-linking methodologies for the OP that achieve high-quality solutions employing short computing times on the benchmark instances. Duque et al. [37] develop an exact method based on the *pulse framework* for the OPTW that outperforms the state-of-the-art algorithm on the benchmark datasets achieving speed-ups of up to 266 times and is able to find optimal solutions to large-scale problems with up to 562 vertices in short computational times.

2.2 The team orienteering problem (with time windows)

The following publications for the TOP are mentioned in the survey paper [145]: [10, 23, 74, 123, 134, 146, 148]. Similarly, solution methods for the TOPTW found in [100, 141, 147] were also discussed by Vansteenwegen et al. [145]. Since the MIP formulations to these two problems will not be used in this dissertation, we refer to the work of Vansteenwegen et al. [145] for a detailed description.

More recently, Kim et al. [77] proposed a large neighbourhood search method for the TOP with three improvement algorithms: a local search improvement, a shift and insertion improvement, and a replacement improvement. They were able to find all but one of the best known solutions on the benchmark instances. Dang et al. [31] propose a *particle swarm optimization* inspired algorithm for the TOP that was also able to find all but one of the best known solutions on the benchmark instances. Ai et al. [3] propose a particle swarm optimization for the TOP together with a new solution representation and decoding method. Lin [93] develop an effective multi-start *simulated annealing* for the TOP and obtain competitive results and improve the result on five benchmark instances.

Labadie et al. [84] propose a hybridized evolutionary local search algorithm for the TOPTW which was able to improve several best known results on the available benchmark instances. Lin and Yu [94] propose a simulated annealing approach for TOPTW and obtain results that are competitive with other solution approaches. Labadie et al. [83] propose a VNS for the TOPTW in which granular instead of complete neighbourhoods are explored. Compared to other methods the method proved to be effective and improved the results on 25 benchmark instances. Cura [29] propose an *artificial bee colony* approach to solve the TOPTW which achieves comparable results. Hu and Lim [65] propose an iterative three-component heuristic for the TOPTW. The first two components are a local search procedure and a simulated annealing procedure by which the solution space is explored and a set of routes is discovered. The third component recombines the routes to identify high quality solutions. Tae and Kim [132] have developed a *branch-and-price* approach

and were able to find optimal solutions for many TOPTW benchmark instances. The proposed algorithm could also find a higher number of optimal solutions on the TOP benchmark instances.

2.3 Orienteering problems with deterministic time-dependent travel times

The time-dependent variant of the OP is relatively new and has, to the best of our knowledge, only been studied in [1, 48, 53, 55, 56, 89, 90]. Fomin and Lingas [48] were the first authors to mention the TD-OP and state that it is NP-hard because the OP is NP-hard. However, they do not develop an algorithm for the TD-OP that can be used in practical situations. Abbaspour and Samadzadegan [1] introduce a solution procedure for the TD-OP based on two adaptive genetic algorithms and multi-modal shortest path finding modules. They are able to solve multi-modal routing problems in the city of Tehran, although no absolute performance measure (gap) was reported. Li et al. [90] propose a mixed integer programming model of the TD-OP combined with an optimal pre-node labelling algorithm based on the idea of network planning and dynamic programming. However, this algorithm is not tested on benchmark instances and therefore no performance metrics are proposed. The same conclusion holds for Li [89] where a mixed integer programming model is proposed and an optimal dynamic labelling algorithm is designed for the time-dependent team orienteering problem (TD-TOP). Garcia et al. [53] develop an *iterated local search* (ILS) heuristic for the time-dependent team orienteering problem with time windows (TD-TOPTW) which allows them to illustrate, based on a case study in the city of San Sebastian, that obtaining near-optimal routes in a few seconds is feasible. However, only a special case of time dependency is considered, which is the result of using public transport in a city environment. For example, when a traveller arrives at a bus stop before the bus arrives he needs to wait. Therefore the travel time between two vertices consists of both the waiting time and the driving time and depends on the departing time at the start vertex. They exploit the fixed frequency of bus services to come up with an efficient solution technique.

The most recent work is from Gavalas et al. [55] and Gavalas et al. [56]. Actually, this work was published after we published our research on the TD-OP [151], which we will discuss in Chapter 4. The former propose several cluster-based heuristics for the TD-TOP applied to tourist trip planning. Unlike Garcia et al. [53], they do not make the assumption of periodic service schedules. The algorithms were tested on datasets compiled from the metropolitan area of Athens.

2.4 Orienteering problems with stochastic travel times

Solution methods for the OPSW or relevant problems similar to the OPSW were discussed in [21, 41, 42, 106, 107, 134, 138]. Note that in most of these works the term weight is used for the travel time and capacity for the time budget or t^{max} . Hence the name orienteering problem with stochastic weights. We will handle real capacity constraints like volume and weight, respectively expressed in cubic metres and kilograms, in Chapter 6. Therefore, in this dissertation we refrain from using these two notations (capacity and weights) when describing time related concepts. However, we will still use the name orienteering problem with stochastic weights (OPSW) because it is the accepted term in the literature.

Discussing the differences between the works presented in this section and our TD-OPSWTW is more useful after the mathematical description of the TD-OPSWTW has been presented in Section 3.3.1. Therefore, these differences will not be discussed here, but in Section 3.3.2.

A related problem to the OPSW is the *time constrained travelling salesman problem* with stochastic travel and service times as introduced by Teng et al. [138]. In this problem, both the travel and service time are independent discrete random variables. The objective is to maximize the total reward collected on a route while a limit is set to the total travel and service time of a route. A penalty is taken proportional to the amount in excess of this time limit. This problem is formulated as a two-stage stochastic problem with recourse, where in the first stage a subset of vertices is sequenced before the travel and service times are realized. Afterwards, in the second stage the expected penalty is imposed on the objective function.

Tang and Miller-Hooks [134] propose an orienteering problem with discrete stochastic service times, travel times and travel costs and call it the *stochastic selective travelling salesperson problem*. They formulate the problem as a chance-constrained stochastic integer program where the objective is to maximize the total reward collected minus the travel cost from an a-priori route while restricting the probability that the route length exceeds a certain threshold. The authors propose both exact and heuristic methods for solving this problem. Their experiments focus on demonstrating algorithmic performance and consider only tight delivery deadlines.

Campbell et al. [21] propose an *orienteering problem with stochastic travel and service times* (OPSTS) where a vertex specific penalty is incurred for each scheduled vertex not reached before a given known general deadline. However, contrary to the regular orienteering problem, the routes do not need to end at an end vertex. They investigate special versions of the problem that can be solved optimally and present VNS based heuristics to solve the general versions of the problem.

Evers et al. [41] compare a robust optimisation to the two-stage orienteering

problem with stochastic weights using a case study. Contrary to the two-stage approach of Teng et al. [138], during the second stage no penalties are imposed but the recourse action of returning to the end depot is made when the remaining time equals the expected time required to return from the current vertex to the end depot. A certain amount of extra time is assumed to be available to cover for possible late arrivals at the end depot.

In Papapanagiotou et al. [107] and its extension Papapanagiotou et al. [106], the objective function of the OPSTS proposed by Campbell et al. [21] is further investigated. The authors compare the effectiveness of an exact objective evaluation versus a Monte Carlo sampling method and a hybrid method which combines aspects of the latter two.

Evers et al. [42] study an orienteering problem with a stochastic travel time on each arc and a hard constraint on the total time of a route. They introduce a two-stage recourse model where a recourse action of aborting the route (returning to the depot) is taken in the second stage. This action is executed when the remaining time equals the expected time required to return from the current vertex to the end depot. They propose a sample average approximation procedure to solve the problem, in which the objective function is approximated using Monte Carlo sampling.

Lau et al. [86] present the time-dependent orienteering problem with stochastic weights and call it the *dynamic orienteering problem with (discrete) stochastic travel times*. The authors propose, apart from the time-dependent stochastic travel times, a risk-sensitive criterion to allow for different risk preferences and develop a branch-and-bound algorithm and a local search algorithm for this problem. They also provide two approaches to calculate the travel time distributions: a sampling approach and a matrix-based approach.

The orienteering problem with stochastic travel times and time windows is discussed in [39, 158]. Firstly, Evers et al. [39] extend the orienteering problem with uncertain travel and service times, time windows and the addition of new vertices during the flight, for the case of online stochastic UAV mission planning. They call this problem the maximum coverage stochastic orienteering problem with time windows, which generates a route with a maximum expected reward of vertices that are known in advance. During the execution of the route a heuristic similar to Vansteenwegen et al. [147] is used to replan the route each time a vertex is visited.

Secondly, Zhang et al. [158] propose a stochastic orienteering problem with time windows where the travel time is deterministic but the waiting time at a vertex is a discrete stochastic variable dependent on the arrival time at a vertex. When serving a vertex within its time window a reward is gained, otherwise a penalty is taken. The authors propose an analytical formula to compute the expected reward from an a-priori route that takes into account the recourse actions of not travelling

to a scheduled vertex and deciding not to wait at a vertex. Furthermore, they propose a VNS heuristic similar to Campbell et al. [21] to solve this problem.

2.5 Other variants of the orienteering problem

Wang et al. [153, 155] and Geem et al. [57] propose solution methods for the *generalized orienteering problem*. In the OP, the total reward is equal to the sum of the individual rewards, while in the generalized orienteering problem the total reward is a more complicated (nonlinear) function of the vertices visited. Pietz and Royset [108] propose a generalized orienteering problem where both the transit on the arcs in the network and the reward collection at vertices consume a variable amount of the same limited resource. They developed a branch-and-bound algorithm that relies on partial route relaxation problems to solve the problem. Erdogan and Laporte [38] discuss another generalization of the orienteering problem, called the *orienteering problem with variable profits*. Every vertex has a reward to be collected and an associated collection parameter. The vehicle may make a number of passes, collecting a certain percentage of the remaining reward at each pass. In an alternative model, the vehicle may spend a continuous amount of time at every vertex, collecting a percentage of the reward given by a function of the time spent.

The *multi-objective orienteering problem* was introduced by Schilde et al. [118]. In this problem formulation each vertex provides different rewards for different categories. Each tourist has different preferences for the different categories when selecting and visiting the vertices. Hence, a multi-objective decision situation arises. The goal is to determine all the pareto optimal solutions. Gendreau et al. [59] presented an OP with compulsory vertices and propose a branch-and-cut algorithm to solve it.

The *capacitated team orienteering problem* was first introduced by Archetti et al. [9], together with an exact and several heuristic solution methods. In addition to the regular TOP, in the capacitated TOP a demand is associated to each vertex and the total demand of the route may not exceed the given capacity. Taranitis et al. [136] developed a bi-level filter-and-fan method for the capacitated TOP. The method takes advantage of different search landscapes. At the upper level, the solution space is explored on the basis of the collected profit, using a filter-and-fan method and a combination of profit oriented neighbourhoods, while at the lower level the routing of vertices is optimized in terms of travelling distance via a variable neighbourhood descent method. Luo et al. [96] propose a heuristic based on the ejection pool framework with an adaptive strategy and a diversification mechanism based on toggling between two priority rules.

Archetti et al. [7] and Archetti et al. [8] present the capacitated TOP where split deliveries are allowed. This problem allows that each vertex may be served

by more than one vehicle. On the tested instances the reward increases between 0% and 50%. Furthermore, they present a branch-and-price exact algorithm and a hybrid heuristic and test these solution methods on new and existing benchmark instances. Wang et al. [154] introduce the capacitated team orienteering problem with split delivery and minimum delivery amounts. A worst-case analysis is performed to determine tight bounds on the maximum possible reward increase.

Angelelli et al. [5] introduce the *clustered orienteering problem* together with two solution methods: a branch-and-cut approach and a *tabu search* (TS). In this problem a reward is associated with each cluster and is gained only if all vertices belonging to the cluster are served. Salazar-Aguilar et al. [117] introduce the multi-district team orienteering problem. In this problem one must schedule a set of mandatory and optional vertices located in several districts, within a planning horizon. The total available time determined by the length of the planning horizon must be distributed among the districts. All mandatory vertices within each district must be performed, while the other vertices can be performed if time allows. A positive reward is collected whenever an optional vertex is performed.

In Divsalar et al. [34] and Divsalar et al. [35] respectively a VNS and a *memetic algorithm* are proposed for the *orienteering problem with hotel selection*. The goal for this variant is to determine a fixed number of connected trips that visits some vertices and maximizes the sum of the collected rewards. Each trip is limited in length and should start and end at one of the hotels from a predefined set.

Souffriau et al. [125] studied the *multi-constrained team orienteering problem with multiple time windows*, which allows defining different and/or multiple time windows for different days. The proposed algorithm is based on a hybrid iterated local search (ILS) and *greedy randomized adaptive search procedure* (GRASP) approach. The GRASP yields an initial solution and the *shake* routine of ILS is used thereafter to derive an improved solution. The ILS-GRASP algorithm yields fairly quality solutions, while achieving computation times suitable for online applications. Vansteenwegen et al. [149] extend the TOPTW to model the optimisation problem of a personalised electronic tourist guide. They introduce multiple time windows per vertex and time windows that can be different on different days. Moreover, the possibility to schedule a lunch break is added and a GRASP is implemented to solve the problem.

The orienteering problem with stochastic profits was introduced by Ilhan et al. [67] and they developed an exact and genetic solution method for the problem and described an application in the car industry. The stochastic scouting problem (orienteering problem with both stochastic profits and weights) was presented by Barros and Evers [13] and they describe an application of efficiently scouting talented sports players where the times required to travel between schools and the time required to scout at these schools is uncertain.

2.6 Vehicle routing problems with time-dependent and/or stochastic travel times

Existing solution methods for the related *time-dependent vehicle routing problem* (TD-VRP) described in [62, 66, 78, 119, 143, 159] and the *time-dependent vehicle routing problem with time windows* (TD-VRPTW) which can be found in [12, 26, 30, 36, 45, 63, 80, 98, 103, 105, 109, 121, 156] provide inspiration on how to efficiently deal with time-dependency and time windows. Most of these works are also mentioned by the survey paper about time-dependent routing problems [58].

An issue that has received some attention in the early time-dependent routing literature is the modelling of time-dependent travel times and the generation of realistic benchmark instances. One of the first approaches defines the travel time between vertices as a function of the distance and the time of day by using a time-dependent cost factor. This results in a piecewise constant distribution of the travel time used in the solution methods proposed by Malandraki and Daskin [98] and Potvin et al. [109]. Although this technique is easily applied, the generated travel times violate the *first in first out* (FIFO) principle, as shown by Ichoua et al. [66]. The FIFO principle states that if two vehicles leave from the same vertex to go to the same vertex and travel on the same route, the one that leaves first also has to arrive first. Based on this piecewise distribution of the travel time, however, it is possible to leave later and arrive earlier than another vehicle. Obviously, this is not realistic.

The proposed speed models of Donati et al. [36], Ichoua et al. [66] are able to model more realistic congestion behaviour between vertices and satisfy the FIFO principle. By adopting such a speed model, more realistic benchmark instances can be created by working with step-like speed distributions and adjusting the travel speed whenever the boundary between two consecutive time periods is crossed. An important decision concerns the way the travel time data is stored. Donati et al. [36] divide the planning period (e.g., one day) in fixed-width time periods (e.g., one hour). Ichoua et al. [66] adjust the time period width to the expected travel time profile, still using the same time period boundaries for all arcs. The latter method is more realistic as the travel time fluctuations due to rush hours are better modelled. Still, these rush-hours do not necessarily occur at the same time of the day on all the arcs. On the other hand, dividing the travel time into time periods that are different for each arc, turns out to be quite difficult as these matrices have to be created artificially. This approach has been used by Chen et al. [26] and Malandraki and Daskin [98] who arbitrarily create variable time zones for all arcs but without mentioning a procedure to control for spatial consistency. In order to realistically imitate real-life traffic congestion, it is not enough to be time consistent (FIFO-conforming). According to Lecluyse et al. [87] the speed model should also be spatial consistent as congestion tends to grow and shrink in

spatially correlated zones and not independently on individual roads. Both time and spatial consistency, will be taken into account when benchmark instances for the TD-OP are created in this dissertation.

As far as solution methods are concerned, tabu search (TS) is the most commonly applied metaheuristic for the TD-VRP(TW) [66, 103, 105, 119, 143, 156, 159]. Other algorithms that have been developed for the TD-VRP(TW) include a genetic algorithm [62], a heuristic combining route construction and route improvement [26, 45, 109], a VNS approach [80], an ILS [63] and three *ant colony systems* (ACS) [12, 36, 159]. Osvald and Stirn [105] present a distribution problem of fresh vegetables in which the perishability represents a critical factor. They apply TS to this problem and test their algorithm on data of the Slovenian food market. They achieved improvements of up to 47% reduction in perished goods. Kok et al. [79] presented a heuristic based on dynamic programming for TD-VRP while considering driving hour regulations. Kuo [81] studied a variant of TD-VRP aiming to minimize the total fuel consumption. The authors proposed a simulated annealing procedure for finding the routing plan with the least total fuel consumption. Experiments showed that the proposed method provided a 24.6% improvement in fuel consumption over the method based on minimizing transportation time and a 22.7% improvement over the method based on minimizing transportation distances. Maden et al. [97] proposed and applied a TS heuristic for the TD-VRPTW on real data from an electrical wholesale company in the SouthWest of the UK. The proposed approach led to savings in CO₂ emissions of about 7%. Soler et al. [121] transform the TD-VRPTW into an asymmetric capacitated vehicle routing problem in order to solve it with a state-of-the-art exact and heuristic solution method. Kok et al. [78] use a time-dependent shortest path algorithm together with a restricted dynamic programming heuristic to tackle real-life TD-VRP test instances. Dabia et al. [30] developed a branch-and-price algorithm for the TD-VRPTW. They used new dominance criteria which enabled them to solve some small and medium sized instances. Nguyen et al. [103] introduce a TS meta-heuristic for the time-dependent multi-zone multi-trip vehicle routing problem with time windows. A diversification strategy, guided by an elite solution set and a frequency-based memory, is used to search for unexplored good regions. Zhang et al. [159] developed a hybrid algorithm that integrates both ACS and TS algorithms for the time-dependent vehicle routing problem with simultaneous pickup and delivery. Wen and Eglese [156] present a new algorithm for TD-VRPTW inspired by TS and shows how costs for freight distribution may be influenced by traffic conditions using a case study in the London area. Setak et al. [119] present the time-dependent vehicle routing problem in a multigraph where one vertex can be accessible from another with more than one arc and proposed a TS solution method for this problem.

Solution methods for the *vehicle routing problem with stochastic travel times*

and time windows are presented in [4, 91, 116, 129, 131]. Ando and Taniguchi [4] propose a genetic algorithm and focus on small-sized problem instances (up to 25 vertices) and Russell and Urban [116] develop a TS method where the performance of the solution approach is evaluated on well-known problem instances with 100 vertices. Li et al. [91] proposed a heuristic algorithm based on a TS method and tested their procedure on problem instances with up to 100 vertices. In the problem proposed by Taş et al. [129] and Taş et al. [131], the vertices have soft time windows. In Taş et al. [129] the authors propose a TS method together with a post-optimization method, while in Taş et al. [131] a column generation procedure and a branch-and-price solution approach are applied.

Research dedicated to the *stochastic time-dependent vehicle routing problem with time windows* or STD-VRPTW can be found in [88, 128, 130]. In Lecluyse et al. [88] the time-dependent travel times are the result of a stochastic process due to traffic congestion. The proposed queueing models are solved for problem instances with up to 80 vertices by an algorithm based on a TS method. In the problem formulation discussed by Taş et al. [130] vertices have soft time windows. The authors propose TS and an adaptive large neighbourhood search to solve this problem. Sun et al. [128] propose a method to convert the STD-VRPTW into a TD-VRP problem and use a formerly presented route construction algorithms to solve the converted problem.

Finally, Franceschetti et al. [50] proposed the *time-dependent pollution-routing problem* in which speed restrictions imposed by traffic congestion are taken into account. They presented an integer linear programming formulation and an analytical characterization of the optimal solutions for a single-arc version of the problem. In this latter they identify conditions under which it is optimal to wait idly at certain vertices in order to avoid congestion and to reduce the cost of emissions. They also illustrated a departure time and speed optimization algorithm for the case where the route is fixed.

2.7 Travel time distributions

The earliest studies modelling the travel time and speed on arcs of the network were performed by Berry and Belmont [15]. The speed on an arc was found to be normally distributed. Kharoufeh and Gautam [76] showed that travel times were roughly normally distributed, although slightly skewed, indicating that a log-normal distribution might be interesting as an alternative. The results of Taniguchi et al. [135] and Kwon et al. [82] show that there is always a certain minimum time needed to cover the distance. After this minimum time, the probability corresponding to travel times higher than this minimum, increases rapidly to a maximum after which this probability slowly decreases with a long tail. Therefore, Taniguchi et al. [135] advise to use log-normal distributions instead of normal distributions. Fur-

thermore, Noland and Polak [104] and Clark and Watling [27] also assume route travel times to be well approximated by normal or log-normal distributions, under the condition that arc travel times are (sufficiently) independent variables on a day-to-day time scale, or that the network is under user equilibrium constraints. Although assuming normality offers numerous analytical and computational advantages, it imposes some unrealistic restrictions such as symmetry and non-zero probability of negative travel times. Subsequently, the conventional (2-parameter) log-normal distribution is bounded below by zero leading to unreasonable free flow speeds.

The distributions of the travel times most commonly applied in the field of vehicle routing problems to solve road logistic problems are uniform, normal, log-normal, shifted gamma and gamma distributions ([21, 43, 71, 88, 91, 106, 116, 129–131, 134]). The gamma distribution is the most commonly used ([21, 43, 88, 106, 116, 129–131]). These distributions can be estimated using different data sources such as floating car data, automatic vehicle identification and inductive loop detectors ([95, 110, 157, 160]). Although the calculation of arc travel time distributions is possible through these data sources, the relationship between these distributions and travel time distributions at the route level is not straightforward ([51, 64]). Because arc flows and travel times are often highly mutually correlated, the calculation of route travel time variability can not always be calculated by summing up the arc travel time variability.

As shown in Chapter 5, the main contribution is to show that it is possible to solve a stochastic time-dependent orienteering problem with (hard) time windows without using for example a 2-stage solution approach or computational expensive Monte Carlo simulations. Instead a metaheuristic is proposed that takes the stochastic aspects of the problem into account during the optimization process. Furthermore, the significant impact of time windows on the variability of the solution is theoretically and empirically shown as time windows are often not included in the problem formulations of the orienteering problem with stochastic travel times. Therefore, in this dissertation, for the problem instances based on real-life data, time slots with a length of 15 minutes will be used and the stochastic travel times are assumed to be normally and independently distributed. Contrary to the commonly used gamma distribution, the normal distribution is chosen as it facilitates the travel time estimation process because of its analytical properties (convolution, truncation, symmetry). In future research the focus could be placed on the adaptation of the solution procedure to handle the more realistic probability distributions such as the gamma distribution.

2.8 Conclusion

The orienteering problem received a lot of attention lately. A lot of different extensions have been introduced and its most popular extension forms the team orienteering problem (with time windows). Time-dependent and/or stochastic extensions to the orienteering problem, however, received little attention in the literature. In this dissertation, problem descriptions and solution methods for two time-dependent extensions (TD-OP, TD-OPTW) and a time-dependent and stochastic extension of the orienteering problem (TD-OPSWTW) are added to the literature. Time-dependency complicates the way solutions can be updated. Therefore, apart from studying the literature on the regular orienteering problem, inspiration was found in the literature on time-dependent vehicle routing problems. More specifically, for the development of an ant colony system for the TD-OP the work of Donati et al. [36], Garcia et al. [52] and Ke et al. [74] was studied. For applying the solution method for the TD-OPTW on a realistic road network the work of Donati et al. [36] is very useful. During the development of the TD-OPSWTW, the contributions of Campbell et al. [21] and Lecluyse et al. [88] were the most interesting. The differences between these works and the TD-OPSWTW will be discussed in Section 3.3.2.

3

Problems descriptions

In this chapter we will present a mathematical formulation for the TD-OP, TD-OPTW and the TD-OPSWTW. Secondly, the instance generation process for each of these problems will be discussed.

3.1 Time-dependent orienteering problem (TD-OP)

Before the MIP of the TD-OP is discussed in Section 3.1.2, the way congestion is mathematically modelled by a so called speed model is explained in Section 3.1.1.

3.1.1 A time-dependent travel speed model

In time-dependent routing problems, a speed model is typically used to determine the travel time between two vertices on a specific moment in time. In this section, the speed model for the TD-OP, together with the necessary input data and assumptions, is discussed. This speed model is based on the speed model of Ichoua et al. [66] and Donati et al. [36] for the TD-VRP. However, different arc categories and different time periods with a non-equal width are used. These choices were made based on an interview with representatives of Be-Mobile, a company specialised in the collection of travel time data [14]. More importantly, during the construction of the problem instances the arcs are not randomly assigned to an arc category, but in a space consistent way.

In our speed model, the average speed, and therefore the travel time, of a vehicle on an arc depends on the time periods it is travelling in and the arc category.

Table 3.1: Used speed matrix

Congestion description Time periods ($\underline{tp}_k - \overline{tp}_k$)	Morning peak 7am–9am	Normal 9am–5pm	Evening peak 5pm–7pm	Normal 7pm–9pm
Arc categories (c)				
1. Always busy	0.5	0.81	0.5	0.81
2. Morning peak	0.5	0.7	1	1.5
3. Two peaks	0.5	1.5	0.5	1.5
4. Evening peak	1	1.5	0.5	0.7
5. Seldom travelled	1.5	1.5	1.5	1.5

Four unequal time periods, reflecting a congestion peak in the morning and in the evening, alternated by a period with normal traffic conditions, are incorporated into the speed model. In addition to this, five arc categories are also included.

- *Always busy*: these arcs represent busy city centres with a lot of traffic during the whole day
- *Morning peak*: these arcs represent roads leading from a living area to the city centre, typically congested in the morning
- *Two peaks a day*: these arcs represent roads near the highway with a morning and evening peak in both directions
- *Evening peak*: these arcs represent roads leading from a city centre to a living area, typically congested in the evening
- *Seldom travelled*: these arcs represent roads in rural and less travelled areas

A speed $v_{c,k}$ is defined for every combination of time period k , $1 \leq k \leq K$ and arc category c , $1 \leq c \leq C$. The used speed matrix with $K = 4$ and $C = 5$ is displayed in Table 3.1. Note that the average speed in the displayed speed matrix is equal to 1, resembling the speed in the *time-independent* problem. Furthermore, we assume that the time periods are the same for every arc category. This speed model adheres to the FIFO principle. That is, leaving a vertex earlier guarantees that one will arrive earlier at the destination.

To calculate the travel time between two vertices (i and j), we need the distance $d_{i,j}$ between these vertices and the departure time t_d . The departure time defines the starting time period and together with the arc category we can look up the starting speed $v_{c,k}$ in the speed matrix. When the vehicle travels, the end of its starting time period can be reached. At that moment a new time period, together with a new speed, will be entered. This is repeated until the vehicle arrives at vertex j . The travel time between two vertices can be calculated by adding the sub-travel times in every time period until vertex j has been reached. This procedure has

been adopted from Ichoua et al. [66] and the corresponding pseudocode is listed below in Algorithm 1. Let $\underline{tp}_k$ and $\overline{tp}_k$ represent the moment at which time period k starts and ends respectively. We suppose that the vehicle leaves vertex i at the departure time $t_d \in [\underline{tp}_k, \overline{tp}_k[$ and that the arc (i, j) belongs to category c . Next, t denotes the current time and t_a denotes the arrival time.

Algorithm 1 Travel time calculation - input: i, j, t_d

```

 $t \leftarrow t_d$ 
 $d \leftarrow d_{i,j}$ 
find  $k$  such that  $t_d \in [\underline{tp}_k, \overline{tp}_k[$ 
 $t_a \leftarrow t + (d/v_{c,k})$ 
while ( $t_a > \overline{tp}_k$ ) do
     $d \leftarrow d - v_{c,k} \cdot (\overline{tp}_k - t)$ 
     $t \leftarrow \overline{tp}_k$ 
     $t_a \leftarrow t + (d/v_{c,k+1})$ 
     $k \leftarrow k + 1$ 
end while
Return  $\text{travel\_time} = (t_a - t_d)$ 

```

3.1.2 Mathematical formulation

This section describes a Mixed Integer Programming (MIP) formulation for the TD-OP, based on the MIP for the OP formulated by Vansteenwegen et al. [145] and described in Section 2.1. Formally, the TD-OP can be described by defining a set $V_c = 1, \dots, v$ of vertices. In this set, vertex 1 represents the start depot and vertex v the end depot. We assume there is an arc (i, j) between all i and j in V_c . Associated with each vertex $i \in V_c$ is a non-negative reward r_i . The reward of the depots is equal to 0. The decision variables and parameters used in this model are listed below:

Decision variables

$x_{i,j,t}$: 1 if a vehicle traverses the arc (i, j) with a departure time in time slot t , 0 otherwise

$w_{i,j,t}$: departure time in time slot t when travelling from i to j

Parameters

$\theta_{i,j,t}$: slope coefficient of the linear time-dependent travel time as defined in Equation 3.3a

$\eta_{i,j,t}$: intercept coefficient of the linear time-dependent travel time as defined in Equation 3.3b

$\tau_{i,j,t}$: start time of time slot t for arc (i, j)
 $\kappa_{i,j}$: number of time slots for arc (i, j) given by Algorithm 2
 r_i : reward of vertex i
 t^{max} : time budget

$$\text{Max} \sum_{i=2}^{v-1} \sum_{j=2}^v \sum_{t=1}^{\kappa_{i,j}} r_i x_{i,j,t} \quad (3.1a)$$

$$\sum_{j=2}^v x_{1,j,1} = \sum_{i=1}^{v-1} \sum_{t=1}^{\kappa_{i,v}} x_{i,v,t} = 1 \quad (3.1b)$$

$$\sum_{i=1}^{v-1} \sum_{t=1}^{\kappa_{i,h}} x_{i,h,t} = \sum_{j=2}^v \sum_{t=1}^{\kappa_{h,j}} x_{h,j,t} \leq 1 \quad \forall h = 2, \dots, v-1 \quad (3.1c)$$

$$\sum_{i=1}^{v-1} \sum_{t=1}^{\kappa_{i,h}} [w_{i,h,t} + (\theta_{i,h,t} \cdot w_{i,h,t} + \eta_{i,h,t} \cdot x_{i,h,t})] = \sum_{j=2}^v \sum_{t=1}^{\kappa_{h,j}} w_{h,j,t} \quad (3.1d)$$

$$\forall h = 2, \dots, v-1$$

$$x_{i,j,t} \cdot \tau_{i,j,t} \leq w_{i,j,t} \leq x_{i,j,t} \cdot \tau_{i,j,t+1} \quad i = 1, \dots, v-1, j = 2, \dots, v, \forall t \quad (3.1e)$$

$$\sum_{i=1}^{v-1} \sum_{j=2}^v \sum_{t=1}^{\kappa_{i,j}} [\theta_{i,j,t} \cdot w_{i,j,t} + \eta_{i,j,t} \cdot x_{i,j,t}] \leq t^{max} \quad (3.1f)$$

$$w_{1,i,1} = 0 \quad \forall i = 1, \dots, v \quad (3.1g)$$

$$x_{i,j,t} \in (0, 1); 0 \leq w_{i,j,t} \leq t^{max} \quad \forall t, i, j = 1, \dots, v \quad (3.1h)$$

The decision variable $x_{i,j,t}$ is equal to 1 when travelling from i to j with a departure time in time slot t , and equals 0 otherwise. $w_{i,j,t}$ is a continuous decision variable which contains the departure time at vertex i when travelling to vertex j in time slot t . Note that both decision variables, $x_{i,j,t}$ and $w_{i,j,t}$, are equal to zero when there is no route from i to j in time slot t in the solution. In short, $w_{i,j,t}$ becomes a proxy for $x_{i,j,t}$ when checking if a route is scheduled between two vertices in time slot t . This is a useful property to avoid the multiplication of decision variables in constraints (3.1d) and (3.1f).

The objective function and the first constraint are similar to the time-independent orienteering problem apart from the fact that they need to be applied for all time slots t . The objective function (3.1a) maximizes the total collected reward. Constraint (3.1b) guarantees that the route starts in vertex 1 and ends in vertex v . Constraints (3.1c) and (3.1d) ensure the connectivity of both route and travel time. More specifically Constraint (3.1c) guarantees that every vertex is visited at most once. Next, Constraint (3.1d) guarantees that the departure time of a succeeding vertex in the route is equal to the sum of the departure time of the previous vertex together with the travel time between these two vertices. The travel time is

calculated as a linear function. In this function the departure time is multiplied with a parameter $\theta_{i,j,t}$, whereafter parameter $\eta_{i,j,t}$ is added. The main difference with the regular MIP formulation of the OP lies in constraints (3.1e) and (3.1f) as the fixed distance is replaced by the time-dependent travel time. Constraint (3.1e) categorizes the departure time in the right time slot which is necessary to multiply the departure time with its corresponding θ and η in Constraint (3.1f).

In addition to this, the following assumptions were made: a route must start in time slot one (Constraint 3.1g) and no waiting is allowed (Constraint 3.1d). These assumptions are motivated by the fact that our travel times are generated by a speed model that is conform the FIFO principle (Section 3.1.1) which implies that waiting has no benefit, nor in theory, nor in practice. Constraint (3.1f) ensures the limited time budget.

To execute this MIP, we need to determine an efficient set of time cut off points (*time slots*) for each arc ($\tau_{i,j,t}$). Please note the difference between *time slots* and *time periods* in the following paragraphs. A set of $K = 4$ different *time periods* is used to define different speed values per arc category and per day segment in Section 3.1.1. The exact number of *time slots* is unique per arc and depends on the number of *time periods* (K) and the actual values of the speed matrix. Each time an increase or decrease is found in the travel time between i to j , this moment in time will be stored. This is stored as the lower limit of the time slot called $\tau_{i,j,t}$, together with two corresponding linear regression coefficients ($\theta_{i,j,t}$ and $\eta_{i,j,t}$). These linear regression coefficients allow the travel time to be calculated as follows:

$$\text{travel_time}_{i,j,w_{i,j,t}} = \theta_{i,j,t} * w_{i,j,t} + \eta_{i,j,t} \quad (3.2)$$

The procedure to find the efficient set of *time slots* for each arc consists of a combination of a forward arrival time calculation and a backward departure time calculation when the end of a *time period* is crossed. This procedure is displayed in the pseudocode of Algorithm 2. Intuitively, the goal is to search for certain moments in time where the rate of the increase or decrease in travel time changes in relation to the departure time. This way if we know these moments and their corresponding rates, the complete relation between the travel time and departure time is captured.

At the start of this procedure, the variable `border` is equal to the lower limit of the first *time period* ($tp_{k=1}$). Subsequently, we calculate the arrival time (t_a) if we depart at `border`. The `border` is stored as a *time slot* limit together with the corresponding travel time. Subsequently, `border` is increased to the lower limit of the next *time period* $k + 1$.

Next, a while loop is executed until `border` is equal to tp_K (lower limit of the last time period). First, in this loop, we calculate the departure time (t_d) that enables t_a to be equal to `border`. The t_d is stored as a lower limit together with its corresponding travel time. Subsequently, we calculate the t_a when we depart at `border` and store this information. Finally, `border` is set equal to the lower

Algorithm 2 Time slot finder algorithm

```

for all pairs  $(i,j)$  do
   $t = 1, k = 1$ 
   $\text{border} = \underline{tp}_k$ 
  Calculate  $t_a$  when  $t_d = \text{border}$ 
  Store  $\text{border}$  as  $\tau_{i,j,t}$  together with the  $\text{travel\_time}_t = t_a - t_d$ 
   $t = t + 1$ 
   $k = k + 1$ 
   $\text{border} = \underline{tp}_k$ 
  while  $\text{border} < \underline{tp}_K$  do
    Calculate the corresponding  $t_d$  when  $t_a = \text{border}$ :  $t_d = \text{border} - \text{travel\_time}_{t-1}$ 
    Store  $t_d$  as  $\tau_{i,j,t}$  together with the  $\text{travel\_time}_t$ 
     $t = t + 1$ 
    Calculate the corresponding  $t_a$  when  $t_d = \text{border}$ 
    Store  $\text{border}$  as  $\tau_{i,j,t}$  together with the  $\text{travel\_time}_t$ 
     $t = t + 1$ 
     $k = k + 1$ 
     $\text{border} = \underline{tp}_k$ 
  end while
  Store  $\text{border}$  as  $\tau_{i,j,t}$  together with the  $\text{travel\_time}_{t-1}$ 
   $\kappa_{i,j} = t - 1$ 
end for

```

Figure 3.1: An example of finding an efficient set of time slots for an arc with $d = 0.5$ and belonging to arc category 1

limit of the next time period $k + 1$.

This procedure is repeated for every arc and is visually represented for one arc in Figure 3.1 using the speed matrix from Section 3.1.1. On this figure the x -axis represents the time divided into K time periods. On the left y -axis the time slots and on the right y -axis the travel time are displayed. The violet dots represent the time slot limits that are stored in the travel time matrix. The first violet dot lying on the orange line is calculated before the start of the while loop. The violet dots lying on the black lines correspond to the backward departure time calculations at the start of the while loop. The violet dots on the other orange lines correspond to the succeeding forward arrival time calculations. As you can see, the violet dots correspond with the start of an increase/decrease in the travel time function. Based on the travel time information and the efficient set of time slot limits $(\tau_{i,j,t})$, $\theta_{i,j,t}$ and $\eta_{i,j,t}$ can be calculated as follows:

$$\theta_{i,j,t} = \frac{\text{travel_time}_{i,j,\tau_{i,j,t+1}} - \text{travel_time}_{i,j,\tau_{i,j,t}}}{\tau_{i,j,t+1} - \tau_{i,j,t}} \quad (3.3a)$$

$$\eta_{i,j,t} = \text{travel_time}_{i,j,\tau_{i,j,t}} - \theta_{i,j,t} * \tau_{i,j,t} \quad (3.3b)$$

3.1.3 TD-OP instance generation

To test the developed solution procedures, adequate datasets and appropriate performance measures are needed. So far no benchmark datasets have been developed

for the time-dependent orienteering problem. Therefore, we have created new datasets based on the ones available for the (time-independent) team orienteering problem [24, 145].

To create a time-dependent dataset every arc needs to be assigned to an arc category. The datasets commonly used to test TD-VRP solution methods have been generated by randomly assigning arcs to an arc category [36, 62, 66, 143].

This can lead to the situation depicted in Figure 3.2 and to datasets that might not be challenging enough to test a solution method that is designed to execute on congested networks. As can be seen in Figure 3.2, travelling from C to B or B to C results in the same evening peak category, furthermore there is no logical relationship between the morning and evening peaks. Thirdly, the always busy category from B to A is easily bypassed through C in the morning. Both time and spatial consistency as explained in Section 2.6, have to be taken into account when creating benchmark instances for time-dependent vehicle routing problems.

Evaluating solution procedures based on datasets where arcs are randomly assigned to arc categories might therefore lead to incorrect performance conclusions as it is, for instance, easier than in practice to find alternative routings, which requires less sophisticated algorithms. This issue is also mentioned by Figliozzi

Figure 3.2: Arcs randomly assigned to an arc category

[45] who developed a set of accessible datasets for the TD-VRP. However, in these datasets of Figliozzi [45], the same speed profile is used for all arcs, which also might lead to less challenging test problems and therefore a different procedure is developed. To avoid a random assignment of arcs to arc categories, existing datasets of the team orienteering problem have been transformed to time-dependent instances by assigning the arcs in an intelligent way to a congestion pattern. Although most of the work could be automated some manual assignments are still needed to make the resulting datasets much more realistic, as will be ex-

plained below.

More specifically, seven well-known graphs, first published by Chao et al. [24] for the team orienteering problem, have been transformed to seven TD-OP datasets. First, city centres, highways and residential zones are arbitrarily marked on each of the seven graphs (one graph per instance). Thereafter, arcs were assigned to one of the five categories based on a number of rules concerning these zones. The four basic rules that were used are:

- Arcs situated completely in a city centre are assigned to the *always busy* category;
- Arcs situated completely in the highway zones are added to the *two peaks* category;
- Travelling from a residential area to a city centre results in a *morning peak* congestion and an *evening peak* is observed when travelling in the opposite direction;
- Arcs that run from or to a vertex which is not a member of a certain zone are assigned to the *seldom travelled* category.

Examples of these rules, and other types of assignments, are displayed in Figure 3.3. This figure also demonstrates that, in practice, a specific combination of two zones not always results in the same arc category between these zones. Especially arcs between vertices situated in non-connected zones (non-neighbouring zones) have to be treated differently than arcs of vertices belonging to connected zones. For example, travelling from a vertex in a highway zone to a vertex in a nearby commuting zone results in the evening peak category, however travelling from that same vertex to a remote commuting zone member results in the seldom travelled category. As a result, we decided to assign a limited number of arcs based on the author's insight, in order to enhance the realistic representation of the datasets.

Subsequently, it is assumed that one-day trips are planned, starting at the start depot at 7 am and ending at the end depot before 9 pm, allowing for a time budget of 14 hours. Varying this available travel time (t^{max}) within a certain range, allows to create multiple instances based on the same graph information, i.e., the same vertices, arcs and time periods. In order to ensure that the instances are difficult enough and that enough vertices can be visited in a time-dependent instance with a lower time budget, the relative distance between the vertices has been rescaled by a specific factor per dataset.

The importance of this rescaling is supported by Vansteenwegen [144], who indicates that the most difficult (time-independent) OP instances are those for which the selected number of vertices is slightly more than half of the total number. If

the time budget allows the selection of half of the vertices, the largest possible number of selections will have to be evaluated by the algorithm. Moreover, determining a route between the selected vertices becomes more time consuming when the number of vertices increases. Therefore, creating instances which contain this property is important to ensure that the benchmark instances are challenging. All datasets, together with the developed solution procedure, are available at: <http://www.mech.kuleuven.be/en/cib/op/>.

Figure 3.3: Fictive example of a time-dependent orienteering problem instance

Note that the instance development procedure implies that arcs are often asymmetrically added to an arc category (e.g. morning peak from A to B and evening peak from B to A). This has considerable implications for the travel time calculation and the local search moves (Section 4.2.1 & 4.2.2).

3.2 Time-dependent orienteering problem with time windows (TD-OPTW)

For this problem no speed model is required as instances will be used where real travel time estimates are available for all arcs and for all time periods. This means that for each arc the number of time slots κ is equal to the number of time periods K .

3.2.1 Mathematical formulation

Formally, the TD-OPTW can be described by defining a set $V_c = 1, \dots, v$ of vertices. In this set vertex 1 represents the start depot and vertex v the end depot. We assume there is an arc (i, j) between all i and j in V_c . Associated with each vertex $i \in V_c$ is a non-negative reward r_i . This reward is earned by visiting the vertex for a duration of s_i between its opening time o_i and closing time c_i . The vertex can not be visited at other moments in time. Note that the time window is defined for the start of the service and not the end. The reward of the depots is equal to 0.

The calculation of the travel time between two vertices (i and j) is similar to the procedure proposed for the TD-OP in Section 3.1.2. We assume that a day is divided in κ time slots and let $\underline{tp}_t$ and $\overline{tp}_t$ represent the moment at which time slot t starts and ends respectively. Subsequently, based on these time slots and their corresponding travel times for each arc, the linear travel time coefficients for each arc $\mu_{i,j,t}$ and $\nu_{i,j,t}$ can be calculated as follows:

$$\mu_{i,j,t} = \frac{\text{travel_time}_{i,j,\overline{tp}_t} - \text{travel_time}_{i,j,\underline{tp}_t}}{\overline{tp}_t - \underline{tp}_t} \quad (3.4a)$$

$$\nu_{i,j,t} = \text{travel_time}_{i,j,\underline{tp}_t} - \mu_{i,j,t} * \underline{tp}_t \quad (3.4b)$$

These linear regression coefficients allow the travel time from i to j , when departing at a departure time ($w_{i,j,t}$) in time slot t , to be calculated as follows:

$$\text{travel_time}_{i,j,w_{i,j,t}} = \mu_{i,j,t} * w_{i,j,t} + \nu_{i,j,t} \quad (3.5)$$

The decision variables and other parameters used in the MIP model are listed below:

Decision variables

$x_{i,j,t}$ = 1 if a vehicle traverses the arc (i, j) with a departure time in time slot t , 0 otherwise

$w_{i,j,t}$ = departure time at vertex i when travelling from i to j in time slot t , 0 otherwise

Parameters

$\mu_{i,j,t}$: slope coefficient of the linear time-dependent travel time as defined in Equation 3.4a

$\nu_{i,j,t}$: intercept coefficient of the linear time-dependent travel time as defined in Equation 3.4b

$\underline{tp}_t$: start of time slot t

κ : number of time slots

r_i : reward of vertex i

o_i : opening time of vertex i

c_i : closing time of vertex i

s_i : service time of vertex i

t^{max} : time budget

$$\text{Max} \sum_{i=2}^{v-1} \sum_{j=2}^v \sum_{t=1}^{\kappa} r_i \cdot x_{i,j,t} \quad (3.6a)$$

$$\sum_{j=2}^v x_{1,j,1} = \sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} x_{i,v,t} = 1 \quad (3.6b)$$

$$\sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} x_{i,h,t} = \sum_{j=2}^v \sum_{t=1}^{\kappa} x_{h,j,t} \leq 1 \quad \forall h = 2, \dots, v-1 \quad (3.6c)$$

$$x_{i,j,t} \cdot \underline{tp}_{i,j,t} \leq w_{i,j,t} \quad i = 1, \dots, v-1, j = 2, \dots, v, \forall t \quad (3.6d)$$

$$w_{i,j,t} \leq x_{i,j,t} \cdot \overline{tp}_{i,j,t} \quad i = 1, \dots, v-1, j = 2, \dots, v, \forall t \quad (3.6e)$$

$$\sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} [w_{i,h,t} + \mu_{i,h,t} \cdot w_{i,h,t} + (\nu_{i,h,t} + s_h) \cdot x_{i,h,t}] \leq \sum_{j=2}^v \sum_{t=1}^{\kappa} w_{h,j,t} \quad (3.6f)$$

$$\forall h = 2, \dots, v-1$$

$$\sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} [w_{i,v,t} + \mu_{i,v,t} \cdot w_{i,v,t} + \nu_{i,v,t} \cdot x_{i,v,t}] \leq t^{max} \quad (3.6g)$$

$$\sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} (o_h + s_h) \cdot x_{i,h,t} \leq \sum_{j=2}^v \sum_{t=1}^{\kappa} w_{h,j,t} \quad \forall h = 2, \dots, v-1 \quad (3.6h)$$

$$\sum_{i=2}^v \sum_{t=1}^{\kappa} w_{h,i,t} \leq \sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} (c_h + s_h) \cdot x_{i,h,t} \quad \forall h = 2, \dots, v-1 \quad (3.6i)$$

$$w_{1,i,1} = 0 \quad \forall i = 1, \dots, v \quad (3.6j)$$

$$x_{i,j,t} \in (0, 1) \quad i, j = 1, \dots, v \quad \forall t \quad (3.6k)$$

$$w_{i,j,t} \in [0, t^{max}] \quad i, j = 1, \dots, v \quad \forall t \quad (3.6l)$$

The continuous decision variable $w_{i,j,t}$ contains the departure time at vertex i

when travelling to vertex j in time slot t . This departure time at vertex i equals the sum of the arrival time at vertex i , the waiting time at vertex i and the service time s_i . Note that both decision variables, $x_{i,j,t}$ and $w_{i,j,t}$, are equal to zero when there is no route from i to j in time slot t in the solution. In short, $w_{i,j,t}$ becomes a proxy for $x_{i,j,t}$ when checking if a route is scheduled between two vertices in time slot t . This is a useful property to avoid the multiplication of decision variables in Constraints (3.6f) and (3.6g).

The objective function and the first three constraints are similar to the TD-OP. The objective function (3.6a) maximizes the total collected reward. Constraint (3.6b) guarantees that the route starts in vertex 1 and ends in vertex v . Constraints (3.6c) make sure that every arc is only travelled once and ensure the connectivity of the route. Constraints (3.6d) and (3.6e) determine the departure time in the right time slot which is necessary to multiply the departure time with its corresponding μ and ν in Constraints (3.6f) and (3.6g).

The difference with the TD-OP's MIP is found in Constraints (3.6f), (3.6g), (3.6h) and (3.6i). Constraints (3.6f) guarantee that the departure time of the next vertex in the route is equal to the sum of the departure time of the previous vertex together with the travel time and service time. The travel time is calculated as a linear function. In this function the departure time is multiplied with a parameter $\mu_{i,j,t}$, whereafter parameter $\nu_{i,j,t}$ is added. Constraint (3.6g) ensures the limited time budget. Constraints (3.6h) ensure that the departure time at each vertex except from the start and end depot is greater than or equal to the opening time. Constraints (3.6i) in turn, ensure that the departure time is less than or equal to the sum of closing time and service time of the vertex under consideration.

Moreover, Constraint (3.6j) states that a route must start at time zero and therefore allows no waiting at the start depot. This assumption can be made without loss of generality.

3.2.2 TD-OPTW instance generation

A set of realistic problem instances was developed based on the road network ($G = (V, A)$) of the Benelux (Belgium, The Netherlands and Luxembourg) containing 425,479 vertices (V) and 519,915 arcs (A). The historical travel time dataset, consisting of accurately recorded travel time observations every 15 minutes for a representative Tuesday, is used to calculate the travel time for each arc. The travel time dataset is recorded for a road network that covers all highways and frequently travelled roads using the floating car system of Be-Mobile [14]. The system uses vehicle probes (e.g., taxi's, commercial vehicles, private cars, etc.) that communicate their position frequently to a central system. The individual data samples are processed to generate a traffic state for each individual road segment. The system is fully operational since October 2007.

More specifically, there are travel time estimates for 96 time periods $k \in K$ per day (with a duration of 15 minutes) for each arc and each arc has a free-flow $ff_{i,j}$ estimate, which is the smallest time needed to traverse the arc.

A set of instances was created by randomly selecting respectively 20, 50 and 100 unique vertices ($V_c \subseteq V$) out of this road network and providing them with a reward, opening and closing time and a service time. The other vertices ($V \setminus V_c$) in the graph can not be visited but might be traversed to reach the vertices in V_c . Two vertices out of V_c are selected as start and end depot.

The reward for each regular vertex is generated by selecting a random number from 1 to 30. The deterministic service time of each regular vertex is generated using a random number from a normal distribution with a mean equal to 20 minutes and a standard deviation equal to 5 minutes. The reward of both depots is set equal to 0. The opening time of the start and end depot (o_1, o_v) is set equal to 6 am. Furthermore, for each set of vertices, 4 variations in t^{max} (8, 10, 12 and 14 h), resulting in 4 corresponding closing times of the depots (2, 4, 6 and 8pm), and 3 random variations in time window widths (small, medium and large) were created. In total 36 problem instances were created which can be found at the following url: <http://www.mech.kuleuven.be/en/cib/op/>.

The time-independent and time-dependent travel times of the corresponding graph are constructed before the start of the solution procedure. The time-independent travel time (ff) between each pair of vertices from the set of vertices to visit $V_c \in V$, is calculated using Dijkstra's shortest path algorithm (using binary heaps) and assuming that arcs can be traversed at their free flow estimates.

The set of time-dependent travel times is calculated by repeatedly using an adapted version of Dijkstra's algorithm with a departure time equal to the start of a time slot. The number of time slots κ is set equal to the number of time periods K . To lower the execution time, the travel time profiles are only calculated from each vertex $i \in V_c$ to its set of neighbours (this set is defined in Section 4.4.2) for all start times equal to the lower limit of all the time slots situated in the feasible time region. The lower bound of this feasible time region is equal to the start time of the time slot corresponding to $o_i + s_i$ and the upper bound is equal to the end time of the time slot where $c_i + s_i$ is situated.

The resulting time-dependent travel times per time slot are stored for each *virtual arc*. A virtual arc is a dummy arc that holds a concatenation of arcs connecting two vertices in V_c . The use of virtual arcs to handle complex road networks was originally discussed by Donati et al. [36].

When calculating the travel time for a departure at time t_d with a shortest path algorithm, the travel time obtained is the right one if t_d is equal to the lower limit of a time slot. However, during the execution of the metaheuristic, these travel time distributions are interpolated for moments in time between two such lower limits of time slots. As this is an estimation of the travel time, the shortest path

when departing at time slot t contained in the virtual arc, might differ from the actual shortest path for departure times $t_d \in]tp_t, \overline{tp_t}]$. In short, the concatenation of arcs that forms the virtual arc might no longer be the shortest path when the departure time is later than the start of the time slot. Apart from the fact that initial experiments showed that this error was very small, no infeasible solutions were encountered during any of our experiments.

After these steps, a new virtual network has been constructed. This network corresponds to a complete graph which consists of vertices representing customer locations to visit and virtual arcs with a fixed set of travel time estimates which model the travel time behaviour on a concatenation of real arcs. For simplicity, the term arc i, j is used in the remainder of this text for the virtual arc between i and j .

As an indication of the performance of the preprocessing procedure: for an instance containing 20, 50 and 100 vertices respectively 2, 5 and 11 minutes are required.

3.3 Time-dependent orienteering problem with stochastic weights and time windows (TD-OPSWTW)

3.3.1 Mathematical formulation

In this section, the TD-OPSWTW is defined based on the formulation of Campbell et al. [21]. Let $V_c = \{1, \dots, v\}$ be a set of serviceable vertices. Vertex 1 represents the start vertex and vertex v represents the end depot. Furthermore, assume there is an arc (i, j) between all i and j in V_c . Subsequently, each vertex $i \in V_c$ has apart from a deterministic non-negative reward r_i also a deterministic non-negative penalty e_i . The reward is earned by visiting the vertex between its opening time o_i and closing time c_i and the penalty is incurred if the arrival would occur after its closing time which means the vertex is not visited. This penalty represents direct payment or a loss of goodwill. In reality, there is a cost associated with a failure to service a planned customer. Note that deterministic models generally do not (have to) consider this cost. Secondly, if the standard deviation is low, the expected reward will be higher than when the standard deviation is high. As a higher standard deviation will lead to higher probability of arriving late. Therefore the risk aversion of the planner can also be controlled by adjusting the penalties. The reward of the depots is equal to 0.

The time window is defined for the start of the service and not the end. Let T_{i,j,t_d} be a non-negative stochastic variable representing the time required to traverse arc (i, j) departing at departure time t_d . We assume that the distribution on T_{i,j,t_d} is known for all i, j and t_d . Next, let s_i be the deterministic non-negative service time at vertex i .

Let the random variable A_i be the planned arrival time at vertex i . For a realization (actually observed value) of A_i , $\bar{A}_i$, we let $R(\bar{A}_i)$ be a function representing the reward earned at vertex i when arriving at time $\bar{A}_i$. We assume that $R(\bar{A}_i) = r_i$ for $\bar{A}_i \leq c_i$ and e_i otherwise. Let ω be an order or route of the vertices in the selected set $M \subseteq V_c$ which begins at the vertex 1 at time t_0 and ends at end vertex v . The opening time of the start and end vertex is set equal to t_0 and their rewards are set to 0. Subsequently, the closing time of the end vertex is set equal to $t_0 + t^{max}$ and its penalty is set equal to e_v . This way a penalty is also obtained if the route length exceeds the predetermined length, t^{max} . The expected reward of the a-priori route is equal to:

$$R(\omega) = \sum_{i \in \omega} [P(A_i \leq c_i) \cdot r_i - (1 - P(A_i \leq c_i)) \cdot e_i] \quad (3.7)$$

We seek an a-priori route ω^* such that $R(\omega^*) \geq R(\omega)$ for every ω . For this problem, an a-priori route, or pre-planned route, is a route which specifies an ordering of all selected vertices that a particular driver may service. The driver may then or may not be able to service these vertices given the realisations of the travel time.

Furthermore, the time is assumed to be discretized in κ time slots. This means that for every vertex i to j and for every time slot $t \in \kappa$ there exists a normally distributed travel time $(T_{i,j,t})$. Finally, let $\underline{t}p_t$ and $\overline{t}p_t$ represent the moment at which time slot t starts and ends respectively.

3.3.2 Position in the literature

Now that the TD-OPSWTW is defined, differences with the literature discussed in Section 2.4 can be better explained. In general only the work of Lau et al. [86] considers time-dependent stochastic travel times, all other papers use time-independent stochastic travel times. Recall that in [21, 42, 86, 134, 138] no time-windows are incorporated.

Unlike the TD-OPSWTW, the problem discussed by Teng et al. [138], is limited to discrete travel and service time distributions. The problem only uses a single penalty parameter as a mechanism for maintaining the feasibility of the solution. The problem introduced by Tang and Miller-Hooks [134] incorporates deadlines via a chance constraint rather than modelling the economic cost of their violation using a penalty as in the case of the TD-OPSWTW. When comparing the work of Campbell et al. [21] to the TD-OPSWTW, the time-dependent stochastic travel times, time windows and the constraint that routes need to start and end at the predefined vertex were added to the TD-OPSWTW. Contrary to Campbell et al. [21], the service time at a vertex is assumed to be deterministic. Apart from these additions the objective function of the TD-OPSWTW is very similar to the one proposed by Campbell et al. [21]. The difference with Lau et al. [86] is that there is no vertex specific penalty, only a general penalty for a route with a length that

exceeds the time budget. The difference between the problem presented by Evers et al. [39, 42] and the TD-OPSWTW is that no penalty is received for scheduled vertices that are not visited. Furthermore, for the TD-OPSWTW all vertices are known in advance and an a-priori route needs to be found. Additionally, in the work of Evers et al. [39] late arrivals at the end vertex are also not penalised. Different than presented in the work of Zhang et al. [158], for the TD-OPSWTW the travel times are time-dependent stochastic variables and the waiting time originates from arriving too early at a vertex. Furthermore, no recourse actions are modelled or taken into account during the evaluation of possible solutions.

In general, these solution methods do not take into account the impact of time windows on the convolution of the departure time and travel time distribution. The proposed sampling approaches only work because, when no time windows are considered, independent distributions of the same type (discrete and gamma distributions are used in most cases) can be added together. However, in case of time windows, the resulting waiting times or late arrivals significantly complicate the calculation of the departure time distribution.

3.3.3 TD-OPSWTW instance generation

In this section, the generation of realistic test instances for the TD-OPSWTW will be explained based on the available data. These test instances are based on the ones proposed for the TD-OPTW in Section 3.2.2. The difference resides in the fact that the travel times are now stochastic and a penalty is defined for every vertex. This penalty, also for the end vertex, is set independent of the magnitude of the lateness. This implies that in theory visiting additional vertices and collecting the reward before going to the end vertex might counter the collected penalty at the end vertex. In order to avoid this behaviour, the closing time of the regular vertices is set less than or equal to t^{max} . The penalty for regular vertices is equal to 20% of their respective reward. The penalty for arriving late at the end vertex is set equal to the highest reward over all regular vertices of the instance. The instances are also available at <http://www.mech.kuleuven.be/en/cib/op/>. The accompanied historical travel time data, consisting of travel time observations for every time period of 15 minutes ($K = 96$) for a representative Tuesday, is used to fit a set of normal travel time distributions for every arc during 96 time slots ($\kappa = 96$). Since for every arc and time slot combination only one travel time estimate is available, the standard deviation for a particular combination of an arc and time slot is set equal to the variability over all time slots of the arc. The average coefficient of variation ($cv = \frac{\sigma}{\mu}$) is therefore equal to 0.5%, which is rather low. Other coefficients of variation and penalty ratios ($pr = \frac{r_i}{e_i}$) will be used when the proposed algorithm is tested (Section 5.2).

The time-independent and time-dependent stochastic travel times between each

pair of vertices representing serviceable customers to visit $V_c \in V$, are constructed. The time-independent travel time ($ff_{i,j}$) is calculated using Dijkstra's shortest path algorithm and assuming that all arcs can be traversed at their free flow estimates. These estimates are also used to solve the problem instances as deterministic time-independent orienteering problems, as will be described in Section 5.2.2. Furthermore, these estimates are also used to construct the neighbourhood structure as described in Chapter 5.

The stochastic time-dependent travel time distributions (T) are calculated using an adapted version of Dijkstra's algorithm inspired by the research of Demeyer et al. [32]. The Dijkstra algorithm [33] is a label-setting algorithm with labels representing the lowest cost between the start vertex and the destination vertex. During initialization the label of the start vertex is set to zero and all other labels are set to infinity. The set P contains all (non-permanent) vertices whose labels have been updated by the algorithm. In every iteration the vertex with the lowest label is removed from the set P and made permanent. Then, all labels of the neighbouring vertices are updated. More specifically, if the sum of the label of the investigated (permanent) vertex and of the arc cost is less than the previous label of the neighbouring vertex, then the label is changed to this sum. This process is repeated until the destination vertex has been made permanent. The time-dependent stochastic Dijkstra's algorithm requires that labels are no longer single values, but probabilistic distributions of travel times. Two operations need to be defined: comparing labels and updating labels. Deciding which of two labels is the best is no unambiguous process. Usually a fixed percentile such as the 90th percentile is used to compare the labels. However, in this research the labels are compared on the 99th percentile since we want to obtain reliable routes. During the update of the labels, in the work of Demeyer et al. [32], 5 equally spread percentiles (10th, 30th, 50th, 70th, 90th) are used. we assume that the travel times on the arcs are independent (no correlation between the arcs) and normally distributed (μ, σ^2), which enables us to easily calculate the arrival distribution as the sum of the departure time distribution and the travel time distribution:

$$\mu_{arrival} = \mu_{departure} + \mu_{travel} \quad (3.8)$$

$$\sigma_{arrival}^2 = \sigma_{departure}^2 + \sigma_{travel}^2 \quad (3.9)$$

Subsequently, the arrival time distribution becomes the departure time distribution when subsequent arcs are added to a route. So unlike the research of Demeyer et al. [32], no discrete approximation of the distribution is used.

We validated our stochastic implementation using a Monte Carlo simulation on the final path returned by the Dijkstra algorithm. Our Monte Carlo simulation was also used to verify the performance of the stochastic Dijkstra procedure of Demeyer et al. [32]. We could conclude that their procedure significantly underestimates the variance of the travel time on the proposed path. A possible explana-

tion provided by Bickel et al. [16] is that the distribution is approximated using 5 percentiles that do not match the moments of the normal distribution.

The resulting time-dependent travel time distributions for each *virtual arc* are stored. As an indication of the performance of this procedure, for the set of 20, 50 and 100 vertices respectively 2, 6 and 13 minutes of calculation time is required.

3.4 Conclusion

In this chapter a mixed integer problem formulation is presented for the TD-OP and the TD-OPTW. However, since the OP and therefore also its extensions are NP-hard, these formulations can not be used to solve problem instances of a realistic size using a commercial solver.

For the TD-OP, realistic time-dependent test instances with known optimal solution are developed based on the original time-*independent* OP instances in combination with a well performing speed model for the TD-VRP. For the TD-OPTW, 36 realistic time-dependent test instances with known optimal solutions are developed with a number of vertices to visit ranging from 20 to 100. These instances were constructed by extracting vertices from a road network containing 84,720 vertices and 116,683 arcs. For the TD-OPSWTW a mathematical formulation is provided and the test instances for the TD-OPTW were modified to match the required problem input. The test instances for all problems are made publicly available at <http://www.mech.kuleuven.be/en/cib/op/>.

4

Solution methods and results for the orienteering problem with time-dependent travel times

In this chapter, the proposed solution methods for the time-dependent orienteering problem (TD-OP) and the time-dependent orienteering problem with time windows (TD-OPTW) will be explained together with the results on their respective test instances. For each problem, the notation explained during the problem descriptions in Chapter 3 is also used here.

4.1 Motivation for the selected solution method

Since high-quality solutions are required and the computation time should be limited to only a few seconds, the literature on vehicle routing suggests the implementation of a local search based metaheuristic to solve problems of realistic size [122]. In this section, a metaheuristic, based on the principles of an ant colony system (ACS), is implemented together with time-dependent local search moves to tackle the time-dependent orienteering problems. This choice is motivated by the fact that generally very complex problems require simple solution frameworks. Moreover, Bullnheimer et al. [19] state that the ACS produces starting vehicle routing problem solutions that are more easily improved by a local search move than starting solutions produced by a genetic algorithm. This choice might seem con-

trary to what in the literature base is recommended. Recall that tabu search (TS) is used frequently for the TD-VRP and iterated local search and variable neighbourhood search seem both appropriate methods to tackle various orienteering problems. However, during initial tests these frameworks and hybrid versions were considered and compared against each other. These methods performed significantly worse than the ACS. Apart from other issues the biggest drawback of TS (and VNS or ILS) is the fact that even removing a vertex from a solution can be a time consuming move. Due to the time-dependency, additional checks and repair steps consume a lot of CPU time. Moreover, the line between TS and ant colony optimization is not clear-cut, both TS and ant colony optimization make use of memory. In this specific case, we found experimentally that memory structures that evaluate the quality of arcs in a solution perform better than memory structures that prohibit certain moves based on recency information. This type of memory is more commonly found in techniques labelled ant colony optimization than in techniques labelled TS. To avoid confusion, we therefore labelled our technique ACS.

The motivation for the used set of local search moves will be explained in Section 4.6 after the technical details and results have been explained.

4.2 Solution method for the TD-OP

The ACS presented here, is based based on the ant colony optimization algorithm of Ke et al. [74] and Schilde et al. [118] for the time-*independent* OP and on the ACS of Donati et al. [36] for the TD-VRP. In order to solve the TD-OP, as for every metaheuristic, a good balance between intensification and diversification is essential [133]. Therefore, a specific *insert* local search move is designed in order to intensify the search for improvement. The strength of this insertion step lies in the fast evaluation of the possible insertion of a vertex. In order to diversify the search, each metaheuristic framework has a specific manner to escape from local optima. The pheromone trails in the ACS are depreciated (“evaporated”) during the construction procedure. The ACS diversification mechanism is discussed in more detail in Section 4.2.3.

The remainder of this section is organized as follows. First, the local search moves are discussed: the insert local search move is explained in Section 4.2.1 and the modified 2-opt move in Section 4.2.2. The ACS framework is explained in Section 4.2.3.

4.2.1 Insert local search move

The problem-specific *insert* local search move iteratively attempts to insert non-included vertices into an existing solution, thus improving its total reward. To

prevent a full and time-consuming evaluation of a solution, we store for every vertex in the current solution the maximum amount of time that its visit can be postponed before the solution becomes infeasible (`max_shift`). This enables an efficient checking and updating mechanism. The `max_shift` metric can be calculated from the last vertex to the first vertex in the solution and only needs to be updated for some vertices when an extra vertex is actually included in the solution, not when it is only considered for inclusion during the insert move. A similar method was discussed in Donati et al. [36] and the calculation is presented in pseudo code in Algorithm 3. Finally, the local search move is also visually presented in Figure 4.1.

Algorithm 3 Calculation of `max_shift` for TD-OP - input: solution sequence *sol*

```

 $t_a \leftarrow \underline{tp}_1 + t^{max}$ 
for every vertex  $i$  in solution sol except the start vertex & end vertex do
     $z \leftarrow sol_{i+1}$ 
     $y \leftarrow sol_i$ 
    find  $k$  such that  $t_a > \underline{tp}_k$ 
     $t_d \leftarrow t_a$ 
    remaining_distance  $\leftarrow d_{y,z}$ 
    distance_covered  $\leftarrow (t_a - \underline{tp}_k) \cdot v_{y,z,k}$ 
    while (remaining_distance > distance_covered) and ( $k > 1$ )
    do
        remaining_distance  $\leftarrow$  remaining_distance -
        distance_covered
         $t_d \leftarrow t_k$ 
        distance_covered  $\leftarrow (\underline{tp}_k - \underline{tp}_{k-1}) \cdot v_{y,z,k-1}$ 
         $k \leftarrow k - 1$ 
    end while
     $t_d \leftarrow t_d - \text{remaining\_distance} / v_{y,z,k}$ 
     $\text{max\_shift}_i \leftarrow (t_d - \text{actual\_departure\_time}_i)$ 
     $t_a \leftarrow t_d$ 
end for

```

When `max_shift` has been calculated for every vertex in the current solution, a list called `include` is created of all vertices that are not yet included in the solution. The vertices are added to this list in random order. Note that the `max_shift` (represented by a kite in the figure) of vertex w is greater than the remaining time at the end vertex ($4.5 - 4 = 0.5$). The reason for this is that the travel time between w and v is a decreasing function of the departure time in that particular time slot.

The insert move tries to insert a vertex from this list into the current solution, if the extra travel time required to visit this new vertex is less than the value of

`max_shift` of the succeeding vertex. For example, when the algorithm attempts to insert vertex y (member of the `include` list) between x and z , the extra time-dependent travel time equals:

$$\Delta \text{travel_time} = \text{travel_time}_{x,y} + \text{travel_time}_{y,z} - \text{travel_time}_{x,z} \quad (4.1)$$

Vertex y can only be inserted into the current solution when the extra travel time is less than or equal to the maximum amount of time vertex z can be shifted to a later moment in time or:

$$\Delta \text{travel_time} \leq \text{max_shift}_z \quad (4.2)$$

When vertex y is actually included in the solution, the algorithm updates the travel time from x to y , as well as the travel times between the vertices succeeding vertex y . This is necessary, because the insertion of vertex y has most likely caused a change in travel time for the arcs following vertex y in the solution.

The `max_shift` of the vertices succeeding vertex y can easily be adjusted (marked with an orange kite on Figure 4.1) based on the previous `max_shift` value, the previous time-dependent travel time and the new time-dependent travel time based on the following equation :

$$\text{max_shift}_i = \text{previous_max_shift}_i + (\text{previous_travel_time}_i - \text{new_travel_time}_i) \quad (4.3)$$

In this equation `travel_timei` represents the travel time to reach i from its immediate predecessor. Second, a recalculation of `max_shift` for the vertices preceding vertex z is also necessary (marked in violet).

As a result of this procedure, the computation time needed to evaluate if a vertex can be included or not, is drastically reduced. As a result, only when a vertex is actually inserted, a complete and time-consuming recalculation of the new solution is required.

Another issue that troubles the insert local search move is the triangle inequality property, which for road networks means that travelling from vertex A to C is shorter than travelling from A to B and from B to C. In a time-dependent problem however, due to traffic jams on the direct road between A and C, it might be actually faster to drive through B. The result of wrongly assuming that this property is still valid in a time-dependent setting is that one may not consider potential changes of the shortest paths due to varying travel times as mentioned by Donati et al. [36] and Fleischmann et al. [47]. In practice, the triangle property will always be respected. If travelling from A to B would be faster through C, the travel time of the path through C would be used in the model as the travel time between A and B. To model this in a proper way, that would mean that we have to check all possible triangles [A,B,C] for each possible departure time at A. We decided not to focus on this issue and we refer to Kok et al. [78] for a possible implementation.

Figure 4.1: The time-dependent insert move

As a result of not always respecting the triangle property, inserting a vertex might actually shorten the travel time of the solution. If for example the sequence AB, displayed in Figure 4.2, is included in the solution, the insert move might insert vertex C in between A en B which results in a decrease in travel time of 3.51 time units:

$$\begin{aligned}
 \text{travel_time}_{A,B,t_d} &> \text{travel_time}_{A,C,t_d} + \text{travel_time}_{C,B,[t_d + \text{travel_time}_{A,C,t_d}]} \\
 \text{travel_time}_{A,B,7} &> \text{travel_time}_{A,C,7} + \text{travel_time}_{C,B,[7 + \text{travel_time}_{A,C,t_d}]} \\
 (1/0.5) + (5/0.81) &> (3/1.5) + \text{travel_time}_{C,B,9} \\
 8.17 &> 2 + (4/1.5) \\
 8.17 &> 4.66
 \end{aligned}$$

4.2.2 Modified 2-opt

The basic 2-opt move removes two arcs and tries to replace them with two new arcs not previously included in the path. If this move reduces the travel time of the solution, the new solution is accepted, otherwise the original solution is retained [145]. This move cannot be easily used in a time-dependent environment because the direction of the part of the solution between the removed arcs is reversed. Traversing arcs in the opposite direction demands a complete and time-expensive recalculation of the solution. It might be possible that arcs that are troubled by morning congestion and therefore originally scheduled in the evening, are reinserted into the morning by a 2-opt move and therefore make the whole solution infeasible or

Figure 4.2: An example where the triangle property is not valid for the time- dependent travel times

at least undesirable.

To circumvent these time-expensive calculations, the algorithm first preselects interesting 2-opt combinations based on the gain in distance. Only, the most interesting combination (biggest gain in distance) is then evaluated in a time-dependent context. If this combination results in a feasible solution, it is accepted. It should be noted that it is not required that the travel time of the new solution is less than the travel time of the previous solution. In our proposed algorithm for the TD-OP, this 2-opt move is merely used as a diversification procedure, rather than to increase the quality of a solution.

4.2.3 Ant colony system

The ACS is based on the behaviour of a foraging ant colony. It is a constructive metaheuristic that constructs several solutions independently (each construction procedure is represented by an agent commonly called an “ant”) and uses memory structures called “pheromones trails” to mark travelled arcs and to allow communication between the different ants. The ACS framework is displayed in Algorithm 4, together with the corresponding input parameters and variables. sol_{ib} is the best solution of the current iteration, sol_{gb} represents the best solution found during the entire optimization procedure and $F(sol_x)$ refers to the objective function (total reward) of solution sol_x .

Before the start of the ACS, the value of the greedy information, $\eta_{i,j}$ for arc (i, j) is calculated as the ratio of the reward of vertex j and the travel distance to reach vertex j from vertex i . This way of working turns out to be less computational expensive in comparison to using the ratio of the reward and the time-dependent travel time to the next vertex. The pheromone value ($\tau_{i,j}$) of all arcs is initially set at a value τ_{init} .

The construct solution procedure creates `max_ants` solutions independently

Algorithm 4 Ant colony system for TD-OP - input parameters:

$\alpha, \beta^*, \rho, \text{max_ants}, \tau_{init}, N_{ni}^{max}$

$sol_{ib} \leftarrow 0, sol_{gb} \leftarrow 0, N_{ni} \leftarrow 0, \text{iteration} \leftarrow 0$

while $\text{iteration} < N_c$ **do**

 Initialize $\tau, \eta (\tau_{init})$

$\beta = \beta^* * |V_c|$

 Construct initial solutions:

for $i \leftarrow 1$ to max_ants **do**

 Construct solution & local pheromone update ($\tau, \eta, \alpha, \beta, \rho$)

 2-opt

 Calculate max_shift

 Insert (max_shift)

end for

$sol_{ib} \leftarrow \arg \max (F(sol_1), F(sol_2), \dots, F(sol_{\text{max_ants}}))$

if $F(sol_{ib}) > F(sol_{gb})$ **then**

$sol_{gb} \leftarrow sol_{ib}$

$N_{ni} \leftarrow 0$

else

$N_{ni} \leftarrow N_{ni} + 1$

end if

 Global pheromone update ($\tau, sol_{ib}, N_{ni}, N_{ni}^{max}$)

$\text{iteration} \leftarrow \text{iteration} + 1$

end while

and sequentially. Each construction starts from an empty solution and adds vertices at the end of the solution until no more vertices can be inserted due to the travel time restriction. At that point, the end vertex is added to finalize the solution, and the algorithm moves on to the next solution until `max_ants` solutions are created. Before adding a vertex following the already included vertex u , a list called C_u of all vertices feasible to include is created. This is done by calculating the time-dependent travel time between the last included vertex u and a vertex under consideration plus the travel time between the vertex under consideration and the end vertex. If the sum of both travel times together with the total travel time of the solution is less than t^{max} , the vertex under consideration can be added to the solution. Afterwards, each vertex in the list receives a probability to be included. This probability (l_i) that vertex i will be added to the solution is calculated as follows:

$$l_i = \frac{\tau_{u,i}^\alpha \cdot \eta_{u,i}^\beta}{\sum_{w \in C_u} \tau_{u,w}^\alpha \cdot \eta_{u,w}^\beta} \quad \forall i \in C_u \quad (4.4)$$

In this equation, α determines how much weight is given to the pheromone value and β defines how much weight is given to the greedy information. Then, a random number is generated to determine, together with the probability l_i , which vertex from the list is added to the solution by using a roulette wheel selection method. Note that the relative values of α and β together determine how much weight is given to the randomness in the selection procedure. For example, the random numbers will have less impact if α and β both equal 3 than when they both equal 1, as the difference in τ or η values of feasible vertices is disproportionately reflected in their l_i values. A vertex that has a slightly better value of τ than another vertex will have a disproportional greater l_i value and thus a higher probability to be selected. In other words, the procedure becomes more greedy. Subsequently, the τ value related to the newly added arc in the solution is decreased. This local pheromone procedure which enhances diversification is executed as follows:

$$\tau_{u,i} = \tau_{u,i} \cdot (1 - \rho) \quad (4.5)$$

ρ is called the evaporation rate and is usually set at a rather low value (e.g. 0.01). The evaporation procedure attempts to enhance the diversity of the solution procedure by preventing the same arcs from being added to a large number of solutions. Note that the pheromone values should be prevented from becoming very small, leading to a possible (near) division by zero in Equation 4.4. When a τ -value is close to zero, it is reset to τ_{init} . After the construction of `max_ants` solutions, the insert and 2-opt moves are executed on all constructed solutions. The modified 2-opt move evaluates all possible 2-opt moves but only the most interesting move is executed. Since executing a 2-opt move is computationally very expensive, the procedure is stopped after one execution in order to limit the computation time.

The insert move however, is executed in a first-improving manner and stops when no more feasible improvements can be found.

The solution of each iteration with the highest reward is stored (s_{ib}) and arcs that are used in this solution are made more attractive in the solution construction procedure of succeeding iterations, by increasing their corresponding pheromone value:

$$\tau_{i-1,i} \leftarrow \tau_{i-1,i} + \tau_{init} \quad \forall i \in s_{ib} \mid i > 1 \quad (4.6)$$

This makes it more likely that these arcs will be used in a subsequent construction procedure (intensification). If the reward (sol_{ib}) is better than the global best reward found during previous iterations (sol_{gb}), sol_{gb} is updated. Finally, these steps are repeated N_c times but to prevent that only a couple of arcs dominate in the solution construction procedure (local optima), the pheromone values are reset to τ_{init} when no improvement can be found during a certain number of iterations (N_{ni}^{max}). This means that all arcs have again an equal probability to be chosen during the next construction procedure, allowing diversification.

4.3 Results for the TD-OP

4.3.1 Comparison to optimal solutions of small instances

The first experiment consists out of a straightforward comparison of the results of the ACS and the optimal solution found by solving the MIP from Section 3.1.2. However, most of the time-dependent instances of Section 3.1.3 are too complex to be solved with a commercial solver using the MIP formulation developed in Section 3.1.2. Since this is not the focus of our research only a basic implementation of this MIP was implemented using the commercial solver, CPLEX 12.5 (64-bit) on a computer with an i5 2.6 GHz processor and 8 GB of memory. The choice to apply this commercial solver to the MILP of the TD-OP was motivated by the fact that commercial solvers like CPLEX and Gurobi are known to be very effective in tackling this kind of problems. We refer to the exact solution methods of [6, 46, 67] for variants of the OP and CPLEX is also used for vehicle routing problems in a time-dependent context by Kok et al. [79]. The restriction of 48 hours (!) of computation time per instance allowed to find an optimal solution for most of the instances of the first 3 datasets. The comparison between the results of the ACS and these optimal solutions for five independent runs of the algorithm is displayed in Table 4.1. In the name of each instance, the number refers to the graph, originally developed by Chao et al. [24] that is used. The characters a to i refer to increasing values of t^{max} .

As performance metric the CPU time together with the percentage gap between the total reward of the optimal solution and the total reward of the heuristic solution

is used:

$$\text{gap} = \frac{\text{optimal reward} - \text{heuristic reward}}{\text{optimal reward}} \quad (4.7)$$

Both the average gap and the minimum and maximum gap on all instances are recorded during our analysis. Optimal % gaps are marked in bold in the Table 4.1.

The average gap is very low at 0.8%. Furthermore, the known optimal solution could be found for 14 out of 24 test instances. Studying the computation time leads to the conclusion that the ACS is very fast, as on average only 0.1 seconds are needed to obtain a solution. The maximum observed CPU time was 0.2 seconds.

The results prove the validity of our MIP model and the excessive time needed to solve these small instances using a commercial solver. The required computation time depends heavily on the number of time slots needed per arc and they exponentially increase with the value of t^{max} . Furthermore, this table illustrates the near-optimal results and the very short computation times of the ACS on small problem instances.

4.3.2 Comparison to known optimal solutions of large instances

To evaluate more thoroughly the performance of the TD-OP metaheuristic, it would be useful to also have optimal solutions for larger benchmark instances. In order to achieve that, all instances were solved first as time-*independent* OPs using the MIP displayed in Section 2.1. During this optimization, the travel time was calculated using, on each arc, the maximum speed of its corresponding arc category. Still not all time-*independent* OP instances could be solved due to time and memory limitations. The excessive CPU time needed to solve these time-*independent* OP instances (not displayed), often more than 100 hours, illustrates again that it would be pointless to try to solve large time-dependent OPs to optimality, certainly based on these straightforward MIP formulations. The found optimal rewards for these time-*independent* instances are displayed in Table 4.3 (column “optimal”).

Following the optimization by CPLEX, the optimal time-*independent* solutions (sequence of vertices) are used to modify the original time-dependent instances in such a way that slightly modified time-dependent instances with known optimal solution are created. More specifically, for each arc included in the optimal time-*independent* solution, the travel speed is set to its maximal value, but only for the time slot(s) during which this virtual arc is traversed. Since the travel speed is only modified in some of the time periods of these arcs, these arcs still have time-dependent travel times. The time-dependent travel times on all other arcs are not modified. In this way, it is ensured artificially that the time-*independent* optimal solution is also an optimal solution to the modified time-dependent instance.

The procedure is explained in detail with a simple problem instance in Figure 4.3. In part A the optimal time-*independent* solution found by CPLEX using the MIP model for the OP (Section 2.1) is displayed. In the next part, the travel speed

Table 4.1: Comparison to optimal solutions of small instances

	$ V_c $	t^{max} h	CPLEX		ACS			
			optimal reward	cpu s	best %gap	avg %gap	worst %gap	cpu sec
1.a	32	5	115	41	0.0	0.0	0.0	0.2
1.b	32	6	135	143	0.0	0.0	0.0	0.1
1.c	32	7	160	351	0.0	0.0	0.0	0.1
1.d	32	8	185	515	0.0	2.2	5.4	0.1
1.e	32	9	210	687	0.0	0.5	2.4	0.1
1.f	32	10	230	37,514	0.0	0.4	2.2	0.1
1.g	32	11	250	6,787	0.0	0.0	0.0	0.1
1.h	32	12	270	70,225	1.9	1.9	1.9	0.1
2.a	21	5	100	11	0.0	0.0	0.0	0.0
2.b	21	6	150	24	0.0	0.0	0.0	0.0
2.c	21	7	195	35	0.0	0.0	0.0	0.1
2.d	21	8	220	49	0.0	0.0	0.0	0.1
2.e	21	9	260	126	0.0	0.0	0.0	0.1
2.f	21	10	310	383	0.0	0.0	0.0	0.1
2.g	21	11	340	671	0.0	0.0	0.0	0.1
2.h	21	12	375	5,356	0.0	0.0	0.0	0.1
2.i	21	13	425	14,217	0.0	0.0	0.0	0.1
3.a	33	5.5	370	96	0.0	4.3	5.4	0.1
3.b	33	6.5	420	272	0.0	0.0	0.0	0.1
3.c	33	7.5	500	371	4.0	4.0	4.0	0.1
3.d	33	8.5	560	503	0.0	2.9	3.6	0.1
3.e	33	9.5	620	662	0.0	1.9	4.8	0.2
3.f	33	10.5	650	1,928	0.0	0.3	1.5	0.1
3.g	33	11.5	690	16,704	0.0	1.2	2.9	0.2
				max	4.0	4.3	5.4	0.2
				avg	0.2	0.8	1.4	0.1
% optimal					91.7	58.3	58.3	

is set to its maximal value (depending on the arc category) for the selected arcs in part A and only during the time periods they are used in the time-independent solution. Second, the time-dependent arcs have different characteristics for each travelling direction. In part C, the ACS is executed on this adapted problem instance. Furthermore, note that the found solution in part C has a lower objective value than the optimal solution in part B which results in a gap of 16.7%.

Figure 4.3: Example of creating a time-dependent instance with a known optimal solution

The creation of the benchmark instances in this way allows a comparison of the performance of the developed solution methods with known optimal solutions for larger instances. For five independent runs of the algorithm, these results are displayed in Table 4.3. In Table 4.2 the % gap per dataset is displayed. The effect of the parameters that are used in the metaheuristic and their exact values used in these experiments are discussed in detail in Section 4.3.3.

Table 4.2: % gap (best, average, worst) and average CPU time per dataset

Dataset	$ V_c $	best % gap	avg % gap	worst % gap	CPU s
1	31	0.0	0.2	0.2	0.1
2	21	0.0	0.0	0.0	0.1
3	33	0.0	0.2	0.7	0.1
4	100	1.8	2.8	3.9	1.1
5	66	1.2	1.4	2.2	0.3
6	64	0.3	0.9	1.7	0.5
7	102	3.1	3.9	5.0	1.1

Table 4.3: Comparison to known optimal solutions of large instances

	$ V_c $	t^{max} hours	optimal reward	best % gap	avg % gap	worst % gap	CPU s
1.a	32	5	135	0.0	0.0	0.0	0.1
1.b	32	6	165	0.0	0.0	0.0	0.1
1.c	32	7	185	0.0	0.0	0.0	0.1
1.d	32	8	210	0.0	0.0	0.0	0.1
1.e	32	9	240	0.0	0.0	0.0	0.1
1.f	32	10	260	0.0	1.5	1.9	0.1
1.g	32	11	275	0.0	0.0	0.0	0.1
1.h	32	12	285	0.0	0.0	0.0	0.1
1.i	32	13	285	0.0	0.0	0.0	0.1
2.a	21	5	165	0.0	0.0	0.0	0.0
2.b	21	6	200	0.0	0.0	0.0	0.1
2.c	21	7	225	0.0	0.0	0.0	0.1
2.d	21	8	275	0.0	0.0	0.0	0.1
2.e	21	9	315	0.0	0.0	0.0	0.1
2.f	21	10	375	0.0	0.0	0.0	0.1
2.g	21	11	415	0.0	0.0	0.0	0.1
2.h	21	12	440	0.0	0.0	0.0	0.1
2.i	21	13	450	0.0	0.0	0.0	0.1
3.a	33	5.5	430	0.0	0.0	0.0	0.1
3.b	33	6.5	490	0.0	0.0	0.0	0.1
3.c	33	7.5	550	0.0	0.0	0.0	0.1
3.d	33	8.5	590	0.0	0.0	0.0	0.1
3.e	33	9.5	630	0.0	0.0	0.0	0.1
3.f	33	10.5	680	0.0	0.0	0.0	0.2
3.g	33	11.5	730	0.0	0.0	0.0	0.2
3.h	33	12.5	770	0.0	1.0	2.6	0.1
3.i	33	13.5	800	0.0	0.8	3.8	0.1
4.a	100	5	486	0.0	0.8	1.0	0.7
4.b	100	6	590	0.5	1.7	3.1	0.8
4.c	100	7	679	1.6	2.5	3.5	0.9
4.d	100	8	771	3.6	4.5	5.1	1.0
4.e	100	9	853	3.4	4.2	5.5	1.1
4.f	100	10	932	2.1	3.0	4.3	1.2
4.g	100	11	1007	0.0	0.9	4.1	1.2
4.h	100	12	1083	1.7	3.0	3.6	1.3
4.i	100	13	1147	1.3	1.7	2.4	1.3
4.j	100	14	1198	4.2	5.3	6.4	1.4
5.a	66	5.5	580	0.0	0.3	1.7	0.3
5.b	66	6	650	2.3	2.3	2.3	0.3
5.c	66	7	770	1.3	1.6	2.6	0.4
6.a	64	6.5	870	0.0	0.3	0.7	0.5
6.b	64	7	930	0.0	0.6	1.3	0.4

6.c	64	8	1056	0.0	0.9	2.3	0.5
6.d	64	9	1152	0.5	0.8	2.1	0.5
6.e	64	10	1236	0.0	1.1	1.9	0.6
6.f	64	11	1308	0.9	1.8	2.8	0.5
6.g	64	12	1344	0.9	1.5	2.2	0.7
6.h	64	13	1344	0.0	0.0	0.0	0.5
6.i	64	14	1344	0.0	0.0	0.0	0.5
7.a	102	4	532	4.5	5.3	7.0	0.7
7.b	102	5	648	4.9	6.1	9.0	0.8
7.c	102	6	774	0.8	1.1	1.7	1.0
7.d	102	7	884	1.7	1.8	2.1	1.1
7.e	102	8	994	4.7	6.1	7.0	1.1
7.f	102	9	1090	1.8	2.5	3.7	1.2
7.g	102	10	1175	1.2	3.1	4.9	1.3
7.h	102	11	1251	3.6	4.6	5.1	1.3
7.i	102	12	1317	2.7	3.2	3.5	1.4
7.j	102	13	1368	5.0	5.6	6.3	1.4
			max	5.0	6.1	9.0	1.4
			avg	1.0	1.4	2.0	0.5
			% optimal	61.0	44.1	44.1	

These results prove the high performance quality of the ACS, since the average gap is very low at 1.4%. Furthermore, the known optimal solution could be found for 26 out of 59 test instances. A second conclusion is that the average gap increases as the test instances become more complex due to a longer travel time limit and an increasing number of vertices. Studying the computation time leads to the conclusion that the ACS is very fast, as on average only one second is needed to obtain a solution. The maximum observed CPU time was 1.4 seconds, which is more than fast enough for most application purposes. Therefore, it can be concluded that the ACS is able to deliver a high performance requiring a minimal computational effort.

To stress the validity of this second test procedure, the average gap on the more limited set of instances that was used in Section 4.3.1 is 0.1% and the maximum gap equals 1.5%. These values do not deviate too much from the values of respectively 0.8% and 4.3% obtained in Section 4.3.1.

4.3.3 Sensitivity analysis

In this section, the impact of the input parameters and the design of the ACS is discussed. It is interesting to know how much the performance of the algorithm depends on the specific values of the input parameters.

A first input parameter that needs separate attention is the number of iterations

Table 4.4: Effect of the number of trial solutions on the average and maximum gap and average CPU time (s)

Number of trial solutions	5,000	10,000	20,000	40,000	80,000
avg % gap	2.0	1.6	1.2	1.0	0.8
max % gap	8.8	8.7	6.4	6.0	5.6
avg time	0.3	0.5	1.2	2.0	4.4

Table 4.5: Overview of the input parameters TD-OP

Input parameter	Description	Value
τ_{init}	initial pheromone value	1
α	importance of pheromone information	4
β^*	importance of heuristic information ($\beta = \beta^* \cdot V_c $)	0.07
max_ants	number of solutions that are constructed and improved per iteration	75
ρ	evaporation rate	0.01
N_{ni}^{max}	maximum number of iterations without improvement (% of N_c)	25%

(N_c) that each construction and improvement cycle is performed. Given the goal of obtaining solutions within a few seconds, the value of this parameter has been severely restricted, and only 10,000 trial solutions were allowed. The exact value of N_c is therefore adjusted to the value of max_ants. For example when 20 ants are used, 500 iterations were allowed in order to generate 10,000 trial solutions. Varying the value of N_c gives an indication on the likelihood of the metaheuristic to get stuck in local optima (lack of diversity). If the algorithm easily gets stuck at a local optimum, increasing the number of trial solutions would not lead to significantly better solutions. The average results of five runs can be found in Table 4.4. From this table, it can be concluded that the results keep improving when the number of iterations is increased. Thus, the algorithm has an appropriate diversification strategy.

Table 4.5 provides an overview of the input parameters of ACS, including their value used for the experiments in Section 4.3.1 and 4.3.2. These values are based on preliminary experiments on a test set of 21 instances, constructed by randomly selecting 3 instances per dataset.

A second effect which needs to be analysed is the robustness of the algorithm when its parameters are changed. Therefore, the orthogonal combinations displayed in Table 4.6 for the ACS were tested on the same random sample set of 21 test instances after 10,000 trial solutions. To eliminate the effect of the random numbers generator, each configuration was executed 5 times and the average gap was used to benchmark the performance. Tables 4.7 shows the impact of deviating from the currently used value. The total %gap corresponds to the average gap over all tested combinations.

Table 4.6: Tested orthogonal combinations

Input parameter	Values
α	1, 3, 6, 9
β^*	0.01, 0.03, 0.05, 0.1
max_ants	10, 25, 50, 75, 100
ρ	0.01, 0.05, 0.1
N_{ni}^{max}	0.25, 0.5, 0.75, 1

Table 4.7: Effect of α , β , N_{ni}^{max} , ρ and max_ants on the average % gap

α	% gap	β^*	% gap	N_{ni}^{max}	% gap	ρ	% gap	max_ants	% gap
1	7.5	0.01	10.5	0.25	5.7	0.01	5.2	10	5.7
3	6.0	0.03	5.2	0.50	5.8	0.05	5.9	25	5.5
6	4.9	0.05	3.7	0.75	5.8	0.10	6.3	50	5.7
9	4.8	0.1	3.8	1.00	5.9			75	5.9
								100	6.2
Total	5.8	Total	5.8	Total	5.8	Total	5.8	Total	5.8

Note that preliminary tests indicated that the value of τ_{init} has no impact on the results. Furthermore, these tests showed that the optimal β parameter strongly depends on the size of the problem instance therefore the altered values of β represent a percentage, β^* , of $|V_c|$.

These results for the ACS indicate that variations in max_ants and N_{ni}^{max} have no significant impact on the performance (ANOVA test). In contrast, changing the values of ρ , α and β^* does have a significant effect on the performance of the ACS. As explained in Section 4.2.3 the absolute values of α and β^* determine the weight given to the random number in the end of the construction procedure. Higher values of α and β^* lead to less randomness in the choice of the next vertex to add to the solution. Based on Table 4.7, we conclude that higher values of both α and β^* are beneficial for the performance, i.e., that randomness in constructing a solution should be kept low. This effect is visually shown in Figure 4.4, where the dark blue area represents a large optimal zone corresponding to an average gap of 1%. This figure, made using the commercial software Origin Pro, represents a 3D surface plot for a wide range of orthogonal combinations of α and β^* . In general we can conclude that the ACS solution procedure is rather robust and small changes in the parameters do not lead to huge gaps in performance.

In addition to the effect on the average gap, the CPU time also remained the same when changing the values of the input parameters.

Table 4.8 yields insights into the performance of some design decisions. The effect is measured by leaving the component out of the algorithm and executing the altered algorithm on the set of 21 instances. For each alternative design of

Figure 4.4: 3D graph showing the impact of α and β^* on the average gap, the other parameters are set at their default value

Table 4.8: Effect of the design components on average and maximum gap

Metric	Normal	No 2-opt	No insert	No 2-opt & No insert
avg % gap	1.4	2.5	6.9	8.4
max % gap	6.1	12.3	30.9	30.3
avg CPU	0.5	0.5	0.3	0.2

the algorithm, the average and maximal gap as well as the computation time is indicated.

The backbone of the ACS is formed by the ant construction method together with the pheromone memory effect. It can be noted that the effect of 2-opt is small but significant, especially to reduce the maximum gap. Executing more 2-opt moves per iterations turns out to have a drastically lower marginal benefit. The added value of the 2-opt move lies in the extra diversity that is created by searching for an alternative sequence of the included vertices.

4.4 Solution method for the TD-OPTW

4.4.1 General outline

In this section, the procedures of the ACS for the TD-OP are altered and significantly improved to deal with the specifics of time window constraints. Moreover, a new and crucial time-dependent local search move called *replace* has been added. An overview of the solution method is presented in Algorithm 5 and in the remainder of this section, the different procedures are discussed in detail.

The ACS consists of 3 phases which are executed on a set of solutions called ants during a predetermined number of iterations. Firstly, ants, which represent solutions, are created in a construction phase. The second phase consists of an improvement phase where the ants are improved using three local search moves. Thirdly, in the global pheromone update phase, the pheromone values belonging to the arcs of the best ant of the iteration are increased.

Algorithm 5 Ant colony system for TD-OPTW - input parameters:
 $\alpha, \beta, \rho, \max_ants, \tau_{init}, N_{ni}^{max}$

```

 $sol_{ib} \leftarrow 0, sol_{gb} \leftarrow 0, N_{ni} \leftarrow 0, iteration \leftarrow 0$ 
while  $iteration < N_c$  do
  Initialize parameters:  $\tau, \eta(\tau_{init})$ 
  for  $i \leftarrow 1$  to  $\max\_ants$  do
    Construct solution & local pheromone update ( $\tau, \eta, \alpha, \beta, \rho$ )
    Calculate  $\max\_shift$ 
    Swap ( $\max\_shift$ )
    Insert ( $\max\_shift$ )
    Replace ( $\max\_shift$ )
  end for
   $sol_{ib} \leftarrow \arg \max(F(sol_1), F(sol_2), \dots, F(sol_{\max\_ants}))$ 
  if  $F(sol_{ib}) > F(sol_{gb})$  then
     $sol_{gb} \leftarrow sol_{ib}$ 
     $N_{ni} \leftarrow 0$ 
  else
     $N_{ni} \leftarrow N_{ni} + 1$ 
  end if
  Global pheromone update ( $\tau, sol_{ib}, N_{ni}, N_{ni}^{max}$ )
   $iteration \leftarrow iteration + 1$ 
end while

```

4.4.2 Pre-processing

In order to maintain acceptable computation times on larger instances, our solution approach needs to be scalable. Therefore, a set of neighbours is defined for each

vertex $i \in V_c$. This set, NB_i , is computed before the start of the ACS and consists of the closest vertices in the sense of spatial-temporal closeness. A vertex j is considered a neighbour of vertex i if the following equation holds:

$$o_i + s_i + ff_{i,j} \leq c_j \quad (4.8)$$

This means that it is possible to reach vertex j before the closing time when leaving at the earliest possible departing time at vertex i . Subsequently, these neighbours are first stored in a list, sorted according to the ratio of the reward over the sum of its service time and the free flow time between vertices i and j .

$$\frac{r_j}{s_j + ff_{i,j}} \quad (4.9)$$

If the number of neighbours in this list is greater than NB_{max} , only the first NB_{max} neighbours are stored as the set of neighbours for the vertex under consideration. These neighbour sets are used in the construction method and the local search moves. In the remainder of this text, a vertex j is called a neighbouring vertex of vertex i if j is included in the neighbour set of i . Various values were tested for NB_{max} and a value equal to 45 seemed a good trade-off between a scalable computational time and solution quality.

4.4.3 Initialize parameters

The value of the greedy information, $\eta_{i,j}$ for arc (i, j) is calculated as follows:

$$\eta_{i,j} = \frac{r_j}{ff_{i,j}} \quad (4.10)$$

The reasoning behind this formula is as follows: vertices with higher rewards and lower time-independent travel time from the last visited vertex are more likely to be interesting vertices to visit next. This way of working turns out to be less computationally expensive in comparison to using the time-dependent travel time as part of the ratio.

The pheromone value ($\tau_{i,j}$) of all arcs is initially set at a value τ_{init} .

$$\tau_{i,j} = \tau_{init} \quad (4.11)$$

The actual value of τ_{init} is of no importance as it is the relative difference between the τ -values that determines the likelihood to be selected in the construction method.

4.4.4 Construct solution & local pheromone update

This method creates `max_ants` solutions independently and sequentially. Each construction starts from an empty solution and adds vertices at the end of the solution until no more vertices can be inserted due to the travel time restriction. At

that point, the end vertex is added to finalize the solution, and the algorithm moves on to the next solution, until `max_ants` solutions are created.

Before adding a vertex following the already included vertex u , a list called V_u of all neighbouring vertices eligible to include is created. This is done by first calculating the time-dependent travel time between the last included vertex u and a vertex under consideration i . Next, vertex i is not added to the list when the arrival time is greater than its closing time or when there is not enough time to reach the end vertex:

$$t_{(u,i)}^a > c_i \quad (4.12)$$

$$t_{(i,v)}^a > t^{max} \quad (4.13)$$

Afterwards, each vertex in the list receives a probability l_i to be added to the solution. This probability is at the beginning of the procedure initialized to 1 for every feasible vertex. It is also possible to arrive before o_i which results in a waiting time equal to:

$$o_i - t_{(u,i)}^a \quad (4.14)$$

Because waiting is not beneficial to obtain good solutions, l_i is decreased for the vertices where waiting is necessary:

$$\text{if } t_{(u,i)}^a < o_i \text{ then } l_i = 1 - \frac{o_i - t_{(u,i)}^a}{t^{max}} \quad (4.15)$$

Finally, when the list is ready, the final probability is updated as follows:

$$l_i = l_i * \frac{\tau_{u,i}^\alpha \cdot \eta_{u,i}^\beta}{\sum_{w \in V_u} \tau_{u,w}^\alpha \cdot \eta_{u,w}^\beta} \quad \forall i \in V_u \quad (4.16)$$

In this equation, α determines how much weight is given to the pheromone value and β defines how much weight is given to the greedy information. Then, a random number is generated to determine, together with the probability l_i , which vertex from the list is added to the solution by using a roulette wheel selection method. Subsequently, the τ -value related to the newly added arc in the solution is decreased. This local pheromone procedure which enhances diversification is executed as follows:

$$\tau_{u,i} = \tau_{u,i} \cdot (1 - \rho) \quad (4.17)$$

Recall that ρ is the evaporation rate.

After the construction of `max_ants` solutions, the local search moves explained below and displayed in Figure 4.5 are executed on each constructed solution. For these examples a service time equal to zero is assumed.

4.4.5 Calculation of the local evaluation metric

As in the ACS for the TD-OP, a local evaluation metric is used to prevent a global and time-consuming evaluation of a solution after every insertion and replacement attempt (Section 4.4.7 and 4.4.8). In short, for every vertex in the current solution, the maximum amount of time that a visit to it can be postponed before the solution becomes infeasible is stored. The calculation of this metric called `max_shift`, is now adapted to take into account the time window restrictions and is presented in the pseudo code of Algorithm 6. Note that in this procedure, sz is the number of vertices stored in solution sol . The difference with the previous implementation is situated in lines 8 and 9, where it is checked if the departure time (after subtracting the service time), which can only be shifted towards the end of the solution sequence, is greater than the closing time of the departing vertex. If this is the case, the departure time is set equal to the closing time. The `max_shift` value is calculated for every vertex of the solution except the start vertex.

Algorithm 6 Calculation of `max_shift` for TD-OPTW - input: sol

```

 $t_a \leftarrow t_0 + t^{max}$ 
for  $i = 0$  to  $sz - 2$  do
     $y \leftarrow sol_{sz-(i+1)}$ 
     $z \leftarrow sol_{sz-i}$ 
     $k \leftarrow \text{get\_time\_slot\_index}(t_a)$ 
     $t_d \leftarrow \text{calculate\_departure\_time}(t_a)$ 
     $t_d \leftarrow t_d - s_y$ 
    if  $t_d > c_y$  then
         $t_d = c_y$ 
    end if
     $\text{max\_shift}_{sz-(i+1)} \leftarrow t_d - \text{actual\_departure\_time}_{sz-(i+1)}$ 
     $t_a \leftarrow t_d$ 
end for

```

4.4.6 Swap local search move

The swap local search move tries to exchange two vertices in the solution sequence in order to save travel time. We define y and z as the vertices that will be exchanged, k and l as the predecessor and successor of y and m and n as the predecessor and successor of z . After verification that k and l are neighbours of z and that m and n are neighbours of y , the vertices y and z are temporary exchanged. Otherwise, this swap pair (y, z) is discarded before further evaluation. Afterwards, the new arrival time for the vertices between k and n is calculated while accounting for time window restrictions. Next, the difference in travel time (calculated from k to n) of the possible swap is calculated and if this difference (local gain in

Figure 4.5: Overview of the time-dependent local search moves

travel time) is negative, the swap is executed. However, it is unsure that this exact difference in travel time can be kept until the end of the solution sequence due to waiting times and the time-dependent nature of succeeding travel times after vertex n (global gain in travel time). However, since our travel time is FIFO we are sure that departing earlier at vertex n guarantees that our new arrival time at the end vertex will be less than or equal to the previous one. Therefore this approach is chosen to avoid a complete evaluation of the solution. This significantly speeds up the search for feasible swap partners. Afterwards, the new arrival times are calculated and the `max_shift` values for the vertices starting from n until the end of the solution can be updated. An update of the `max_shift` variable is different from a recalculation and is executed according to the following formula:

$$\text{max_shift}_i = \text{max_shift}_i + (\text{previous_departure_time}_i - \text{new_departure_time}_i) \quad (4.18)$$

Finally, the `max_shift` variable needs to be recalculated for all vertices before y .

Since this local search move does not use the `max_shift` local evaluation metric, a lot of travel time calculations are needed and therefore the procedure is executed in a first improvement manner. The procedure stops when no more improvements can be found.

4.4.7 Insert local search move

The problem-specific *insert* local search move iteratively attempts to insert non-included neighbouring vertices into an existing solution, thus improving its total reward. The insert move tries to insert vertices into the current solution if the extra travel time, required to visit this new vertex, is less than the value of `max_shift` of the succeeding vertex. For example, when the algorithm attempts to insert vertex y between x and z , the extra time-dependent travel time equals:

$$\Delta \text{travel_time} = \text{travel_time}_{x,y} + \text{travel_time}_{y,z} - \text{travel_time}_{x,z} \quad (4.19)$$

Note that the procedure first checks if vertex y is a neighbour of x and z , otherwise this possible insertion combination is discarded. During the calculation of these components, the required service time is added to the travel time. Waiting time is also added when:

$$t_{a_{(x,y)}} < o_y \quad \text{or} \quad t_{a_{(y,z)}} < o_z \quad (4.20)$$

The insert move is cancelled when during the calculation of the arrival time the following time window is violated:

$$t_{a_{(x,y)}} > c_y \quad (4.21)$$

Recall that this violation is still possible as the set of neighbours is constructed based on the time-independent travel time (Section 4.4.2). Vertex y can only be inserted into the current solution when the extra travel time is less than or equal to the maximum amount of time a visit to vertex z can be shifted towards the end of the solution sequence or:

$$\Delta \text{travel_time} \leq \text{max_shift}_z \quad (4.22)$$

When vertex y is actually included in the solution, the algorithm updates the travel time from x to y , as well as the travel times between the vertices succeeding vertex y . This is necessary, because the insertion of vertex y has most likely caused a change in waiting and/or travel time for the arcs of the solution succeeding vertex y . The `max_shift` of the vertices succeeding vertex y can easily be adjusted based on the previous `max_shift` value, the previous travel time and the new travel time of the vertex under consideration:

$$\text{max_shift}_i = \text{max_shift}_i + (\text{previous_departure_time}_i - \text{new_departure_time}_i). \quad (4.23)$$

Secondly, for the vertices preceding vertex z only a recalculation of `max_shift` is required.

As a result of this procedure, the computation time needed to check if a vertex can be included or not, and actually executing a feasible insertion is drastically

reduced. A visual overview on which travel times are calculated and which are stored in memory is provided in Figure 4.5.

The insert move is executed in a best-improving manner and stops when no more feasible improvements can be found. The best improving combination is found using a ratio equal to the increase in reward divided by the increase in travel time of a feasible insertion. If the increase in travel time is less than or equal to zero, the ratio is set equal to:

$$(1 - \Delta \text{travel_time}) * r_y. \quad (4.24)$$

4.4.8 Replace local search move

After the insert local search move, the replace move is executed. This move tries to replace a vertex from the current solution with a non-included vertex with a higher reward at the same position in the solution sequence. We define y as the potential replacement vertex to include, z as the vertex in the current solution that will be replaced and x and w respectively as the predecessor and successor of z . The reward of z is compared with the reward of its neighbours, when the reward of a non-included neighbour (y) is higher, it is checked if the potential replacement is feasible. A replacement is possible when y is a neighbour of both x and w and when the following inequality is valid:

$$\Delta \text{travel_time} \leq \text{max_shift}_w \quad (4.25)$$

$$\Delta \text{travel_time} = (\text{travel_time}_{x,y} + \text{travel_time}_{y,w}) - (\text{travel_time}_{x,z} + \text{travel_time}_{z,w}) \quad (4.26)$$

During the calculation of these components, a waiting time is added to the travel time when:

$$t_{a_{(x,y)}} < o_y \quad \text{or} \quad t_{a_{(y,w)}} < o_w \quad (4.27)$$

The replace move is cancelled when the following equation is valid:

$$t_{a_{(x,y)}} > c_y \quad (4.28)$$

As soon as a feasible replacement option is found, it is executed (first improvement manner). Afterwards, the `travel_time` is recalculated for vertices after y while the `max_shift` variable is recalculated for vertices before y and updated for vertices after y . The replace move is repeated for each included vertex until no more feasible improvements can be found.

4.4.9 Update global best

After the construction and local search moves, the algorithm checks which solution has the highest solution reward and stores it as the iteration best solution (sol_{ib}).

If its reward is better than the best reward found during previous iterations (sol_{gb}), we update sol_{gb} . If not, variable N_{ni} , that keeps track of the number of iterations without improvement, is incremented. When sol_{gb} is updated, N_{ni} is reset to 0.

4.4.10 Global Pheromone update

If N_{ni} is less than a maximum threshold (N_{ni}^{max}), the pheromone values of the arcs, used in sol_{ib} , are augmented with the value of τ_{init} . This makes it more likely that these arcs will be used in a subsequent construction procedure (intensification):

$$\tau_{i-1,i} \leftarrow \tau_{i-1,i} + \tau_{init} \quad \forall i \in sol_{ib} | i > 1 \quad (4.29)$$

If N_{ni} equals N_{ni}^{max} , all the pheromone values are reset to τ_{init} . This means that all arcs have again an equal probability to be chosen during the next construction procedure, allowing diversification.

4.5 Results for the TD-OPTW

In this section the pre-processing procedure as well as the performance of the proposed algorithm for TD-OPTW will be discussed using various tests. In Subsection 4.5.1 the results of the ACS are compared against known optimal solutions. Since only a few instances could be solved to optimality, a set of artificial benchmark instances are designed in Subsection 4.5.2. In Subsection 4.5.3 the ACS is compared against a state-of-the-art solution method for the TD-OPTW. This section concludes with an overview on the sensitivity in Subsection 4.5.4 and the (negative) impact of not accounting for time-dependent travel times is shown in Subsection 4.5.5.

4.5.1 Comparison to optimal solutions of small instances

The first experiment consists of a straightforward comparison of the results of the ACS and the optimal solution found by solving the MIP for TD-OPTW from Section 3.2.1. However, most of the time-dependent instances discussed in Section 3.2.2 are too complex to be solved with the MIP using a commercial solver. Since this is not the focus of our research, only a basic implementation of this MIP was implemented using the commercial solver, CPLEX 12.6 (64-bit) using a *high performance computing* (HPC) infrastructure which consists of 48 Intel Xeon X5675 3.07 GHz processors and a total of 384 GB of memory. The comparison between the results of these optimal solutions and the ACS for five independent runs is displayed in Table 4.9. The labels S, M, L display the span (closing time minus opening time) of the time windows (small, medium, large)

Table 4.9: Comparison to optimal solutions of small instances

name	$ V_c $	t^{max} h	TW S,M,L	CPLEX		ACS			
				optimal reward	cpu s	worst %gap	avg %gap	best %gap	cpu s
20.1.1	20	8	S	159	500	0.0	0.0	0.0	0.1
20.1.2	20	8	M	173	343	0.0	0.0	0.0	0.1
20.1.3	20	8	L	183	761	0.0	0.0	0.0	0.1
20.2.1	20	10	S	188	4,308	0.0	0.0	0.0	0.1
20.2.2	20	10	M	201	5,394	0.0	0.0	0.0	0.1
20.2.3	20	10	L	195	2,545	0.0	0.0	0.0	0.1
20.3.1	20	12	S	277	130,118	0.0	0.0	0.0	0.2
20.3.2	20	12	M	245	9,692	0.0	0.0	0.0	0.1
					max	0.0	0.0	0.0	0.2
					avg	0.0	0.0	0.0	0.1
					% optimal	100.0	100.0	100.0	

As performance metrics, the CPU is used together with the percentage gap between the total reward of the optimal solution and the total reward of the heuristic solution:

$$\% \text{ gap} = \frac{\text{optimal reward} - \text{heuristic reward}}{\text{optimal reward}} * 100$$

(4.30)

The average, the best and the worst gap over all instances is recorded. The ACS is executed until 10,000 trial solutions have been created. The exact value of N_c is therefore adjusted to the value of `max_ants`. For example when 20 ants are used, 500 iterations were allowed in order to generate 10,000 trial solutions. The results show the validity of our MIP model but also the excessive time needed to solve these small instances using an commercial solver and a straightforward MIP formulation. Apart from the number of vertices in the problem instance, the required computation time depends in the first place on the value of t^{max} and in the second place on the time window span. Furthermore, this table illustrates optimal results obtained for all instances and in each run the very short computation times of the ACS on small problem instances.

The reward of the ACS for the instances without known optimal solution are also displayed in Table 4.10 to allow for future benchmarking.

4.5.2 Comparison to known optimal solutions of large instances

To evaluate the performance of the TD-OPTW metaheuristic developed, it would be useful to have an optimal solution for each benchmark instance. However, the time-dependent instances are too complex to be solved with a commercial solver and a straightforward MIP formulation. Therefore, a procedure is developed which

Table 4.10: Detailed results TD-OPTW

name	$ V_c $	t^{max} h	TW S,M,L	ACS			cpu s
				worst avg reward	avg avg reward	best avg reward	
20.3.3	20	12	L	259	259	259	0.1
20.4.1	20	14	S	274	274	274	0.1
20.4.2	20	14	M	275	275	275	0.2
20.4.3	20	14	L	268	268	268	0.2
50.1.1	50	8	S	288	288	288	0.3
50.1.2	50	8	M	274	274	274	0.3
50.1.3	50	8	L	289	289	289	0.3
50.2.1	50	10	S	298	298	298	0.3
50.2.2	50	10	M	310	310	310	0.4
50.2.3	50	10	L	340	340	340	0.4
50.3.1	50	12	S	339	339	339	0.3
50.3.2	50	12	M	404	404	404	0.5
50.3.3	50	12	L	366	366	366	0.6
50.4.1	50	14	S	471	476.6	478	0.5
50.4.2	50	14	M	435	439.8	441	0.7
50.4.3	50	14	L	450	450	450	0.7
100.1.1	100	8	S	275	275	275	0.5
100.1.2	100	8	M	278	278	278	0.5
100.1.3	100	8	L	343	343	343	0.6
100.2.1	100	10	S	351	351.2	352	0.7
100.2.2	100	10	M	366	366.6	367	0.6
100.2.3	100	10	L	370	370	370	0.7
100.3.1	100	12	S	435	436	437	0.8
100.3.2	100	12	M	444	446.6	454	0.9
100.3.3	100	12	L	466	467	468	1.0
100.4.1	100	14	S	478	480	484	1.0
100.4.2	100	14	M	491	494.6	497	1.0
100.4.3	100	14	L	519	526.8	538	1.1

allows to test the proposed algorithm on larger instances with known optimal solutions.

Firstly, all instances were solved as time-independent OPTWs using the HPC infrastructure requesting 48 Intel Xeon X5675 3.07 GHz processors and a total of 384 GB of memory. The mixed integer problem formulation of the OPTW from Section 2.1 was used to find these optimal solutions. The obtained solutions together with their corresponding computation times are available on: <http://www.mech.kuleuven.be/en/cib/op/>.

The excessive resources and required CPU time needed to solve these time-independent OP instances, illustrates that it would be pointless to solve the datasets as time-dependent OPTWs. The found (near-)optimal solutions are displayed in Table 4.11 (column “optimal”). If the optimal solution could not be found, the best solution after 72 hours of computation time is taken as benchmark solution. These instances are marked with a star in the column “optimal”. Following the optimization by CPLEX, the optimal time-independent solutions (sequence of vertices) are used to modify the original time-dependent instances in such a way that slightly modified time-dependent instances with known (near-)optimal solutions are created using the procedure described in Section 4.3.2.

The creation of the benchmark instances in this way allows to compare the performance of the developed solution methods with known optimal solutions for larger instances. For five independent runs of the algorithm, these results are displayed in Table 4.11. In Table 4.12 the % gap per dataset is displayed.

These results prove the high performance quality of the ACS, since the average gap is very low, i.e. only 0.2%. Furthermore, the known optimal solution could be found for 32 out of 36 test instances. A second conclusion is that the average gap increases as the test instances become more complex due to a longer travel time limit and an increasing number of vertices. Studying the computation time leads to the conclusion that the ACS is very fast, as on average only 0.4 seconds are needed to obtain a solution. The maximum observed CPU time was 1.1 seconds, which is more than fast enough for most application purposes. Therefore, it can be concluded that the ACS is able to deliver a high performance requiring a minimal computational effort.

4.5.3 Comparison to a state-of-the-art solution procedure

The iterated local search method (ILS) of Garcia et al. [53] was developed for the TD-TOPTW and is, apart from the complex algorithm of Abbaspour and Samadzadegan [1], the only practical algorithm that is capable to deal with time-dependency. Since this ILS method exploits the fixed frequency of buses to perform quick arrival time and departure time updates, it is not capable to deal with time-dependency originating from congestion on the arcs. Therefore, the ILS was reimplemented

Table 4.11: Results for the TD-OPTW per problem instance

name	$ V_c $	t^{max} h	TW S,M,L	optimal reward	worst %	avg %	best %	CPU(s) s
20.1.1	20	8	S	177	0.0	0.0	0.0	0.1
20.1.2	20	8	M	193	0.0	0.0	0.0	0.1
20.1.3	20	8	L	201	0.0	0.0	0.0	0.1
20.2.1	20	10	S	213	0.0	0.0	0.0	0.1
20.2.2	20	10	M	219	0.0	0.0	0.0	0.1
20.2.3	20	10	L	211	0.0	0.0	0.0	0.1
20.3.1	20	12	S	306	0.0	0.0	0.0	0.2
20.3.2	20	12	M	262	0.0	0.0	0.0	0.1
20.3.3	20	12	L	286	0.0	0.0	0.0	0.1
20.4.1	20	14	S	293	0.0	0.0	0.0	0.2
20.4.2	20	14	M	299	0.0	0.0	0.0	0.2
20.4.3	20	14	L	283	0.0	0.0	0.0	0.2
50.1.1	50	8	S	314	0.0	0.0	0.0	0.3
50.1.2	50	8	M	290	0.0	0.0	0.0	0.3
50.1.3	50	8	L	316	0.0	0.0	0.0	0.3
50.2.1	50	10	S	322	0.0	0.0	0.0	0.3
50.2.2	50	10	M	347	0.0	0.0	0.0	0.4
50.2.3	50	10	L	346	0.0	0.0	0.0	0.4
50.3.1	50	12	S	380	0.0	0.0	0.0	0.3
50.3.2	50	12	M	444	0.0	0.0	0.0	0.5
50.3.3*	50	12	L	403	0.0	0.0	0.0	0.6
50.4.1	50	14	S	498	0.0	0.0	0.0	0.5
50.4.2*	50	14	M	463	0.0	0.0	0.0	0.7
50.4.3*	50	14	L	479	0.0	0.0	0.0	0.7
100.1.1	100	8	S	297	0.0	0.0	0.0	0.4
100.1.2	100	8	M	320	0.0	0.0	0.0	0.4
100.1.3	100	8	L	373	0.0	0.0	0.0	0.6
100.2.1	100	10	S	397	0.0	0.0	0.0	0.7
100.2.2*	100	10	M	393	0.0	0.0	0.0	0.7
100.2.3	100	10	L	394	0.0	0.0	0.0	0.7
100.3.1	100	12	S	490	0.0	0.0	0.0	0.9
100.3.2*	100	12	M	511	0.0	0.0	0.0	0.8
100.3.3	100	12	L	525	1.5	0.3	0.0	1.0
100.4.1*	100	14	S	505	4.2	3.1	1.0	1.0
100.4.2*	100	14	M	543	3.1	2.5	1.7	1.1
100.4.3*	100	14	L	590	3.4	1.4	0.0	1.1
max					4.2	3.1	1.7	1.1
avg					0.3	0.2	0.1	0.4
% optimal					88.9	88.9	94.4	

Table 4.12: % gap (best, average, worst) and average CPU time per dataset

$ V_c $	best % gap	avg % gap	worst % gap	CPU s
20	0.0	0.0	0.0	0.1
50	0.0	0.0	0.0	0.4
100	1.0	0.6	0.2	0.8

Table 4.13: Comparison to a state-of-the-art technique after 10,000 solutions

$ V_c $	ACS cpu s	ILS cpu s	gap %
20	0.1	0.1	-3.8%
50	0.4	0.6	-8.7%
100	0.8	2.4	-6.0%
Overall	0.4	1.1	-6.2%

with a few modifications. Firstly, the local evaluation metric and solution update procedure described in Section 4.4.7 was used. Secondly, after the shake procedure, the solution is completely evaluated since it turned out to be too time expensive to verify which parts of the solution sequence needed to be updated and to execute the partial updates afterwards. These issues, together with implementation details and not making use of parallel computing given the sequential structure of ILS, results in greater computational times for the ILS method compared to the ACS. It should be noted that these issues contributed to the choice of the ACS as the preferred solution framework in the first place. The results of ILS and ACS, both with 10,000 trial solutions, are compared and can be found in Table 4.13. More detailed results can be found on <http://www.mech.kuleuven.be/en/cib/op/>. The first two columns display the average computational time of both solution procedures and the third column displays the gap defined as:

$$gap = \frac{ILSreward - ACSreward}{ILSreward}$$

(4.31)

On average the performance of the ACS is 6.2% better than that of the ILS. Furthermore, the detailed results show that the ILS could not improve the reward of the ACS solution method on a single instance. Finally, we also compared the performance of the ILS after 50,000 solutions with the results of the ACS after 10,000 trial solutions and noticed the same -6.2% performance gap which might indicate that the ILS more easily gets stuck in local optima. Secondly, iteratively removing vertices from a solution turns out to be impractical and time consuming.

Table 4.14: Overview of the input parameters TD-OPTW

Input parameter	Description	Value
τ_{init}	initial pheromone value	1
α	importance of pheromone information	4
β	importance of heuristic information	1
max_ants	number of solutions that is constructed and improved per iteration	50
ρ	evaporation rate	0.05
N_{ni}^{max}	maximum number of iterations without improvement (% of N_c)	25%

Table 4.15: Effect of α , β , N_{ni}^{max} , ρ and max_ants on the average % gap

α	% gap	β	% gap	N_{ni}^{max}	% gap	ρ	% gap	max_ants	% gap
1	2.9	1	0.7	0.25	1.5	0.01	2.6	10	2.8
3	1.1	3	0.8	0.50	1.6	0.05	1.2	25	1.8
6	1.1	6	1.6	0.75	1.7	0.10	1.2	50	1.2
9	1.5	9	3.4	1.00	1.8			75	1.2
								100	1.2
Total	1.7%	Total	1.7	Total	1.7	Total	1.7	Total	1.7

4.5.4 Sensitivity analysis

Table 4.14 provides an overview of the input parameters of ACS, including their value used during all the discussed experiments. These values were determined after preliminary tests. A sensitivity study, similar to the one of the ACS for the TD-OP as discussed in Section 4.3.3, is performed to measure the impact on the performance for various combinations of input parameter values. The results are displayed in Table 4.15. The significant input parameters that have an impact on the performance, based on an ANOVA test, are α , β , ρ and max_ants.

Table 4.16 provides insights into the performance of some design decisions. These effects are measured by leaving out some or all of the 3 local search moves (S: swap, I: insert, R: replace) and repeating the test procedure discussed in Section 4.5.2. The missing components in the ACS framework (Algorithm 5) are marked with a "-". Mainly based on the average gap, we conclude that the strength of the ACS lies in the interaction effect between the swap, insert and replace local search moves.

4.5.5 Impact of time-dependent travel times

In order to measure the impact of ignoring the time-dependency of travel times when dealing with practical problems, the optimal OPTW solutions are evaluated in a time-dependent context. This means that all travel times, arrival times and departure times are updated. Two events can turn a solution infeasible: a violation of the time window at a regular vertex and secondly, arriving too late at the end

Table 4.16: Effect on the % gap and average CPU time (s) of different design components

Metric	S I R	- - -	S - -	- I -	- - R	- I R	S I -	S - R
max % gap	3.1	30.6	24.5	12.7	26.7	9.5	7.6	24.3
avg % gap	0.2	10.6	8.8	2.4	9.7	0.8	0.8	0.8
avg cpu	0.4	0.1	0.2	0.4	0.2	0.4	0.4	0.2

vertex.

Two different repair algorithms are proposed that inspect the solution sequence (starting from the start vertex to the end vertex) and implement different recourse actions. The first algorithm is called *heuristic repair* and removes the first vertex with a violated time window and inspects the solution again for further violations. When a late arrival occurs at the end vertex, the vertex with the lowest reward in the solution sequence is removed and the solution is again inspected for further violations. Vertices are only set eligible for removal if no waiting occurs at subsequent vertices in the solution sequence. The reason for this is that removing a vertex which is positioned before a vertex where waiting occurs, would not lead to a decrease of the total travel time as it would only lead to an increase in waiting time at a subsequent vertex. The second algorithm is called *business logic repair* and also removes the first vertex with a violated time window and afterwards inspects the solution again for further violations. For a late arrival at the end vertex, however, the last regular vertex of the solution sequence is removed and the solution is inspected for further violations. This last recourse action is chosen because it might resemble the action of actual planners in logistics companies. The last customers on a trip are more likely to be removed when it becomes clear that the driver is falling behind schedule and wants to return to the end depot. Finally, OPTW solutions, evaluated by both algorithms, are compared against the TD-OPTW solutions generated by the ACS.

The results are displayed in Tables 4.17 and 4.18. The first four columns list respectively the name, number of vertices, maximum allowed total travel time, time window restrictiveness of the problem instance and the total reward of the optimal OPTW instance. The next two columns display respectively the reward of the evaluated optimal OPTW solution by the *heuristic repair* (HR) and the *business logic repair* (BL). In the 8th column the reward of the time-dependent OPTW solution produced by the ACS is given. Finally, the last four columns display the % improvement (imphr and impbl) of the ACS versus the two different evaluated OPTW solutions and also the diversity (divhr and divbl) between the two solution sequences for each evaluation method. The diversity between two solutions A and B is calculated as the sum of the number of vertices in A not present in B and the number of vertices in B not present in A, divided by the total number of

vertices present in A and B (start and end vertex not included). The results are road network specific but nonetheless stress the severe impact of congestion on the quality and structure of the proposed solutions. Apparently there seems to be no strong connection between the degree of improvement and diversity, on the one hand, and the most important instance parameters like the number of vertices, t^{max} and the time window restrictiveness on the other hand. However, instances with smaller time windows are more likely to become infeasible and therefore the improvement of using the ACS is more important. A possible explanation for the greater impact of congestion for the instances with 100 vertices is that the routes for these problems are more complex and are therefore also more vulnerable to congestion.

An overall conclusion concerning the diversity of the routes proposed for both problems is that a "good" route for the TD-OPTW differs significantly from the optimal route for the OPTW. This strengthens the claim that time-independent route planning is not appropriate in a time-dependent business context. These results also show that using a time-independent routing planner together with some basic recourse actions, that are often used in practice in order to deal with time-dependent disturbance, does not work out well. Time-independent routes cannot be easily altered or transformed into robust routes that have some "buffer" against the negative effects of time-dependent disturbance since congestion is a time-spatial phenomenon.

Table 4.17: Impact of time-dependency per problem instance

name	$ V_c $	t^{max}	TW	OPTW	HR	BL	ACS	imphr	impbl	divhr	divbl
		h	S,M,L	r	r	r	r	%	%	%	%
20.1.1	20	8	S	177	151	118	159	5.3	34.7	33.3	33.3
20.1.2	20	8	M	193	166	166	173	4.2	4.2	16.7	16.7
20.1.3	20	8	L	201	163	137	184	12.9	34.3	33.3	50.0
20.2.1	20	10	S	213	172	152	188	9.3	23.7	38.5	53.8
20.2.2	20	10	M	219	168	181	201	19.6	11.0	33.3	42.9
20.2.3	20	10	L	211	153	148	195	27.5	31.8	50.0	38.5
20.3.1	20	12	S	306	246	236	277	12.6	17.4	15.8	15.8
20.3.2	20	12	M	262	228	221	246	7.9	11.3	17.6	17.6
20.3.3	20	12	L	286	225	231	259	15.1	12.1	33.3	36.8
20.4.1	20	14	S	293	221	223	274	24.0	22.9	47.4	23.8
20.4.2	20	14	M	299	241	220	275	14.1	25.0	26.3	10.0
20.4.3	20	14	L	283	241	224	268	11.2	19.6	40.0	33.3
50.1.1	50	8	S	314	236	259	288	22.0	11.2	37.5	15.8
50.1.2	50	8	M	290	261	261	274	5.0	5.0	22.2	22.2
50.1.3	50	8	L	316	287	287	289	0.7	0.7	26.3	26.3
50.2.1	50	10	S	322	252	274	298	18.3	8.8	11.1	15.8
50.2.2	50	10	M	347	280	280	310	10.7	10.7	55.6	55.6
50.2.3	50	10	L	346	300	317	340	13.3	7.3	9.1	13.0
50.3.1	50	12	S	380	328	328	339	3.4	3.4	71.4	71.4
50.3.2	50	12	M	444	383	380	404	5.5	6.3	13.0	21.7
50.3.3	50	12	L	403	301	281	366	21.6	30.2	52.4	52.4
50.4.1	50	14	S	498	423	430	478	13.0	11.2	40.7	42.9
50.4.2	50	14	M	463	406	389	441	8.6	13.4	23.1	23.1
50.4.3	50	14	L	479	413	413	450	9.0	9.0	40.7	40.7
100.1.1	100	8	S	297	241	229	275	14.1	20.1	66.7	77.8
100.1.2	100	8	M	320	223	210	276	23.8	31.4	50.0	50.0
100.1.3	100	8	L	373	331	323	343	3.6	6.2	20.0	20.0
100.2.1	100	10	S	397	334	332	351	5.1	5.7	73.9	68.0
100.2.2	100	10	M	393	290	289	367	26.6	27.0	71.4	81.8
100.2.3	100	10	L	394	338	338	370	9.5	9.5	44.0	44.0
100.3.1	100	12	S	490	421	415	437	3.8	5.3	20.0	26.7
100.3.2	100	12	M	511	426	441	447	4.9	1.4	25.9	3.4
100.3.3	100	12	L	525	458	447	470	2.6	5.1	9.1	15.2
100.4.1	100	14	S	491	415	395	483	16.4	22.3	53.3	53.3
100.4.2	100	14	M	534	476	464	491	3.2	5.8	9.1	15.2
100.4.3	100	14	L	590	510	489	528	3.5	8.0	48.6	48.6
avg								11.4	14.3	35.6	35.5

Table 4.18: Impact of time-dependency (heuristic repair) versus problem characteristics

$ V_c $	%imphr	%divhr	t^{max}	%imphr	%divhr	TW	%imphr	%divhr
20	14.5	32.1	8	12.7	34.0	S	16.3	42.5
50	12.8	33.6	10	17.2	43.0	M	11.7	30.4
100	11.5	41.0	12	9.3	28.7	L	10.9	33.9
			14	12.6	36.6			

4.6 Motivation for the local search moves

As far as the choice of the local search moves is concerned, the survey paper of the OP ([145]) states that insert, replace, 2-opt and swap are the most commonly used local search moves but also mentions twoinsert, tworeplace and change as less frequently used local search moves. Recall that the proposed solution method for the TD-OP incorporates a time-dependent version of the insert and a modified 2-opt local search move while the solution method for the TD-OPTW uses a time-dependent insert, replace and swap local search move.

During initial tests, the change local search move was tried but appeared not efficient due to a large removal of vertices which in a time-dependent setting can increase the total travel time and is computationally inefficient. Secondly, the twoinsert local search move was tried but appeared difficult to implement in a time-dependent context as the proposed local evaluation metric no longer could be used. Thirdly, looking at Table 4.16 the replace move is only marginal effective therefore we did not implement the tworeplace local search move. However, we suspect the same issues as the ones mentioned for the twoinsert local search move will be present.

The modified 2-opt developed for the TD-OP first checks if there is a time-independent gain. Afterwards it validates this biggest gain in a time-dependent context. In the solution method for the TD-OPTW the modified 2-opt is replaced by a swap move that first locally checks if there is a time-dependent gain in travel time in between the two swapped vertices and later globally validates this gain in travel time. A 2-opt variant for the TD-OPTW, where the gain in time-dependent travel time is also first checked locally and afterwards validated globally, was also tested and similar results except for a slightly greater maximum gap (avg gap 0.2%, max gap 4% avg cpu 0.4) were obtained. Overall, there can be concluded that evaluating changes to a solution first locally followed by a global validation provides better performance than preselecting interesting moves by the gain in time-independent travel time.

4.7 Conclusion

In this chapter, the time-dependent orienteering problem (TD-OP) and the time-dependent orienteering problem with time windows (TD-OPTW) are presented. These problems are extensions of the orienteering problem in which the travel time between two vertices depends on the departure time at the first vertex. Apart from a practical mathematical formulation to these problems that obtains optimal solutions for small problem instances, the main contribution is a fast local search based metaheuristic for each problem, inspired by an ant colony system.

For the TD-OP, the local search mechanism itself is a time-dependent insert

move, speeded up by a local evaluation metric. The algorithm obtains high-quality results on these instances requiring very small computation times, even for instances with around 100 vertices. An average run obtains solutions with a reward gap of only 1.4% using 0.5 seconds of computation time. For 44% of the test instances, the known optimal solution is found. A sensitivity analysis demonstrates that the performance of the algorithm is not sensitive to small changes in the parameter settings. This indicates that robust behaviour might therefore be expected when incorporating this algorithm into real-life applications. The fast execution time of this algorithm enables some interesting business applications where it is necessary to update routes when new traffic information becomes available and to provide proper guidance to drivers/tourists on the road.

For the TD-OPTW, the local search moves themselves represent respectively a time-dependent insert, replace and swap move. The evaluation of possible improvement moves is fastened by a local evaluation metric and by efficiently limiting the number of vertices that are considered by the procedure.

The proposed algorithm obtains very good results on these benchmark instances, requiring small computation times due to the combination of a well performing metaheuristic framework and the interaction effect between three efficient local search moves tailored to the problem characteristics. The average reward gap with the known optimal solution on these test instances is only 0.2% with an average computation time of 0.4 seconds. A comparison between the ACS and a state-of-the-art technique showed that the performance of the latter is 6.2% worse. The average impact of not accounting for time-dependency on the proposed network is at least 11.4%. Finally, there is a significant difference (35.6%) between the structure of the solution sequence of a time-dependent solution and a solution ignoring time-dependency. These results also show that some basic recourse actions which are often used in practice to create a kind of "buffer" for the time-dependent disturbance, often do not lead to high quality solutions.

5

Solution method and results for the orienteering problem with time-dependent and stochastic travel times

In this chapter, the proposed solution method for the time-dependent orienteering problem with stochastic weights and time windows (TD-OPSWTW) will be explained together with the experimental results on benchmark instances.

In the TD-OPSWTW, the travel times are stochastic, therefore the arrival and departure times become stochastic variables too. This means that all actions that are performed at a vertex occur with a certain probability: waiting because of arriving too early, servicing in the case of on-time arrival and not servicing the vertex because of a late arrival. A reward can only be collected with a certain probability, while arriving too late a vertex should be penalised with a certain probability. This leads to the fact that the objective function of this problem is represented by an expected reward function.

5.1 Solution method for the TD-OPSWTW

The altered ant colony optimization algorithm takes into account the expected reward and the stochastic travel times using an estimation algorithm while it sequentially adds vertices to the solution. Before the metaheuristic is discussed in more

detail, the Monte Carlo sampling and the estimation algorithm that provide us with a way of calculating the travel time is described in the next section.

5.1.1 Arrival time and objective function calculation

Given a sequence of vertices, an algorithm is required to calculate for every included vertex the arrival time distribution. This distribution allows us to determine the probability that a reward or a penalty is collected and therefore allows us to calculate the expected profit of the vertex. The output of this algorithm is the expected profit when executing the proposed sequence (the a-priori route) in a stochastic context. In this research, two methods are proposed for this task: a Monte Carlo simulation and an estimation algorithm.

In general, three cases can occur when travelling from one vertex to another vertex:

1. The traveller arrives within the opening hours of the vertex which enables the traveller to collect the reward after servicing the vertex. A service time is added to the arrival time.
2. The traveller arrives before the opening time of the vertex which forces him to wait first and collect the reward after servicing the vertex. The departure time is equal to the opening time plus the service time.
3. The traveller arrives after the closing time of the vertex which forces him to collect the penalty without servicing the vertex. No service time is added in this case and the departure time is equal to the arrival time.

Recall that the closing time of the end depot is set equal to $t_0 + t^{max}$ and a penalty, e_v is defined. Therefore, the third case also penalizes for the fact of having a total time greater than t^{max} .

Given the fact that our travel time distributions are assumed to be normally distributed, the resulting arrival distribution which is equal to the convolution of the departure time distribution and the corresponding travel time distribution will also be normally distributed if the departure time is also normally distributed. In most cases this departure time will be normally distributed. However, problems arise when waiting is necessary at a vertex, when the vertex can not be served or when the width of the departure time distribution spans multiple time slots. As will be discussed in the three subsequent subsections, in such cases the arrival and/or departure time distribution and the resulting distribution can no longer be categorized as normal. Therefore, executing the convolution of such an irregular departure time distribution with a subsequent normal travel time distribution becomes a complex task. Sampling these distributions during the execution of a metaheuristic is computationally expensive. Therefore, an estimation algorithm

(Section 5.1.1.5) is used during the execution of the metaheuristic and its final solution sequence is validated using a Monte Carlo simulation (Section 5.1.1.4).

These three cases will now be explained, together with the way they are dealt with during the execution of the metaheuristic, i.e. by the estimation algorithm. Note that for a given random variable D , ϕ_D stands for its probability density function, Φ_D stands for its cumulative distribution function and Φ_D^{-1} represents its inverse cumulative distribution function. All equations derived in the subsequent sections were obtained and validated using the commercial software Mathematica 10.0 combined with the mathStatistica 2.7 package.

5.1.1.1 Width of departure time distribution spanning multiple time slots

When calculating the arrival time distribution, a time slot needs to be found which corresponds with a departure time distribution. Based on this time slot the correct travel time distribution can be obtained. Recall that each arc can have a different travel time distribution for each time slot. Therefore, it is important that the departure time distribution is convoluted with the travel time distribution that matches the time slot of that departure time distribution. However, the departure time distribution may span more than one time slot. If the mean departure time would be used to find an appropriate time slot, a part of the departure time distribution is convoluted with the wrong travel time distribution.

$D \sim \mathcal{N}(\mu_0, \sigma_0^2)$ represents the departure time and $g > 0$ the start time of a time slot. Furthermore, let $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ be the travel time corresponding to the first time slot and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ the travel time corresponding to the second time slot. Then, the random variable A represents the arrival time and is equal to the following piecewise function:

$$A = \begin{cases} D + X_1 & \text{if } D \leq g \\ D + X_2 & \text{if } D > g. \end{cases}$$

The probability density function of A therefore equals:

$$\begin{aligned} \phi_A = & \frac{e^{-\frac{(\mu_0 + \mu_1 - t)^2}{2(\sigma_0^2 + \sigma_1^2)}} \left(\operatorname{erf} \left(\frac{\sigma_1^2(g - \mu_0) + \sigma_0^2(g + \mu_1 - t)}{\sqrt{2}\sigma_0\sigma_1\sqrt{\sigma_0^2 + \sigma_1^2}} \right) + 1 \right)}{\frac{\sqrt{\sigma_0^2 + \sigma_1^2}}{2\sqrt{2\pi}}} \\ & - \frac{e^{-\frac{(\mu_0 + \mu_2 - t)^2}{2(\sigma_0^2 + \sigma_2^2)}} \left(\operatorname{erf} \left(\frac{\sigma_2^2(g - \mu_0) + \sigma_0^2(g + \mu_2 - t)}{\sqrt{2}\sigma_0\sigma_2\sqrt{\sigma_0^2 + \sigma_2^2}} \right) - 1 \right)}{\frac{\sqrt{\sigma_0^2 + \sigma_2^2}}{2\sqrt{2\pi}}}. \end{aligned} \quad (5.1)$$

The mean of the arrival time distribution is equal to:

$$\mu_A = \mu_0 + \frac{1}{2}\mu_1 \left(\operatorname{erf} \left(\frac{g-\mu_0}{\sqrt{2}\sigma_0} \right) + 1 \right) + \frac{1}{2}\mu_2 \left(\left(-\operatorname{erf} \left(\frac{g-\mu_0}{\sqrt{2}\sigma_0} \right) - 1 \right) + 1 \right). \quad (5.2)$$

The standard deviation is equal to:

$$\sigma_A = \frac{1}{2} \sqrt{ e^{-\frac{(g-\mu_0)^2}{2\sigma_0^2}} \left(e^{\frac{(g-\mu_0)^2}{2\sigma_0^2}} \left((\mu_1 - \mu_2)^2 \left(-\left(\operatorname{erf} \left(\frac{g-\mu_0}{\sqrt{2}\sigma_0} \right) - 1 \right) \right) \right. \right. \right. \\ \left. \left. + 2\sigma_1^2 \operatorname{erf} \left(\frac{g-\mu_0}{\sqrt{2}\sigma_0} \right) + 2 \left(-\sigma_2^2 \left(\operatorname{erf} \left(\frac{g-\mu_0}{\sqrt{2}\sigma_0} \right) - 1 \right) + 2\sigma_0^2 + \sigma_1^2 \right) \right) \right. \\ \left. + 4\sqrt{\frac{2}{\pi}} (\mu_2 - \mu_1) \sigma_0 \right)}. \quad (5.3)$$

In these equations *erf* stands for the error function. Details on the derivations of these equations can be found in Appendix A.1.

As an example we choose the departure time distribution $D \sim \mathcal{N}(11, 1^2)$. The start of time slot 2 is located at 12 (g), the travel time associated with time slot 1 is $X_1 \sim \mathcal{N}(1, 0.1^2)$ and the travel time associated with time slot 2 is $X_2 \sim \mathcal{N}(1.2, 0.5^2)$. Note that at this point no information concerning the destination vertex is needed since the arrival time distribution is calculated. This example is visualised in Figure 5.1. On the left panel we can see the departure time distribution in black, as well as the start and end of time slot 1 indicated by a black vertical line. The part of the distribution that is located after the end of time slot 1 is shaded in orange. The part of the distribution that is located before the start of time slot 1 is very small and is therefore ignored. In this case the part of departure time distribution ranging from 7 to 12 needs to be convoluted with X_1 while the part of departure time distribution ranging from 12 to 15 needs to be convoluted with X_2 . On the right panel the real arrival time distribution is drawn in black.

As can be seen in this specific case, the original arrival time distribution does not look normally distributed but is relatively well approximated using a normal distribution. During the execution of the metaheuristic the arrival time distribution is assumed to be normally distributed with a mean and standard deviation equal to equation 5.2 and 5.3. The approximation for this specific case is drawn in dashed orange.

Figure 5.1: Effect of departure time spanning multiple time slots on arrival time distribution with $\mu_0 = 11$, $\mu_1 = 1$, $\mu_2 = 1.2$, $\sigma_0 = 1$, $\sigma_1 = 0.1$, $\sigma_2 = 0.5$ and $g = 12$

5.1.1.2 Arriving early at a vertex

When a traveller arrives before the opening time of a vertex, a waiting time and service time are added to the arrival time. In a stochastic case, waiting at a vertex converts the normally distributed arrival time distribution into a departure time distribution which is a combination of a mass point and a truncated normal distribution.

The random variable D represents the departure time and equals the following piecewise function:

$$D = \begin{cases} o + s & \text{if } A \leq o \\ A + s & \text{if } A > o. \end{cases}$$

Its probability density function therefore equals

$$\phi_D = \begin{cases} \frac{e^{-\frac{(\mu_a + s - t)^2}{2\sigma_a^2}}}{\sqrt{2\pi}\sigma_a} & o + s - t < 0 \\ \frac{1}{2} \left(\operatorname{erf}\left(\frac{-\mu_a - s + t}{\sqrt{2}\sigma_a}\right) + 1 \right) & t = o + s. \end{cases} \quad (5.4)$$

The mean and standard deviation (μ_d and σ_d) of this departure time distribution are given by the following equations:

$$\mu_d = \frac{1}{2}(o - \mu_a) \left(\operatorname{erf}\left(\frac{o - \mu_a}{\sqrt{2}\sigma_a}\right) - 1 \right) + \frac{\sigma_a e^{-\frac{(o - \mu_a)^2}{2\sigma_a^2}}}{\sqrt{2\pi}} + o + s, \quad (5.5)$$

$$\sigma_d = \frac{1}{2} \sqrt{4 \left(\frac{\sigma_a(\mu_a - o)e^{-\frac{(o-\mu_a)^2}{2\sigma_a^2}}}{\sqrt{2\pi}} - \frac{\sigma_a^2}{2} \right) \operatorname{erf} \left(\frac{o - \mu_a}{\sqrt{2}\sigma_a} \right) - (o - \mu_a)^2 \operatorname{erf} \left(\frac{o - \mu_a}{\sqrt{2}\sigma_a} \right)^2 + \mu_a^2 + o^2 - \frac{2\sigma_a^2 e^{-\frac{(o-\mu_a)^2}{2\sigma_a^2}}}{\pi} - 2\mu_a o + 2\sigma_a^2}. \quad (5.6)$$

Details on the derivations of these equations can be found in Appendix A.2. Note that the service time does not occur in Equation 5.6.

As an example, we define the arrival time distribution $A \sim \mathcal{N}(\mu_a, \sigma_a^2)$ with a mean of 9 and a standard deviation of 1, the opening time is equal to 10 (o) and the service time (s) is set equal to 1. This example is visualised in Figure 5.2. On the left panel we can see the arrival time distribution in black and the opening time indicated by a black vertical line. The part of the distribution that is located before the opening time (84%) is shaded in orange. On the right panel the real (piecewise) departure time distribution is drawn in black. Since we have a 84% chance to arrive early, the real departure time consists of a mass point of 84% at time 11 (the sum of the opening time and the service time). The other part of the departure time distribution is formed by shifting, the part of the arrival time distribution that is located after the opening time, 1 time unit to the right.

During the execution of the metaheuristic, this piecewise departure time distribution is approximated as a normal distribution with a mean and standard deviation equal to equation 5.5 and 5.6 respectively. The approximation for this example is drawn in dashed orange.

Figure 5.2: Arrival and departure time distribution in the case of an early arrival with $\mu_a = 9$, $\sigma_a = 1$, $s = 1$ and $o = 10$.

As can be seen in the expression of both the mean and the standard deviation

of the departure time distribution, the opening time has an effect on the departure time distribution. An example of this effect for an arrival time distribution with a mean equal to 9 and a standard deviation and service time equal to 1 can be seen in Figure 5.3. Note that in this figure the opening time is variable while the arrival time is kept constant. This means for example that an opening time greater than 9 corresponds to a scenario with a high probability of arriving early. In Figure 5.3a can be seen that for a high probability of arriving early (right side of Figure 5.3a) the mean departure time is equal to the opening time plus the service time. However, for low probability of arriving early (left side of Figure 5.3a) the mean is equal to the mean arrival time (9) plus the service time (1) of the vertex. In between, a convex function limits the growth of the departure time when the opening time increases. On Figure 5.3b can be seen that the standard deviation decreases in a non-linear way when the probability of arriving early increases up to the fact that the departure time becomes deterministic. It can be concluded that arriving early at vertices decreases the variability in a TD-OPSWTW solution due to the waiting before the time window opens.

Figure 5.3: Effect of the opening time on μ_d and σ_d with $\mu_a = 9$, $\sigma_a = 1$ and s equal to 1.

5.1.1.3 Arriving late at a vertex

A vertex cannot be served for the part of the arrival time distribution that is greater than the closing time, so no service time should be added to this part of the arrival time distribution. Recall that the time windows are defined for the start of the service and not for the completion of the service.

When the arrival time distribution is defined as $A \sim \mathcal{N}(\mu_a, \sigma_a^2)$, the random variable D represents the departure time and equals the following piecewise function:

$$D = \begin{cases} A + s & \text{if } A \leq c \\ A & \text{if } A > c. \end{cases}$$

Note that the transformation from A to D is not one to one. In particular, there is a zone of overlap where both subfunctions of the piecewise function can contribute to one value of D .

The probability density function of D is given by:

$$\phi_D = \begin{cases} \frac{e^{-\frac{(t-\mu_a)^2}{2\sigma_a^2}}}{\sqrt{2\pi}\sigma_a} & c + s \leq t \\ \frac{e^{-\frac{(\mu_a+s-t)^2}{2\sigma_a^2}}}{\sqrt{2\pi}\sigma_a} & c \geq t \\ \frac{e^{-\frac{(\mu_a+s-t)^2}{2\sigma_a^2}}}{\sqrt{2\pi}\sigma_a} + e^{-\frac{(t-\mu_a)^2}{2\sigma_a^2}} & c < t < c + s. \end{cases} \quad (5.7)$$

The mean and standard deviation (μ_d and σ_d) of the resulting departure time distribution are given by the following equations:

$$\mu_d = \frac{1}{2} \left(s \operatorname{erf} \left(\frac{c - \mu_a}{\sqrt{2}\sigma_a} \right) + 2\mu_a + s \right), \quad (5.8)$$

$$\sigma_d = \sqrt{-\frac{1}{4}s^2 \operatorname{erf} \left(\frac{c - \mu_a}{\sqrt{2}\sigma_a} \right)^2 - \sqrt{\frac{2}{\pi}}\sigma_a s e^{-\frac{(c-\mu_a)^2}{2\sigma_a^2}} + \frac{s^2}{4} + \sigma_a^2}. \quad (5.9)$$

Details on the derivations of these equations can be found in Appendix A.3. Note that the service time does occur in Equation 5.9.

As an example we define the arrival time distribution $A \sim \mathcal{N}(\mu_a, \sigma_a^2)$ with a mean of 16 and a standard deviation of 1, the closing time (c) is equal to 17 and the service time (s) is set equal to 1. This example is visualised in Figure 5.4. On the left panel we can see the arrival time distribution in black and the closing time indicated by a black vertical line. The part of the distribution that is located after the closing time is shaded in orange. On the right panel the real (piecewise) departure time distribution is drawn in black and the proposed approximation in orange. Note that the elevated part of the original departure time distribution between 17 and 18 originates from both arrival times that were either on-time or too late. For example an arrival time equal to 16.5 is transformed to a departure time equal to 17.5 but an arrival time equal to 17.5 leads also to the same departure time of 17.5.

During the execution of the metaheuristic, this piecewise departure time distribution is approximated as a normal distribution with a mean and standard deviation equal to equation 5.8 and 5.9 respectively. The approximation for this example is drawn in dashed orange.

Figure 5.4: Arrival and departure time distribution in the case of a late arrival with $\mu_a = 16$, $\sigma_a = 1$, $s = 1$ and $c = 17$.

Similarly to the effect of the opening time, an example of the effect of the closing time on the mean and standard deviation of the departure time distribution for two different values of the service time is visualised in Figure 5.5. For the case where $s = 1$, the mean departure time increases and becomes closer to the default value of the mean departure time (17) as the probability of arriving late (higher values of c) increases (Figure 5.5a). The effect of the closing time on the standard deviation resembles in this case a parabolic function which opens downwards, indicating that the standard deviation first decreases for high probabilities of arriving late, then hits a minimum for the 50% probability of arriving late and then increases again when the probability of arriving late decreases (Figure 5.5b). For the case where the service time equals 3, the curve for the mean departure time follows a similar pattern as discussed above (Figure 5.5c). However, the curve of the standard deviation shown in Figure 5.5d deviates strongly from Figure 5.5b. The standard deviation increases for low probabilities of arriving late and achieves a maximum for the 50% probability whereafter the standard deviation drops for high probabilities of arriving late. Finally, there can be concluded that late arrivals severely complicate the calculation of the departure time distribution.

Figure 5.5: Effect of the closing time on μ_d and σ_d with $\mu_a = 16$, $\sigma_a = 1$ and s respectively equal to 1 and 4.

5.1.1.4 Monte Carlo simulation

The pseudo code of the Monte Carlo is provided in Algorithm 7. Since in every iteration the departure time is deterministic, it is possible to find exactly one corresponding travel time distribution with a deterministic departure time. Subsequently, a random percentile of this travel time distribution is chosen as travel time value. The deterministic arrival time therefore equals the sum of the deterministic departure time and the randomly chosen percentile of the travel time distribution. Afterwards, the three cases mentioned above are checked for this arrival time value and a corresponding reward or penalty is obtained. This process is repeated for every vertex in the solution. This algorithm returns the expected reward of the input solutions sequence by dividing the sum of all rewards belonging to the 10,000 iterations by the number of iterations. This maximum number of iterations is set equal to 10,000 since preliminary experiments showed that the objective function no longer shows significant deviations. Although not shown in Algorithm 7, this Monte Carlo simulation is also extended to capture the arrival time distribution

at the end vertex of the solution. This allows to measure the difference in mean and standard deviation between the arrival time distribution, originating from the Monte Carlo simulation, and the one produced by the estimation algorithm. This comparison can be found in Section 5.2.1.

Algorithm 7 Monte Carlo simulation- input: solution sequence S

```

expreward = 0
for iter to maxiterations do
  reward = 0
  departuretime =  $t_0$ 
  for  $i = 1$  to  $|S|$  do
     $a = S[i - 1]$ 
     $b = S[i]$ 
     $k = \text{findtime slot}(\text{departuretime})$ 
     $\text{rnd} = \text{random number between } ]0, 1[$ 
     $\text{arrivaltime}+ = \Phi_{T_{a,b,k}}^{-1}(\text{rnd})$ 
    if  $\text{arrivaltime} < o_b$  then
       $\text{departuretime} = o_b + s_b$ 
       $\text{reward}+ = r_b$ 
    else if  $\text{arrivaltime} \leq c_b$  then
       $\text{departuretime}+ = s_b$ 
       $\text{reward}+ = r_b$ 
    else
       $\text{reward}- = e_b$ 
       $\text{departuretime} = \text{arrivaltime}$ 
    end if
  end for
   $\text{expreward}+ = \text{reward}$ 
end for
RETURN  $\text{expreward}/\text{maxiterations}$ 

```

5.1.1.5 Estimation algorithm

This algorithm provides an estimate of the departure time distribution when travelling from vertex i to vertex j with a departure time distribution at i equal to ϕ_{D_i} . The pseudo code is displayed in Algorithm 8. During the execution, every distribution is assumed to be normal which speeds up calculating the convolution product of two distributions.

In general, to construct the arrival time distribution, the departure time distribution is convoluted with the matching normal travel time distribution (lines 10-12). The time slot k , corresponding to the mean of the departure time distribution, is selected to find the corresponding travel time distribution (line 1). However, if the

proportion of the departure time distribution that is located in another time slot than the time slot that corresponds with the mean of the distribution, is bigger than 1%, the arrival time distribution will be approximated. If a part (greater than 1%) of the departure time distribution is situated in a lower time slot, then g is set equal to k . The arrival time distribution is approximated using a normal distribution with a mean equal to equation 5.2 and a standard deviation equal to equation 5.3 (lines 2-5). In the other case that a part (greater than 1%) of the departure time distribution is located in a later time slot, then g is set equal to $k + 1$. The arrival time distribution is approximated using a normal distribution with a mean equal to equation 5.2 and a standard deviation equal to equation 5.3 (lines 6-9).

After the convolution of the departure time and the travel time into the arrival time distribution, three scenarios can occur on arrival at vertex j : an early arrival, a late arrival or an on-time arrival. For an early arrival the departure time is approximated using a normal distribution with a mean equal to equation 5.5 and a standard deviation equal to equation 5.6 (lines 14-16). In the case of a late arrival the departure time is approximated using a normal distribution with a mean equal to equation 5.8 and a standard deviation equal to equation 5.9 (lines 17-19). An on-time arrival requires the addition of a service time to the mean of the arrival time distribution in order to obtain the departure time distribution. The standard deviation remains unchanged (lines 20-22). Finally, this algorithm outputs the expected reward of vertex j .

5.1.2 Pre-processing

As discussed more in detail in Section 4.4.2, a set of neighbours is defined for each vertex $i \in V_c$. This set, NB_i , is computed before the start of the solution procedure and consists of the closest (based on the time-independent travel time) vertices in the sense of spatial-temporal closeness. In the remainder of this text, a vertex j is called a neighbouring vertex of vertex i if j is included in the neighbour set of i .

5.1.3 Stochastic ant colony system (SACS)

The structure of the *stochastic ant colony system* (SACS) is similar to the ant colony system proposed for the TD-OPTW as described in Section 4.4 and consists of 3 phases which are executed on a set of solutions during a predetermined number of iterations. Firstly, ants, which represent solutions, are created in a construction phase. The second phase consists of an improvement phase where the ants are improved using three local search moves. Thirdly, in the global pheromone update phase the pheromone values belonging to the arcs of the best ant (of the iteration) are increased. An overview of the algorithm is provided in Algorithm 9 and the different components are described below. The best solution found in an iteration

Algorithm 8 Estimation algorithm -input: vertex i , vertex j , departure time distribution $D_i \sim \mathcal{N}(\mu, \sigma^2)$

```

1:  $k = \text{findtimeslot}(D_i, \mu)$ 
2: if  $\Phi_{D_i}(tp_k) > 0.01$  then
3:    $g = tp_k$ 
4:    $A_j.\mu = \text{eq. 5.2}(g, D_i, T_{i,j,k-1}, T_{i,j,k})$ 
5:    $A_j.\sigma = \text{eq. 5.3}(g, D_i, T_{i,j,k-1}, T_{i,j,k})$ 
6: else if  $\Phi_{D_i}(tp_{k+1}) - \Phi_{D_i}(tp_k) < 0.99$  then
7:    $g = tp_{k+1}$ 
8:    $A_j.\mu = \text{eq. 5.2}(g, D_i, T_{i,j,k}, T_{i,j,k+1})$ 
9:    $A_j.\sigma = \text{eq. 5.3}(g, D_i, T_{i,j,k}, T_{i,j,k+1})$ 
10: else
11:    $A_j.\mu = D_i.\mu + T_{i,j,k}.\mu$ 
12:    $A_j.\sigma = \sqrt{D_i.\sigma^2 + T_{i,j,k}.\sigma^2}$ 
13: end if
14: if  $\Phi_{A_j}(o_j) > 0.01$  then
15:    $D_j.\mu = \text{eq. 5.5}(o_j, A_j)$ 
16:    $D_j.\sigma = \text{eq. 5.6}(o_j, A_j)$ 
17: else if  $\Phi_{A_j}(c_j) < 0.99$  then
18:    $D_j.\mu = \text{eq. 5.8}(c_j, A_j)$ 
19:    $D_j.\sigma = \text{eq. 5.9}(c_j, A_j)$ 
20: else
21:    $D_j.\mu = A_j.\mu + s_j$ 
22:    $D_j.\sigma = A_j.\sigma$ 
23: end if
24: RETURN  $D_j$ 

```

is defined as sol_{ib} , sol_{gb} represents the best solution found during the entire optimization procedure and $F(sol_x)$ refers to the objective function (expected reward) of solution sol_x .

Algorithm 9 Ant colony system for TD-OPSWTW - input parameters:
 $\alpha, \beta, \rho, \text{max_ants}, \tau_{init}, N_{ni}^{max}$

```

 $sol_{ib} \leftarrow 0, sol_{gb} \leftarrow 0, N_{ni} \leftarrow 0, \text{iteration} \leftarrow 0$ 
while  $\text{iteration} < N_c$  do
  Initialize parameters:  $\tau, \eta(\tau_{init})$ 
  for  $i \leftarrow 1$  to  $\text{max\_ants}$  do
    Construct solution & local pheromone update ( $\tau, \eta, \alpha, \beta, \rho$ )
    Swap
    Calculate  $\text{max\_start}$ 
    Insert ( $\text{max\_start}$ )
    Replace ( $\text{max\_start}$ )
  end for
   $sol_{ib} \leftarrow \arg \max(F(sol_1), F(sol_2), \dots, F(sol_{\text{max\_ants}}))$ 
  if  $F(sol_{ib}) > F(sol_{gb})$  then
     $sol_{gb} \leftarrow sol_{ib}$ 
     $N_{ni} \leftarrow 0$ 
  else
     $N_{ni} \leftarrow N_{ni} + 1$ 
  end if
  Global pheromone update ( $\tau, sol_{ib}, N_{ni}, N_{ni}^{max}$ )
   $\text{iteration} \leftarrow \text{iteration} + 1$ 
end while
Monte Carlo simulation ( $sol_{gb}$ )

```

5.1.3.1 Construction phase

The construction phase, displayed in Algorithm 10, starts with an empty solution. During each iteration a random vertex from the candidate list is selected using a roulette wheel selection method. During the execution of this method, the probability of selection is based on a memory structure (τ) and a heuristic ratio (η). The candidate list consists of all unvisited vertices that can be inserted into the solution and increase the expected reward of the solution when added. This increase in expected reward of the solution consists of the increase in expected reward, due to adding the unvisited vertex to the solution, minus the change in expected reward when travelling from this newly added vertex to the end vertex. In short, this means that late arrivals are allowed during the construction phase as long as it is beneficial. Finally, this phase stops when the candidate list is empty except for the end vertex. The memory structure follows the same logic as described in Section 4.4. The pheromone values of arcs used during the construction phase are

decreased while the values of the arcs belonging to the iteration best solution are increased.

Algorithm 10 SACS construction phase- input parameter: α, β

```

i = 0
solution sol with sol[i] = 1
D[i]. $\mu$  = 0
D[i]. $\sigma$  = 0
while S[i]  $\neq v$  do
    CL =  $\emptyset$ 
    a = sol[i]
    for all available neighbours b of a except v do
        X = estimation algorithm(a, b, D[i])
        expreward =  $\Phi_X(c_b) * r_b + (1 - \Phi_X(c_b)) * e_b$ 
        V = estimation algorithm(b, v, X)
        endexpreward = expreward -  $(1 - \Phi_V(c_v)) * e_v$ 
        if endexpreward > 0 then
            add vertex b to CL
            store X and expreward
        end if
    end for
    if CL  $\neq \emptyset$  then
        for all vertices r  $\in$  CL do
            calculate lr ( $\alpha, \beta$ )
        end for
        roulette wheel selection method
        add selected vertex to sol and update the solution
    else
        sol[i + 1] = v
    end if
    i = i + 1
end while
local pheromone update

```

5.1.3.2 Using local search moves in a stochastic context

Firstly, the swap local search move tries to exchange two solution member vertices in order to save travel time. We define *y* and *z* as the vertices that will be exchanged, *k* and *l* as the predecessor and successor of *y* and *m* and *n* as the predecessor and successor of *z*. After verification that *k* and *l* are neighbours of *z* and that *m* and *n* are neighbours of *y*, the vertices *y* and *z* are temporarily exchanged. Otherwise, this swap pair (*y*, *z*) is discarded before further evaluation. Afterwards, the new arrival time for the vertices between *k* and *n* is calculated

while accounting for time window restrictions. Next, the decrease in travel time (calculated from k to n) of the possible swap is calculated using the estimation algorithm. Only if this decrease in mean and standard deviation is positive, the swap is executed. However, it is unsure that this exact decrease in travel time can be gained also for the whole solution due to waiting times and the time-dependent stochastic nature of succeeding travel times after vertex n . Nevertheless, since the mean and standard deviation of the departure time distribution at n are reduced we are sure that departing earlier from vertex n guarantees that our new arrival time at the end depot will be less than or equal to the previous one. Therefore this approach is chosen to avoid a complete evaluation of the solution.

Secondly, the insert and replace local search moves make use of a local evaluation metric called `max_start`. This metric is created by calculating for every vertex in the current solution, the latest possible start time of the service at the vertex, before the objective function of the solution decreases, using the 99th percentile of the travel time distribution. Other percentiles (50th, 77.1th, 90th) were tested but the results did not deviate much. This can be explained by the relatively high penalty for arriving late at the end vertex. This process is shown in Algorithm 11. Note that the latest possible start time is searched given the current total length of the route. This means that if the solution already contains a late arrival at the end depot, this increase in travel time compared to t^{max} is taken into account as feasible time during the calculation of `max_start` (lines 7-9).

Concerning the insert procedure, a vertex y can only be inserted between x and z into the solution when the 99th percentile of the new arrival time distribution at z (calculated using the estimation algorithm) is less than the `max_start` of vertex z . When vertex y is actually included in the solution, the algorithm updates the arrival times from x to y , as well as the arrival times at the vertices succeeding vertex y . This is necessary, because the insertion of vertex y has most likely caused a change in waiting and/or travel time for the arcs of the solution succeeding vertex y . Secondly, for the vertices preceding vertex z , a recalculation of the `max_start` local evaluation metric is required. The insert local search move is executed in a best-improving manner and stopped when no more feasible improvements can be found.

A similar logic is applied for the replace move which tries to replace an included vertex by a non-included vertex with a higher expected reward. The replace local search move is run in a first-improving manner. Both local search moves are stopped when no more feasible improvements can be found.

Algorithm 11 Calculation of max_start - input solution *sol*

```

1:  $sz = |sol|$ 
2: latestarrival =  $\Phi_{D[sz]}^{-1}(0.99)$ 
3: if latestarrival  $\leq t^{max}$  then
4:   max_start[sz] =  $t^{max}$ 
5:    $t_a = t_0 + t^{max}$ 
6: else
7:   max_start[sz] = latestarrival
8:    $t_a = \text{latestarrival}$ 
9: end if
10:  $i = sz - 1$ 
11: while  $i > 0$  do
12:    $z = sol[i + 1]$ 
13:    $y = sol[i]$ 
14:    $k = \text{find time slot } (t_a)$ 
15:    $t_d = t_a - \Phi_{T_{y,z,k}}^{-1}(0.99)$ 
16:   if  $t_d > c_y + s_y$  then
17:      $t_d = c_y + s_y$ 
18:   end if
19:   max_start[i] =  $t_d$ 
20:    $t_a = t_d - s_y$ 
21:    $i = i - 1$ 
22: end while

```

5.2 Results for the TD-OPSWTW

5.2.1 Validation of the estimation algorithm

The estimation algorithm is compared with the Monte Carlo procedure, both described in Section 5.1.1, based on the final solution sequence (sol_{gb}) outputted by the SACS. Recall that different values for the mean and standard deviation of the arrival and departure time distribution lead to different expected rewards and penalties. Therefore the same solution sequence can have a different objective function when evaluated by the estimation algorithm or the Monte Carlo procedure.

The average difference after five runs over all instances concerning the arrival time distribution at the end vertex is less than 0.01% for the mean arrival time at the end vertex and 0.67% for the standard deviation. This results in a objective function deviation of only 0.01%. As a comparison, during the initial research phase, the SACS was also executed using an inferior estimation algorithm. During this algorithm each normal arrival/departure time distribution was discretised using four-point Gaussian quadratures of a normal distribution. The benefits and downsides of this approximation are discussed by Bickel et al. [16]. The corresponding action to each of the three cases discussed in Section 5.1.1 is then applied separately to the four different percentiles of the arrival/departure distribution. When comparing this method to the results of the Monte Carlo procedure, this way of working lead to an average deviation of 0.04% for the mean, 7.9% for the standard deviation and 0.15% for the objective function. The simplest estimation algorithm consists of taking the convolution of the departure time distribution and the travel time distribution (which corresponds with the mean of the departure time distribution) while ignoring the three different cases. This leads to an average deviation of 0.02% for the mean, 8.19% for the standard deviation and 0.32% for the objective function.

5.2.2 Evaluation of the SACS algorithm

To the best of our knowledge, this is the first time the TD-OPSWTW is proposed. Therefore the SACS algorithm cannot be compared to other solution methods. However, to evaluate its performance, all TD-OPSWTW instances were also solved as deterministic and time-independent orienteering problems with time windows (OPTW) according to the mixed integer problem formulation presented in Section 2.1 using a commercial solver. Afterwards, this optimal solution is evaluated in a stochastic context using the Monte Carlo simulation. Then, a comparison can be made between this stochastically evaluated OPTW solution and a genuine stochastic solution proposed by the SACS. The same input parameter values for the SACS are used as the ones mentioned in Table 4.14 for the ACS for TD-OPTW. More specifically, the percentage difference between the expected re-

ward of SACS solution method and the expected reward of the evaluated optimal OPTW solution is calculated as follows :

$$\% imp = \frac{SACS\ expreward - OPTW\ expreward}{OPTW\ expreward} * 100 \quad (5.10)$$

Another performance metric is the required CPU time. The SACS was allowed to generate 10,000 trial solutions. The performance of the algorithm is displayed in the second and third column of Table 5.1.

Secondly, the SACS will be compared with the ACS for the TD-OPTW using a predefined travel time percentile on every arc. The resulting deterministic TD-OPTW solutions are subsequently validated using a Monte Carlo simulation. This way of working differs from the two stage approach used in [41, 42] for the fact that the route is never aborted and penalties are imposed here for unvisited vertices and late arrivals. Table 5.2 provides an overview of the effect of executing the algorithm with a different percentile of the travel time distribution on every arc. Constructing routes with the 1st percentile corresponds to searching for an optimistic solution, while using the 99th percentile corresponds to finding a limited-risk or pessimistic solution. In the last column the average improvement is displayed when the best solution is taken over all solutions corresponding to the 5 different travel time percentiles. Based on this table, using the 77th percentile of the travel time distribution results in the best performance for this set of instances. In general, the optimal percentile (π) is instance specific and depends on the reward-penalty ratio and the amount of uncertainty in the network.

The SACS outperforms the regular ACS on quality (1%) but needs significantly more computation time than the deterministic version of the ACS. However, this computation time is relative as in reality the ACS needs to be executed for every percentile of the travel time distribution in order to determine which π returns the best result. The solutions proposed by the SACS are on average 23.8% better than executing the optimal solution sequence calculated for a stochastic environment. It turns out that not taking into account the time-dependent and stochastic nature of the travel time has a significant impact on the quality of the proposed solutions in a realistic problem context. Finally, it can be concluded that accounting for time-dependent and stochastic travel times during the optimization process augments the performance of the proposed solutions but at the same time also requires higher computational efforts.

Table 5.1: Overview on the performance of SACS and ACS

$ V_c $	SACS		ACS ($\pi=77.1\text{th}$)	
	% imp	cpu (s)	% imp	cpu (s)
20	21.9	1.2	21.0	0.1
50	22.7	3.8	21.5	0.4
100	26.8	5.9	26.0	0.7
Overall	23.8	3.7	22.8	0.4

Table 5.2: The avg % imp for the ACS for different percentiles of the travel time distribution

$ V_c $	Percentile of travel time distribution (π)					
	1st	22.9th	50th	77.1th	99th	best
20	7.9	13.2	19.0	21.0	21.0	22.1
50	16.4	18.4	21.9	21.4	21.4	22.6
100	16.5	21.9	26.8	26.0	25.7	27.3
Overall	13.6	17.8	22.6	22.8	22.7	24.0

5.2.3 Sensitivity analysis

Table 5.3 provides an overview of the input parameters of SACS, including their value used during all the discussed experiments. These values were determined after preliminary tests. A sensitivity study, similar to the one of the ACS for the TD-OP as discussed in Section 4.3.3, is performed to measure the impact on the performance for various combinations of input parameter values. The results are displayed in Table 5.4. An ANOVA test points out that the significant input parameters that have an impact on the performance are β , ρ and `max_ants`.

Table 5.3: Overview of the input parameters TD-OPSWTW

Input parameter	Description	Value
τ_{init}	initial pheromone value	1
α	importance of pheromone information	4
β	importance of heuristic information	1
<code>max_ants</code>	number of solutions that is constructed and improved per iteration	50
ρ	evaporation rate	0.01
N_{ni}^{max}	maximum number of iterations without improvement (% of N_c)	25%

Table 5.4: Effect of α , β , N_{ni}^{max} , ρ and `max_ants` on the average gap

α	% gap	β	% gap	N_{ni}^{max}	% gap	ρ	% gap	<code>max_ants</code>	% gap
1	24.4	1	25.8	0.25	25.3	0.01	24.4	10	24.1
3	25.8	3	25.8	0.50	25.2	0.05	25.5	25	25.0
6	25.4	6	25.2	0.75	25.1	0.10	25.5	50	25.5
9	24.9	9	23.7	1.00	25.0			75	25.5
								100	25.5
Total	25.1	Total	25.1	Total	25.1	Total	25.1	Total	25.1

Table 5.5 provides insights into the performance of some design decisions. These effects are measured by leaving out some or all of the 3 local search moves (S: swap, I: insert, R: replace) and repeating the test procedure discussed in Section 5.2.2. The missing components in the SACS framework (Algorithm 9) are marked with a "-". Based on the average improvement, we conclude that the strength of the SACS lies in the interaction effect between the swap, insert and replace local search moves. However, the insert local search move remains the most prominent local search move while the replace move helps to reduce the computation time if the swap and insert local search moves are used together.

Table 5.5: Effect on the % gap and average CPU time (s) of different design components

Metric	S I R	- - -	S - -	- I -	- - R	- I R	S I -	S - R
avg % imp	23.8	16.9	19.0	22.5	17.9	23.0	23.4	18.5
avg cpu	3.7	0.8	0.9	2.9	1.0	3.4	5.1	1.3

5.2.4 Impact of stochastic travel times

To measure the impact of not taking into account stochastic travel times during the optimisation process, the SACS approach is compared against the solution to the certainty equivalence problem using the ACS. When solving the certainty equivalence problem the mean of the stochastic travel time is taken as travel time estimate.

The detailed results are displayed in Table 5.6 and a summary is provided in Table 5.7. The first four columns of Table 5.6 list respectively the name, number of vertices, the maximum allowed travel time and time window restrictiveness for each problem instance. The succeeding column displays the expected reward of the evaluated optimal ACS solution. In the 6th column the expected reward (expreward) of the TD-OPSWTW solution created by the SACS is given. Finally, the last two columns display the percentage improvement (imp) of the SACS versus the evaluated TD-OPTW solution and also the diversity (div) between the two solution sequences. The diversity between two solutions A and B is calculated as the

sum of the number of vertices in A not present in B and the number of vertices in B not present in A, divided by the total number of vertices present in A and B (start and end vertex not included).

The SACS on average can improve the expected reward with 1.3% in comparison with the ACS. A time-dependent stochastic orienteering problem has on average a solution that differs 28.3% from the time-independent and deterministic one. Based on Table 5.7, the SACS is more likely to improve instances with short time windows and a shorter allowed maximum travel time as more late arrivals can be avoided. Instances with more vertices and a higher maximum allowed travel time are more difficult to improve by the SACS and therefore negative improvements are sometimes obtained. As the problem becomes less constrained, the advantage of the deterministic approach is rewarded because of its simplicity.

Table 5.7: Impact of stochasticity versus problem characteristics

$ V_c $	imp	div	t^{max}	imp	div	TW	imp	div
20	3.3	22.0	8	3.2	26.8	S	3.0	30.9
50	0.8	25.5	10	0.6	23.7	M	0.4	29.4
100	-0.2	37.5	12	0.1	22.5	L	0.5	24.7
			14	0.8	28.5			

These results are road network and instance specific but nonetheless they reflect that including stochastic information in the model can improve the objective value.

Recall that the coefficient of variation ($cv = \frac{\sigma}{\mu}$) for the original set of instances was rather low (0.5%) and the penalty ratio ($pr = \frac{r}{e}$) was equal to 20% for the regular vertices while the penalty of the end depot was set equal to the highest score of the instance. Therefore, the focus of the algorithm lies in generating routes with a high probability to arrive on time.

In order to test a more diverse set of stochastic situations, the same test setup was performed with higher coefficients of variation and also different penalty ratios for all vertices including the end depot. To achieve this, for every arc and timeslot combination the standard deviation of the travel time (σ) was set equal to $cv \cdot \mu$. The percentage average improvement over all instances is displayed in Table 5.8. Firstly, there can be seen that the average improvement drastically increases when both the coefficient of variation and the penalty ratio are set to higher levels. The lowest result obtained in Table 5.8 equals 12.5% while the overall average improvement for the original instances displayed in Table 5.6 is 1.3%. Therefore, the logical conclusion that can be made is that it is worthwhile to take stochasticity into account when the variability of the travel times is high. Secondly, given a certain coefficient of variation a higher penalty ratio always leads to a bigger increase in overall average improvement. Thirdly, given a certain penalty ratio the overall average improvement decreases when the coefficient of variation increases. This is due to the fact that the `max_start` local evaluation metric uses the 99% travel time percentile. Therefore, a vertex can only be inserted when it is almost certain that the insertion will not lead to an additional lateness for subsequent vertices. This leads to very few candidates for feasible insertion when the variation is extremely high. For a cv equal to 30%, the results can be improved by using a lower percentile (e.g. 77.1%) to calculate `max_start`.

Table 5.8: Impact on the average improvement (%) for different coefficients of variation (%) and penalty ratios (%)

$\begin{array}{c} cv \\ \swarrow \\ pr \end{array}$	10	20	30
25	16.7	14.2	12.5
50	22.0	19.1	17.9
100	42.7	38.2	36.5

5.2.5 Practical insights

When it is 100% certain that a vertex is visited on time, its standard deviation of the departure time is equal to the standard deviation of its arrival time. The probability of arriving early at a vertex decreases the standard deviation of the departure time (Section 5.1.1.2). As mentioned in Section 5.1.1.3, the probability of a late arrival can lead to an increase or decrease of the standard deviation of the departure time (σ_d) depending on the ratio of the standard deviation of the arrival time (σ_a) and the service time (s). When s is much higher than σ_a (more than twice the size), σ_d can become higher than σ_a . In reality the service time is also likely to be higher than σ_a as the service time is usually several minutes while σ_a is zero at the beginning of the solution sequence and increases towards the end of the solution sequence as variance propagates through the system. Therefore, in some cases, a possible late arrival at the end of the solution sequence decreases the variation of the route.

Contrary to the deterministic version, a swap operator can, apart from reducing the travel time, also increase the expected reward in a stochastic context. For example, if the swap operator is used on a solution with a late arrival at a vertex, the reduction in travel time can lead to the fact that the traveller no longer arrives late at this vertex. This means that the expected reward obtained at this vertex will increase. Therefore, an update of the expected reward starting from vertex z is necessary after a successful swap move has been executed. Two additional (less restrictive) decision criteria have been tried for evaluating possible swap pairs apart from comparing both the difference between the mean and standard deviation values of the departure time at n : only comparing the difference between the mean departure times at n and comparing the difference between the 99th percentiles of the departure time at n . The problem with these last two criteria is that a reduction in mean departure time can go along with a significant increase in standard deviation of the departure time which might cause a decrease in the objective function. This can happen because the probability of arriving late at subsequent vertices might increase as well. There exist multiple reasons for this phenomenon. Firstly, executing a swap move means that a set of arcs is replaced by another set of arcs in the solution and this new set of arcs might have a higher standard deviation. Sec-

only, as already mentioned above, late arrivals at the end of the solution decrease the variation of the route. If after the execution of a swap move the late arrivals become on time arrivals, this in turn might lead to an increase in route variability which in most cases means a late arrival at the end vertex. Only a full evaluation of the solution for every swap pair can ensure that this adverse effect is not present. But since a full evaluation is too computationally expensive we opted to compare the difference of a possible swap move on both the mean and standard deviation.

Finally, using a 99th percentile of an updated arrival time distribution whether to decide if a vertex can be inserted or replaced means that there is also a 1% probability that you arrive late at one or more subsequent vertices (in most cases also the end vertex) because of this decision. Furthermore, inserting a vertex also leads to an increase in variance that propagates through the system. However, the adverse effects of this assumption are marginal and the expected reward of inserting a vertex largely compensates for the very small expected penalty of a late arrival towards the end of the solution sequence. Nonetheless, the insert and replacement moves might cause small violations to the time budget but this effect is preferred as similar results cannot be achieved using the deterministic ant colony system and its local search moves.

5.3 Conclusion

In this chapter, the orienteering problem with time windows and time-dependent stochastic travel times (TD-OPSWTW) is introduced and a solution method is proposed. This specific problem formulation applies to congestion related issues in realistic routing problems that deal with uncertainty. The solution method consists of a stochastic version of an ant colony optimization (SACS) specifically developed for the TD-OPSWTW. The SACS uses a swap local search move and an insert and replace local search move equipped with a local evaluation metric which speeds up the insertion/replacement process. The combination of time-dependency, time windows and stochastic travel time severely complicates the calculation of departure and arrival time distributions. Therefore, three prominent complications are defined and the way they were dealt with by the proposed estimation algorithm is explained. This might also be useful for other vehicle routing problems with time windows and time-dependent stochastic travel times.

The proposed solution method is evaluated based on solution quality and computational performance on altered TD-OPTW problem instances. Promising results were obtained in a reasonable amount of computation time. A first comparison is made by comparing the results of the SACS with optimal solutions for the regular OP with time windows, found using an exact solution method, that are evaluated in a stochastic environment. The solutions proposed by the SACS are on average 23.8% better than executing the optimal solution sequence in a time-

dependent stochastic environment. Secondly, the solutions generated by the ACS for TD-OPTW, using different percentiles of the travel time distribution as input, were also evaluated in a stochastic environment and compared to the solutions of the SACS. The SACS performs better than the ACS, independent of the used travel time percentile, as it takes into account the stochastic nature of this problem. However, the SACS also requires a greater computation time. Thirdly, when solving the certainty equivalence problem using the ACS method for the TD-OPTW, the SACS improves solutions by 1.3% on average. When solving the certainty equivalence problem, the mean of the stochastic travel time is taken as travel time estimate. The fourth observation is the solution sequence being on average 28.3% different compared to the proposed solution by the state-of-the-art algorithm. An important practical insight is that a probability of a late arrival at a customer can lead to either an increase or decrease of the standard deviation of the departure time. When it is 100% certain that a customer is visited on time, its standard deviation of the departure time is equal to the standard deviation of the arrival time. The probability of arriving early at a customer decreases the standard deviation of its departure time. The effect of a possible late arrival depends on the ratio of the standard deviation of the arrival time and the length of the service time. In general, at the beginning of the solution sequence, the service time is likely to be higher than the standard deviation of the arrival time as the service time comprises usually several minutes while the standard deviation of the arrival time is equal to zero at that point. However, the standard deviation of the arrival time tends to increase towards the end of the solution sequence. In short, a probability of a late arrival in the beginning of the solution sequence increases the standard deviation of the arrival time whereas a possible late arrival at the end of the solution sequence might decrease the standard deviation of the arrival time. The combination of these results should motivate vehicle trip planners to consider the time-dependent and stochastic nature of travel times in the future.

6

Case studies: solving a multi-constrained team orienteering problem with time-dependent travel times

In this chapter, the solution method for the TD-OPTW, discussed in Chapter 4, will be adapted and applied to a real case of two logistics companies. Apart from company specific data, the road network with travel time information from Be-Mobile (Section 3.2.2) is used to solve these cases. Both problem instances can be modelled using the same mathematical problem formulation. In comparison to the problem formulation of the TD-OPTW, this formulation has to consider multiple days and vehicles and has more practical requirements to take into account. These additional requirements consist out of a lunch break, customers having a different time window per day and two capacity limitations for each vehicle. This problem bears many similarities with some of the variants of the orienteering problem discussed in Section 2.5 and is called the multi-constrained time-dependent team orienteering problem with time windows (multi-constrained TD-TOPTW). Unlike in Chapter 5, the travel times are assumed to be dynamic but not stochastic in this chapter.

The mathematical formulation of this problem is discussed in more detail in Section 6.1. Details about the logistic cases are described in Section 6.2. As some of the practical requirements drastically change the way a solution can be

evaluated and altered by the local search moves, the adaptations to the solution method are discussed in Section 6.3 and 6.4. Finally, the results of this algorithm will be compared to solutions that do not take time-dependent travel times into account in Section 6.5.

6.1 Mathematical formulation

Formally, the multi-constrained TD-TOPTW can be described by defining a set of vertices $V_c = 1, \dots, v$. In this set vertex 1 represents the start depot and vertex v the end depot. We assume there is an arc (i, j) between all i and j in V_c . As this is an extension of the TOPTW the goal is to determine δ number of routes, each route d limited by t_d^{max} . The predefined vehicle that is assigned to route d has a limited volume capacity (vol_d^{max}) and weight capacity (wei_d^{max}). Moreover, the predefined driver needs to take a break for a duration of brk^{dur} during a certain time window (brk^{start}, brk^{end}). Note that this break time window is the same for all routes but this break needs to be taken on arrival at a vertex. This assumption represents the current practice of both companies and therefore allows for a comparison of our proposed solutions to the solutions of the commercial VRP solver as will be explained in Section 6.5. Associated with each vertex $i \in V_c$ is a non-negative reward r_i , weight wei_i , volume vol_i , service time s_i and a set of time windows that differ per route ($o_{i,d}, c_{i,d}$). The reason for this will be explained in the next paragraph. Note that a break can be taken at a vertex before its opening time. A day is divided in κ time slots and let $\underline{tp}_t$ and $\overline{tp}_t$ represent the moment at which time slot t starts and ends respectively. Subsequently, based on these time slots and their corresponding travel times for each arc, the linear travel time coefficients for each arc $\mu_{i,j,t}$ and $\nu_{i,j,t}$ can be calculated as discussed in Section 3.2.1.

The set V_c represents customer locations that can be served by a fleet of vehicles during a fixed planning period i.e. a week. The reward is related to the urgency of visiting this vertex, the higher the reward, the more urgent it is to visit this vertex during this planning period. The time window can be different for each day in the planning period. However, for simplicity reasons the time window is coupled with a particular route index as stated in the previous paragraph. For example if the planning period is equal to 3 days and the fleet size equals 5, 15 unique route indices are used in the mathematical formulation. For each vertex and route index, an unique time window is possible. The reason for this is twofold: in one of the cases only one vehicle is used during the whole planning period. For example route 1 stands for a vehicle used on Monday and route 2 stands for the same vehicle used on Tuesday. So there is no distinction between a route index and a day index. Secondly, this way of modelling also enables the modelling of specific customer requirements such as a request to be served by a specific driver or enables the mod-

elling of the restriction that a vehicle is not able to enter the hangar because of the vehicle dimensions. Finally, note that the volume and weight requirements are not related to the reward of the vertex.

The decision variables and parameters are listed before the mixed integer problem formulation is shown and discussed.

Decision variables

$x_{d,i,j,t}$ = 1 if a traveller traverses the arc (i, j) with a departure time in time slot t on route d , 0 otherwise.

$w_{d,i,j,t}$ = departure time at vertex i when travelling from i to j in time slot t on route d , 0 otherwise.

$b_{d,i,j,t}$ = equals 1 if a break is scheduled on arrival at vertex j when traversing arc (i, j) in time slot t on route d , 0 otherwise.

Parameters

κ : number of time slots for all arcs

δ : number of routes

$V_c = 1, \dots, v$: set of vertices

$\underline{tp}_t$: start of time slot t

r_i : reward of vertex i

wei_i : weight of vertex i

vol_i : volume of vertex i

$o_{i,d}$: opening time of vertex i when serviced on route d

$c_{i,d}$: closing time of vertex i when serviced on route d

s_i : service time of vertex i

$\mu_{i,j,t}$: slope coefficient of the linear time-dependent travel time

$\nu_{i,j,t}$: intercept coefficient of the linear time-dependent travel time

t_d^{max} : time budget of route d

wei_d^{max} : maximum weight of route d

vol_d^{max} : maximum volume of route d

brk^{start} : earliest start of the break

brk^{end} : latest start of the break

brk^{dur} : duration of the break

$$\text{Max} \sum_{d=1}^{\delta} \sum_{i=2}^{v-1} \sum_{j=2}^v \sum_{t=1}^{\kappa} r_i x_{d,i,j,t} \quad (6.1a)$$

$$\sum_{d=1}^{\delta} \sum_{j=2}^v \sum_{t=1}^{\kappa} x_{d,1,j,t} = \sum_{d=1}^{\delta} \sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} x_{d,i,v,t} = \delta \quad (6.1b)$$

$$\sum_{d=1}^{\delta} \sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} x_{d,i,j,t} \leq 1 \quad \forall j \quad (6.1c)$$

$$\sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} x_{d,i,h,t} = \sum_{j=2}^v \sum_{t=1}^{\kappa} x_{d,h,j,t} \leq 1 \quad \forall d, h = 2, \dots, v-1 \quad (6.1d)$$

$$x_{d,i,j,t} \cdot \underline{tp}_t \leq w_{d,i,j,t} \quad \forall d, i = 1, \dots, v-1, j = 2, \dots, v, \forall t \quad (6.1e)$$

$$w_{d,i,j,t} \leq x_{d,i,j,t} \cdot \underline{tp}_{t+1} \quad \forall d, i = 1, \dots, v-1, j = 2, \dots, v, \forall t \quad (6.1f)$$

$$\sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} [w_{d,i,h,t} + \mu_{i,h,t} \cdot w_{d,i,h,t} + (\nu_{i,h,t} + s_h) \cdot x_{d,i,h,t}] \quad (6.1g)$$

$$+ b_{d,i,h,t} * brk^{dur} \leq \sum_{j=2}^v \sum_{t=1}^{\kappa} w_{d,h,j,t} \quad \forall d, h = 2, \dots, v-1$$

$$\begin{aligned} & \sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} [w_{d,i,v-1,t} + \mu_{i,v-1,t} \cdot w_{d,i,v-1,t} + \nu_{i,v-1,t} \cdot x_{d,i,v-1,t} + b_{d,i,v-1,t}] \\ & * brk^{dur} \leq t_d^{max} \quad \forall d \end{aligned} \quad (6.1h)$$

$$\sum_{i=1}^{v-1} \sum_{j=2}^v \sum_{t=1}^{\kappa} x_{d,i,j,t} \cdot wei_j \leq wei_d^{max} \quad \forall d \quad (6.1i)$$

$$\sum_{i=1}^{v-1} \sum_{j=2}^v \sum_{t=1}^{\kappa} x_{d,i,j,t} \cdot vol_j \leq vol_d^{max} \quad \forall d \quad (6.1j)$$

$$b_{d,i,j,t} \cdot brk^{start} \leq w_{d,i,j,t} + \mu_{i,j,t} \cdot w_{d,i,j,t} + (\nu_{i,j,t} + s_j) \cdot x_{d,i,j,t} \quad \forall d, i, j, t \quad (6.1k)$$

$$\begin{aligned} & w_{d,i,j,t} + \mu_{i,j,t} \cdot w_{d,i,j,t} + (\nu_{i,j,t} + s_j) \cdot x_{d,i,j,t} \\ & \leq b_{d,i,j,t} \cdot brk^{end} + (1 - b_{d,i,j,t}) \cdot t_d^{max} \quad \forall d, i, j, t \end{aligned} \quad (6.1l)$$

$$\sum_{i=1}^{v-1} \sum_{j=2}^v \sum_{t=1}^{\kappa} b_{d,i,j,t} = 1 \quad \forall d \quad (6.1m)$$

$$x_{d,i,j,t} \geq b_{d,i,j,t} \quad \forall d, i, j, t \quad (6.1n)$$

$$\sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} (o_{h,d} + s_h) \cdot x_{d,i,h,t} \leq \sum_{j=2}^v \sum_{t=1}^{\kappa} w_{d,h,j,t} \quad \forall d, h = 2, \dots, v-1 \quad (6.1o)$$

$$\sum_{j=2}^v \sum_{t=1}^{\kappa} w_{d,h,j,t} \leq \sum_{i=1}^{v-1} \sum_{t=1}^{\kappa} (c_{h,d} + s_h) \cdot x_{d,i,h,t} \quad \forall d, h = 2, \dots, v-1 \quad (6.1p)$$

$$x_{d,i,j,t} \in (0, 1) \quad (6.1q)$$

$$b_{d,i,j,t} \in (0, 1) \quad (6.1r)$$

$$0 \leq w_{d,i,j,t} \leq t_d^{max} \quad \forall d, i, j, t \quad (6.1s)$$

The decision variable $x_{d,i,j,t}$ is equal to 1 when travelling from i to j with a departure time in time slot t on route d , and equals 0 otherwise. The continuous decision variable $w_{d,i,j,t}$ contains the departure time at vertex i when travelling to vertex j in time slot t on route d . This departure time at vertex i equals the sum of the arrival time at vertex i , the break time, the waiting time at vertex i and the service time s_i . Finally, $b_{d,i,j,t}$ equals 1 if a break is scheduled on arrival at j when traversing arc (i,j) in time slot t on route d , and equals 0 otherwise.

The objective function and the first five constraints are similar to the TD-OPTW except for the fact that multiple routes have to be created. The objective function (6.1a) maximizes the total collected reward over all routes. Constraint (6.1b) guarantees that each route starts at vertex 1 and ends in vertex v . Constraints (6.1c) make sure that every arc is only travelled once and constraints (6.1d) ensure the connectivity of the route. Constraints (6.1e), (6.1f) determine the departure time in the right time slot which is necessary to multiply the departure time with its corresponding μ and ν value in Constraints (6.1g) and (6.1h).

Constraints (6.1g) guarantee that the departure time of the next vertex in the route is equal to or greater than the sum of the departure time of the previous vertex together with the travel time, break time and service time. The travel time is calculated as a linear function. In this function the departure time is multiplied with a parameter $\mu_{i,j,t}$, whereafter parameter $\nu_{i,j,t}$ is added. Constraint (6.1h) ensures the limited time budget. Constraints (6.1i and (6.1j) ensure that the total weight and volume for each route is within the acceptable limits. Constraints (6.1k) and (6.1l) make sure that the break for every route is taken during the allowed period. Furthermore, Constraints (6.1m) make sure that only one break is taken and Constraints (6.1n) ensure that this break is taken on arrival at a vertex. Constraints (6.1o) ensure that the departure time at each vertex except from the start and end vertex is higher than the opening time. Constraints (6.1p) in turn, ensure that the departure time is lower than the closing time of the vertex under consideration. The range of the decision variables is controlled in Constraints (6.1q), (6.1r) and (6.1s).

A small problem instance and its optimal solution, found using the commercial solver CPLEX, are shown in Figure 6.1. For this instance we assume that all travel times are time-independent and equal to the euclidean distance between the vertices.

Figure 6.1: An example of the multi-constrained TD-TOPTW

6.2 Case studies

In this section some background information is given about both case studies.

6.2.1 Case study 1: setting and instance generation

The first case study consists of a logistics company with a start and end depot at two different locations in The Netherlands (Europe). The company is charged with

the task of picking up containers of used batteries at a set of customer locations (represented by vertices) spread throughout the whole of The Netherlands. People can drop off their used batteries at containers that are placed at various frequently visited locations (i.e. supermarkets). A recycle company is responsible for picking up (almost) full containers and replace them with empty ones. Locations that have an (almost) full container are more urgent to visit than locations with (almost) empty containers.

For this particular instance the planning period was situated in the first week of 2015 and the total number of vertices that could be visited is equal to 832. For each of these vertices the required information is available: geographical location (latitude, longitude), reward, time window for each day of the planning period, required weight and required volume. For this case the average length of the time window at a vertex is rather large. Recall that the reward of the vertex depends on their urgency level. This urgency level is based on a forecast system and companies calling the recycle company to inform them that their container is (almost) full. When a phone call is received, the vertex representing this location is defined as an obligatory vertex. Obligatory vertices have a reward between 1,000 and 2,000 while non-obligatory vertices have a reward between 1 to 10. Note that since this is an OP and not a VRP, a solution without the inclusion of all obligatory vertices is still a feasible solution (given that all other constraints are met). The solution method does not force obligatory vertices to be included in a solution. However, due to the large difference in rewards, any good solution will include all obligatory vertices, if this is feasible.

The planning period consists of 3 days and the company has a fleet of 5 vehicles with the same weight and volume capacities available for each day. According to the reasoning mentioned above, this leads to a total of 15 different routes that need to be generated. All routes start at 4:00 and the first 10 routes have to end before 17:00 and the last 5 before 15:00. On all routes, a break of 45 minutes needs to be taken at a vertex between 10:00 and 15:30. An overview of the characteristics of this case can be found in Table 6.1.

Table 6.1: Overview characteristics case 1

# depots	2
# vertices	832
# obligatory vertices	146
# non-obligatory vertices	686
# vehicles	5
# days	3
# avg time window length	10.46h
# start break	10:00
# end break	15:30

The travel times between the set of vertices is calculated using the same procedure as discussed in Section 4.4.2. However, since the historic travel time profile consisted of car measurements, these travel times were limited to a maximum of 80km/h on every arc which is the maximum allowed speed for trucks driving on highways in The Netherlands. Subsequently, a matching needs to occur between the set of vertices representing customer locations (V_c) and the set of vertices belonging to the Be-Mobile road network (V) discussed in Section 3.2.2. Note that previously, during the creation of instances for the TD-OPTW in Section 3.2.2, a matching was not necessary as V_c was chosen as a subset of V . In this chapter, however, V_c is not a subset of V . Therefore, for each vertex representing a customer location the nearest vertex on the Be-Mobile road network is searched using the great-circle distance. The remaining distance in kilometres between the customer location and the nearest vertex on the Be-Mobile graph is multiplied with 2 minutes of travel time to approximate the remaining travel time. Since the graph of the Be-Mobile network is dense, the average remaining distance equals 0.17 kilometres and the maximum distance amounts 1.81 kilometres. The travel times on the arcs are assumed to be FIFO.

6.2.2 Case study 2: setting and instance generation

The second case models the problem of a logistics company that needs to deliver bakery products from a single depot to a set of 160 customers (represented by vertices) in the upper part of Belgium. For this particular instance the planning period was situated in the last week of February 2015. For each of these customers the required information is available: geographical location (latitude, longitude), reward, time window for each day of the planning period, required weight and required volume. Almost all customers are obligatory and therefore have a reward equal to or higher than 1,000.

For this case, the time period consists of a single Friday for a heterogeneous fleet of 9 vehicles which means that 9 routes need to be planned. All routes start at 6:30 and the trucks need to be back at the end depot before 16:30. Furthermore, a break of 45 minutes needs to be taken between the above mentioned start and end hour of each route. Secondly, the maximum weight and volume capacity differs within the set of vehicles. An overview on the characteristics of this case can be found in Table 6.2.

Concerning the generation of the travel times, since all vertices are situated in Belgium, the maximum speed of the vehicles is limited to 90km/h which corresponds to the maximum allowed speed on highways for trucks. The same matching method is applied as explained in Section 6.2.1. For this case, the average remaining distance equals 0.15 kilometres and the maximum distance amounts 1.96 kilometres.

Table 6.2: Overview characteristics case 2

# depots	1
# vertices	160
# obligatory vertices	153
# non-obligatory vertices	7
# vehicles	9
# days	1
# avg time window length	9.79h
# start break	6:30
# end break	16:30

6.3 Pre-processing

A k-means clustering algorithm is performed on the subset of obligatory vertices with k , the number of clusters, equal to the number of routes (δ). The k-means algorithm was originally published by Forgy [49] and aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space. The average weight and volume requirements are calculated for all the members of a certain cluster. Subsequently, the cluster with the highest capacity requirements is assigned to a route with the highest available capacity.

Secondly, the same neighbourhood structure as discussed in Section 4.4.2 is used. However, a different ratio is used to sort the feasible neighbours of every vertex:

$$\frac{r_j}{ff_{i,j} + s_j + wei_j + vol_j} \quad (6.2)$$

The maximum number of neighbours (NB_{max}) is set equal to 200. This means that no limit on the maximum number of neighbours is set in the second case as the vertices are located very close to each other and the time-windows are too loose. So all feasible neighbours are retained for this case.

6.4 Solution method

The ant colony system proposed in Chapter 4.4 for the TD-OPTW is adapted in order to solve this multi-constrained TD-TOPTW problem. In this section the difference between the two solution methods will be described. The construction method and the local search moves are altered to work on multiple routes and to include the two capacity constraints. The development of a relocate local search move which removes a vertex from one route and inserts it in another route also aided to this course.

The main challenge, however, resides in the incorporation of the breaks when evaluating local search moves. Firstly, during the optimization process we assume that breaks have to be scheduled as soon as possible. More specifically, if no break is scheduled in the tour and a new vertex is added to the solution which causes the arrival time to become greater than brk^{start} , then a break is scheduled on arrival at this newly added vertex. This assumption is made to facilitate the solution method and for computational efficiency. Determining when a break should be positioned during the construction of a solution would require too much travel time calculations. Secondly, if no break is necessary (because of an arrival at the end vertex before the start time of the break) it is placed at the end vertex but the actual break time is added to the arrival time at the end vertex. This helps to reposition the break later on (i.e. after a successful insert local search move) at a regular vertex. Thirdly, the addition of breaks to this problem also violates the FIFO principle. This means that arriving earlier at a vertex does not always lead to a decrease or status quo of the departure time at succeeding vertices in the solution sequence. For example, assume a break can only start at 10:00 (brk^{start}) and in the current solution a break is scheduled at vertex i at 10:00. If for instance after performing a swap local search move, the arrival time at vertex i decreases to 9:50, the break might need to be rescheduled at a succeeding vertex ($i + 1$). This means that arriving earlier at vertex i still leads to an earlier arrival at vertex $i + 1$ but to a postponed departure at vertex $i + 1$. Due to this postponed departure time succeeding travel times might increase which can eventually lead to a late arrival at the end vertex. Additionally, note that it is therefore possible that this decrease in travel time of 10 minutes can be offset by an increase in travel time for arcs in the solution sequence after vertex $i + 1$. In both examples a violation of the FIFO principal occurs.

The general outline of this solution method is given in Algorithm 12. In the succeeding subsections the adaptations and additions to the original TD-OPTW solution method will be discussed. The main strategy is to create diverse solutions during the construction procedure using only obligatory vertices. Subsequently, these routes are then improved using local search moves that can also use non-obligatory vertices. To facilitate the understanding of the local search moves, figures which contain examples, have been made and were added to each section. For all these examples the vertices are assumed to have no time windows and a service time equal to zero. The legend for these figures can be found in Figure 6.2. The difference with previous figures is that a break is now shown by colouring the border of a vertex or depot in orange.

No changes were made to the initialisation procedure as compared to Section 4.4.3.

Algorithm 12 Ant colony system for multi-constrained TD-TOPTW - input parameters: $\alpha, \beta, \rho, \text{max_ants}, \tau_{init}, N_{ni}^{max}$

```

 $sol_{ib} \leftarrow 0, sol_{gb} \leftarrow 0, N_{ni} \leftarrow 0, \text{iteration} \leftarrow 0$ 
while  $\text{iteration} < N_c$  do
  Initialize parameters:  $\tau, \eta(\tau_{init})$ 
  for  $i \leftarrow 1$  to  $\text{max\_ants}$  do
    Construct solution & local pheromone update ( $\tau, \eta, \alpha, \beta, \rho$ )
    Swap
    2-opt
    Calculate  $\text{max\_shift}$ 
    Insert ( $\text{max\_shift}$ )
    Relocate ( $\text{max\_shift}$ )
    Insert ( $\text{max\_shift}$ )
  end for
   $sol_{ib} \leftarrow \arg \max(F(sol_1), F(sol_2), \dots, F(sol_{\text{max\_ants}}))$ 
  if  $F(sol_{ib}) > F(sol_{gb})$  then
     $sol_{gb} \leftarrow sol_{ib}$ 
     $N_{ni} \leftarrow 0$ 
  else
     $N_{ni} \leftarrow N_{ni} + 1$ 
  end if
  Global pheromone update ( $\tau, sol_{ib}, N_{ni}, N_{ni}^{max}$ )
   $\text{iteration} \leftarrow \text{iteration} + 1$ 
end while

```

Figure 6.2: Legend to figures of Chapter 6

6.4.1 Construct solution

For every route, feasible (obligatory) vertices belonging to the assigned cluster are subsequently added to a candidate list. Feasible vertices are vertices that do not incur violations to the volume and weight limitation and the time budget restriction. For the time budget restriction, the time required to go back to the end depot and to schedule a required break are taken into account. This break is added as soon as the arrival time at a vertex becomes greater than brk^{start} . Based on this candidate list, a vertex is chosen based on the roulette wheel selection as discussed in Section 4.4.4. The process stops when the candidate list is empty and the end vertex is added to the route. If no break is incorporated yet in the solution sequence, the break is positioned at the end depot.

6.4.2 Swap

In the original implementation (Section 4.4.6), a swap move was executed when a decrease in travel time was found at vertex n (the successor of the swap partner positioned the closest to the end vertex) and all neighbour restrictions held. Because of the violation to the FIFO property, the feasibility of a swap local search move can only be evaluated at the end vertex because it is possible that a break needs to be repositioned. After a neighbour check, possible interesting swap pairs are searched by subsequently performing a local evaluation and a global evaluation. Recall that during the execution of the local search moves all vertices (obligatory and non-obligatory) are taken into account and no cluster information is used. During the local evaluation the break can not be repositioned. The subsequent global evaluation is only considered when a lower travel time is obtained at vertex y . If the arrival at the end vertex is decreased by executing the possible swap move, the swap combination is performed on the solution in a first improvement manner. If after the execution of a swap move the arrival time at the end vertex is less than brk^{start} , the break is scheduled at the end vertex.

An example of a swap move is visually depicted in Figure 6.3. In this example, vertex y is swapped with vertex z . The local evaluation shows that a decrease in travel time of 0.4 units can be found when evaluating the difference at vertex n . During the global evaluation, the break can no longer be positioned at vertex l as the arrival time (2.8) at vertex l is less than brk^{start} (3). Therefore, the break is postponed and positioned at vertex m . The result of the swap move is evaluated at the end vertex and results in a decrease in travel time of 0.5 units.

Figure 6.3: Example of the swap local search move

6.4.3 2-opt

This intra route local search move changes the sequence between two vertices y and z with index i and j in one and the same route as discussed in Section 4.2.2. After a neighbour check, possible interesting 2-opt pairs are searched by subsequently performing a local evaluation and a global evaluation. During the local evaluation the break can not be repositioned. The subsequent global evaluation is only considered when a lower travel time is obtained at $j + 1$. This global evaluation is only performed when the break in the new solution would be positioned before brk^{start} or after brk^{end} which turns the solution infeasible. During the global evaluation the infeasible solution is repaired by repositioning the break and this new solution is accepted when the new arrival time at the end vertex is lower than the original arrival time. This local search move is employed in a first improvement manner.

The 2-opt local search move is visually presented in Figure 6.4. In this example y and z are interchanged and the sequence between these two vertices is reversed. During the local evaluation the break remains scheduled at the third position of the solution sequence, meaning at vertex m . A decrease in travel time (0.5 units) is obtained at vertex n during the local evaluation. Due to our arrival at vertex m on 2.8, the break is positioned too early and will be repositioned to the fourth position during the global evaluation. An update of the travel times from vertex k until the end vertex reveals that a decrease in travel time of 0.4 time units can be achieved when executing this specific 2-opt pair.

Figure 6.4: Example of the 2-opt local search move

6.4.4 Calculation of `max_shift`

This local evaluation metric will be used by the insert and relocate local search move. Recall from Section 4.4.5 that this metric measures the maximum amount of time a vertex can be shifted towards the end of the route before the solution becomes infeasible. However, apart from the time budget and the closing time an additional limitation, namely the latest start of the break (brk^{end}), needs to be considered. This means that when a break is scheduled at a vertex and this vertex is shifted towards the back of the solution sequence, it is possible that the break is positioned after brk^{end} . Therefore, an additional check needs to be performed when calculating the `max_shift` of a vertex where a break is positioned. Reducing the `max_shift` of a vertex leads to less room for a possible insertion and is therefore not beneficial to the optimization process. Therefore we decided to schedule a break as soon as possible. However, as will be explained in Section 6.4.5 it is still possible for this phenomenon to occur due to a series of insertions.

6.4.5 Insert

This local search move is very similar to the insert move described in Section 4.4.7. This move does not reposition the break, when inserting vertices in a solution, but postpones them towards the end of the solution sequence. Recall that postponing a break after brk^{end} is prohibited by the use of the `max_shift` metric. The only exception to this rule occurs when a break was scheduled at the end vertex and the arrival time of the end vertex has become equal to or greater than brk^{start} due to the insertion of a vertex in the route. In this case the break is repositioned towards the first regular vertex with an arrival time greater than or equal to brk^{start} . This local search move is executed in a best improvement manner.

The insert move is visually presented in Figure 6.5. In this example vertex y is tested for insertion between x and z . Note that the break is scheduled at the end vertex since the arrival at z is less than brk^{start} . Vertex y can be inserted because the increase in travel time (0.5 units) is less than the `max_shift` of vertex z .

However, the arrival time at z (2.5) is now equal to brk^{start} , so the break needs to be placed at z instead of at the end vertex.

Figure 6.5: Example of the insert local search move

6.4.6 Relocate

The idea behind this local search move is to reposition a vertex x from route d on position i to route e ($d \neq e$) on position j in order to obtain an overall decrease in travel time. So this local search move consists of two parts: first a removal on route d and an insertion on route e without repositioning the break. At the start of the relocate local search move the variable *besttotaldecrease* is set equal to 0. After a neighbour and capacity check on both routes, two evaluations (local and global) are performed for every vertex of route d and for every position j on route e . During the local evaluation the break remains stationary on route d and the relocation is assumed feasible when the increase in arrival time at position i on route d is less than or equal to $\max_shift_{j+1}$. For all feasible insertions the variable *localtotaldecrease* is set equal to the decrease in travel time on route d measured at position $i + 1$ minus the increase in travel time on route e measured at $j + 1$. If the *localtotaldecrease* is bigger than *besttotaldecrease*, a global evaluation is performed. During the global evaluation the break on route d can be repositioned and the decrease in travel time on route d is evaluated at the end vertex. The variable *totaldecrease* can now be calculated as the decrease in travel time on route d minus the increase in travel time on route e . If this *totaldecrease* is greater than *besttotaldecrease* then the relocate combination is set as the best relocate combination. The best relocate combination is performed until no more improvements can be found. The relocate local search move is executed in a best improving manner.

An example of this local search move is visually presented in Figure 6.6. Vertex x is removed from route d and inserted between vertices a and b on route e .

This causes a decrease in time of 1.6 units on route d evaluated at vertex z and an increase for 0.2 units on route e . Vertex x can be inserted on route e since the increase is less than the `max_shift` of vertex b . Note that the break at vertex y on route d is now positioned too early ($1.4 < 2$). Therefore it needs to be repositioned at vertex z during the global evaluation. Afterwards, the *localtotaldecrease* becomes equal to $1.6 - 0.2 = 1.4$. A global evaluation for route d reduces this overall decrease (*totaldecrease*) to $1.5 - 0.2 = 1.3$ time units. The decrease in travel time is positive, so this relocate combination can be executed.

Figure 6.6: Example of the relocate local search move

6.4.7 Remaining procedures

If the relocate local search move can improve the solution, some additional room for insertion might be created. Therefore an additional insert move is performed. Finally, the update of the global best solution and the global pheromone update are performed as described respectively in Section 4.4.9 and 4.4.10. The ACS is iteratively executed until 10,000 solutions have been created.

6.5 Results

In this section, the results of the cases will be discussed. The results for each case consist out of three tests. In the first test, the proposed algorithm is run with time-independent travel time data. This means that for every arc all time-dependent travel time estimates are set equal to one time-independent travel time estimate. The algorithm still searches for a solution with the time-dependent solution procedures as described above. This way of working enables a comparison on quality with the time-independent solution provided by a commercial VRP solver that does not take time-dependency into account but uses a single average travel time estimate. As this travel time estimate does not represent a free-flow state estimate on the arc it is possible that the travel times can be overestimated by the commercial VRP solver. So when this solution is evaluated in a time-dependent context the total length of a route can be less than, greater than or equal to the original one. This also means that the objective function of the time-dependent solution can be higher than the time-independent one because it is possible to visit more vertices within the same time span.

During the second test the algorithm is run with the time-dependent travel time information in order to obtain a time-dependent solution. For the third test setup the time-independent solution of the commercial VRP solver is evaluated in a time-dependent environment. Firstly, this allows for an analysis of the effects of time-dependency on this solution sequence. As a second step, a variant to the *heuristic repair* algorithm, described in Section 4.5.5, will be applied to make this solution feasible and allows for a direct comparison with the time-dependent solution created by the proposed algorithm. The difference resides in the fact that after a vertex is removed, a check is performed. This check verifies that the break is positioned between br_k^{start} and br_k^{end} . If this is not the case, then the break is repositioned at the earliest vertex in the sequence with an arrival time greater than br_k^{start} .

6.5.1 Results Case study 1

Using the ACS solution method with the same time-independent travel time estimates as the commercial VRP solver, a solution (TID) with objective function 149,379 is found while the commercial solver itself returns a solution (TID_com) with 149,417. A graphical depiction of this solution can be found in Appendix A.4. In both solutions all (146) obligatory vertices are included. The commercial VRP solvers manages to included more non-obligatory vertices (210) into the solution than our solution method (163). The total travel time of the commercial VRP solver is 8,248 and the total distance amounts 5,364. The total travel time of our solution is 10,439 and the total distance amounts 6,088. The lack in performance can be explained by the fact that our algorithm is not optimized for time-independent travel times. The increase in time and distance can be explained

by the fact that our proposed solution method does not take the travel time and the distance into account in the objective function. For this setup, travel time forms the most restrictive factor in comparison to the volume and weight limitations.

If we evaluate the commercial VRP solution without performing recourse actions in a time-dependent context, for an average route a driver arrives 15 minutes too late at the end depot. The average increase in arrival time at the end depot is 12.5 minutes which also indicates that for some routes the arrival times decrease when taking time-dependency into account. This is possible as the VRP commercial solver uses average travel times which will typically overestimate the travel times before the morning congestion peak (7:00-9:00). If we look at the route level, we notice that some routes have a rather high violation to the time budget:

- Route 6: 38 min too late
- Route 8: 24 min too late
- Route 9: 57 min too late
- Route 12: 53 min too late

This short study validates the fact that congestion is a time-spatial phenomenon that can not be modelled by working with averages. On some routes this leads to an overestimation of the travel time and for others this will lead to a possible significant underestimation of the travel time.

If we perform the time-dependent evaluation on the solution (TD_com_HR), an objective function of 148,385 is obtained as 33 vertices (of which 1 obligatory) were removed from the solution. The time-dependent solution (TD) for this case has an objective function of 149,406 which is higher than the evaluated solution. This solution contains 146 obligatory and 172 non-obligatory vertices and is visually presented in Appendix A.5. The total time amounts 10,381 and distance 6,118. The required computation time to obtain this solution amounts 406 seconds. This again validates that it is better to take congestion into account during the optimization than applying recourse actions to time-independent solutions. Also in the time-dependent environment, travel time forms the most restrictive factor. An overview on the results is presented in Table 6.3.

Table 6.3: Overview results case 1

	obj	# oblig	# non-oblig	time	distance
TID_com	149,417	146	210	8,248	5,364
TID	149,379	146	163	10,439	6,088
TD_com_HR	148,385	145	178	10,264	5,344
TD	149,406	146	172	10,381	6,118

6.5.2 Results Case study 2

The commercial VRP solver is able to find an time-independent solution to this problem which contains all possible vertices (160). This solution has an objective function equal to 154,205, the total time is 4,394 minutes and the total distance in kilometres equals 2,375. Our proposed solution method was also able to find a solution with the same objective function. In this solution all 160 vertices were included, the total time is equal to 4,688 minutes and the total distance amounts 2,869 kilometres. A graphical depiction of this solution can be found in Appendix A.6.

Evaluating the commercial VRP solution without performing recourse actions in a time-dependent context, a driver arrives on average 7.8 minutes too late at the end depot. The average increase in travel time was equal to 37 minutes. Due to time-dependency, a late arrival is observed at a vertex on route 3 and both route 6 and 8 have an arrival time that respectively was 45 and 20 minutes later than the time budget. Note that given this high increase in travel time, only a few routes have a late arrival at the end depot. This is due to the fact that the travel time is not the most limiting factor for this case but the weight capacity is.

After performing the necessary recourse actions on the time-independent solution, the objective function drops to 148,065 as 7 vertices (of which 6 obligatory) are removed from the solution. Solving this problem, using the proposed solution method and the time-dependent travel time information, a solution with an objective function equal to 152,205 is obtained. In this solution 151 obligated vertices and 7 non-obligated vertices are included. The total travel time amounts 4,793 minutes and the total distance amounts 2,810. The solution is visually presented in Appendix A.7. The required computation time to obtain this solution amounts 120 seconds. In this case it also pays off to take time-dependency into account during the optimization process. An overview on the results is presented in Table 6.4.

Table 6.4: Overview results case 2

	obj	# oblig	# non-oblig	time	distance
TID_com	154,205	153	7	4,394	2,375
TID	154,205	153	7	4,688	2,869
TD_com_HR	148,065	147	6	4,584	2,380
TD	152,205	151	7	4,793	2,810

6.6 Conclusion

In this chapter, a real case of two logistics companies is solved by adapting the solution method proposed for TD-OPTW in Chapter 4. In comparison to the problem

formulation of the TD-OPTW, this formulation has to consider multiple days and vehicles and has more practical requirements to take into account. These additional requirements consist out of a lunch break, customers having a different time window per day and two capacity limitations for each vehicle. The problem is called the multi-constrained time-dependent team orienteering problem with time windows (multi-constrained TD-TOPTW). While most of these requirements could easily be implemented into the solution method, the inclusion of a break in every route required substantial adaptations. Furthermore, it substantially increases the computational effort when executing local search moves.

The results of the adapted algorithm are compared to solutions, originating from a commercial VRP solver, that do not take time-dependent travel times into account. These solution are evaluated in a time-dependent context which simulates the performance of a possible execution in real-life. For both cases the proposed solution method outperforms the evaluated solutions within a reasonable computation time. So although time-dependency forms a burden for the computation time, especially when breaks need to be planned in a route, it still pays off to take time-dependency into account during the optimization process. Increasing the performance of the algorithm and considering a multi-objective function that consists of route related costs and time-dependent travel times offers the main avenue for further research. Especially the neighbourhood structure needs to be improved by using more advanced partitioning methods. Moreover, the capacity limitations need to be more actively included during the execution of the construction method and during the selection of promising local search moves.

7

Conclusion

7.1 Concluding summary

Vehicle routing problems (VRP) focus on determining an optimal route to visit all customers using a fleet of vehicles, each constrained by a maximum travel time. The goal is to determine the best order to visit these customers and how to assign the customers to the available vehicles. In the classic VRP the travel time between two customers is assumed to be a constant value such as a distance metric or an average travel time estimate. However, this is not a realistic way of modelling travel times, since planners nowadays have to take into account congestion issues: the travel time between two customers will vary significantly during the day due to external factors such as congestion, road accidents, weather conditions etc. In general, taking time-dependency into account, which means that the travel time between two customers depends on the departure time at the first customer, leads to a more realistic vehicle trip planning process. Apart from an increase in travel time due to a reduction of the speed and queuing, congestion also increases the variability of the travel times and therefore increases the uncertainty about the planned arrival times. Congestion is modelled by making various assumptions about the travel time required to traverse an arc (set of roads that links two customers). If the travel time is a fixed value it is defined as being deterministic. If this travel time is not a fixed value but the value is subject to variations, the travel time is said to be stochastic. On the other hand, a distinction has to be made between time-independent and time-dependent travel times. If the time to traverse an arc is not

dependent on the hour of the day, it is considered to be time-independent and can be represented as a single value. In the other case, when the travel time is different on different moments of the day, the travel time is said to be time-dependent. It should be noted that nowadays in almost all realistic road networks, travel times are both time-dependent and stochastic.

Variants on the vehicle routing problem concerning different types of travel times on the arcs between customers are investigated in this dissertation. More specifically, the main focus lies on developing solution methods for the orienteering problem (OP), a vehicle routing problem where it is not obligatory to visit all customers but visiting a customer allows to gain a specific reward. The goal of the OP is to select a subset of customers that maximises the total collected reward given a maximum allowed travel time. Apart from different types of travel times, additional practical requirements are added to this problem and taken into account during the optimization process.

The first problem that is discussed is called the time-dependent orienteering problem (TD-OP) in which the travel time between two customers depends on the departure time at the first customer. Apart from a mathematical formulation for this problem that allows to calculate optimal solutions for small problem instances, the main contribution is a fast local-search based metaheuristic, inspired by an ant colony system (ACS). The local search mechanism itself is a time-dependent insertion procedure, sped up by a local evaluation metric. Moreover, realistic time-dependent test instances with known optimal solution are developed based on the time-independent OP instances in combination with a well performing speed model for the time-dependent vehicle routing problem, a closely related problem.

The proposed algorithm obtains high-quality results on these instances requiring very small computation times, even for instances with around 100 customers. An average run obtains solutions with a reward gap with the optimal solution of only 1.4% using 0.5 seconds of computation time. For 44% of the test instances, the known optimal solution is found. A sensitivity analysis demonstrates that the performance of the algorithm is not sensitive to small changes in the parameter settings. This indicates that robust behaviour might therefore be expected when incorporating this algorithm into real-life applications. The fast execution time of this algorithm enables some interesting business applications where it is necessary to update routes when new traffic information becomes available and to provide proper guidance to drivers/tourists on the road.

The second problem being considered, is the extension of the time-dependent orienteering problem with time windows. This problem is called the time-dependent orienteering problem with time windows (TD-OPTW). A mixed integer problem formulation and an altered ACS approach are proposed. Apart from an adapted time-dependent insert local search move, a replace and a swap procedure are also

developed. The evaluation of possible improvement moves is fastened by a local evaluation metric and by efficiently limiting the number of customers that are considered in each attempt to improve the current solution. Moreover, realistic time-dependent test instances with known optimal solutions are developed and made publicly available. 36 test instances with a number of customers to visit ranging from 20 to 100 were extracted from a road network containing 84,720 customers and 116,683 arcs.

The TD-OPTW algorithm obtains very good results on these benchmark instances, requiring small computation times due to the combination of a well performing metaheuristic framework and the interaction effect between three efficient local search moves tailored towards the problem characteristics. The average reward gap with the known optimal solution on these test instances is only 0.2% with an average computation time of 0.4 seconds. A comparison between the ACS and a state-of-the-art technique showed that the performance of this state-of-the-art technique is 6.2% worse. The impact of not accounting for time-dependency on the proposed network means a loss in objective function of at least 11.4% on average. Finally, there is a significant difference (on average 35.6%) between the structure of the solution sequence of a time-dependent solution and a solution ignoring the time-dependency of the arcs. These results also showed that some basic recourse actions which are often used in practice to create a kind of "buffer" for the time-dependent disturbance do not pay off.

The orienteering problem with time-dependent stochastic travel times and time windows (TD-OPSWTW) is the third problem considered in this dissertation. The solution method consists of a stochastic version of an ant colony optimization (SACS) specifically developed for the TD-OPSWTW. The SACS uses three local search moves: swap, insert and replace. The latter two are equipped with a local evaluation metric which speeds up the process. The combination of time-dependency, time windows and stochastic travel times severely complicates the calculation of departure and arrival time distributions. Three prominent complications were defined in this dissertation and the way they were facilitated by the proposed estimation algorithm is explained. These complications and estimation method will also be useful for other vehicle routing problems with time windows and time-dependent stochastic travel times.

The proposed solution method is evaluated based on solution quality and computational performance on altered TD-OPTW problem instances. Promising results were obtained in a reasonable amount of computation time. A first comparison is made by comparing the results of the SACS with optimal solutions for the regular OP with time windows, found using an exact solution method, that are evaluated in a stochastic environment. The solutions proposed by the SACS are on average 23.8% better than executing the optimal solution sequence in a time-dependent stochastic environment. Secondly, the solutions generated by the ACS

for TD-OPTW, using different percentiles of the travel time distribution as input, were also evaluated in a stochastic environment and compared to the solutions of the SACS. The SACS performs better than the ACS, independent of the used travel time percentile, as it takes into account the stochastic nature of this problem. However, the SACS also requires a greater computation time. Thirdly, when solving the certainty equivalence problem using the ACS method for the TD-OPTW, the SACS improves solutions by 1.3% on average. When solving the certainty equivalence problem the mean of the stochastic travel time is taken as travel time estimate. The fourth observation is the solution sequence being on average 28.3% different than the proposed solution by the state-of-the-art algorithm. An surprising practical insight is that a probability of a late arrival at a customer can lead to either an increase or decrease of the standard deviation of the departure time. When it is 100% certain that a customer is visited on time, its standard deviation of the departure time is equal to the standard deviation of the arrival time. The probability of arriving early at a customer decreases the standard deviation of its departure time. The effect of a possible late arrival depends on the ratio of the standard deviation of the arrival time and the length of the service time. In general, at the beginning of the solution sequence, the service time is likely to be higher than the standard deviation of the arrival time as the service time comprises usually several minutes while the standard deviation of the arrival time is equal to zero at that point. However, the standard deviation of the arrival time tends to increase towards the end of the solution sequence. In short, a probability of a late arrival in the beginning of the solution sequence increases the standard deviation of the arrival time whereas a possible late arrival at the end of the solution sequence might decrease the standard deviation of the arrival time. The combination of these results should motivate vehicle trip planners to consider the time-dependent and stochastic nature of the travel times in the future.

The acquired insights and solution concepts are also applied on a practical case. For this application, two problem instances where a (different) logistics company has a fleet of vehicles, located at a start depot. These vehicles have a limited capacity in volume and weight and the assigned drivers need to take a break during a certain time window. This break needs to be taken on arrival at a customer. To model the additional practical requirements being different time windows per day, multiple vehicles, the inclusion of a lunch break and two capacity limitations on the vehicles, additional constraints were added to the mathematical formulation of the TD-OPTW. The new problem is called the multi-constrained TD-TOPTW. Subsequently, the solution method originally developed for the TD-OPTW has been adapted as some of these practical requirements drastically change the way a solution can be evaluated and altered by the local search moves. The results of the adapted algorithm are compared to solutions, originating from a commercial VRP solver, that do not take time-dependent travel times into account. These so-

lution are evaluated in a time-dependent context which simulates the performance of a possible execution in real-life. For both cases the proposed solution method outperforms the evaluated solutions within a reasonable computation time. So although time-dependency forms a burden for the computation time, especially when breaks need to be planned in a route, it still pays off to take time-dependency into account during the optimization process.

7.2 Contributions

To have a better overview of the contributions of this dissertation to the research field, these contributions are summarized here:

1. Reviewing the literature on the class of vehicle routing problems and orienteering problems in particular with time-dependent (stochastic) travel times and time windows.
2. Mathematical formulation of the TD-OP, TD-OPTW, TD-OPSWTW and the multi-constrained TD-TOPTW.
3. Creation of benchmark instances with a known optimal solution for the TD-OP based on the original OP instances. Creation of realistic instances for the TD-OPTW and TD-OPSWTW based on the road network of the Benelux. In the case of the TD-OPTW, optimal solutions are available. All instances are made publicly available.
4. Developing multiple ant colony system variants to handle the medium to large scale instances of the TD-OP, TD-OPTW and TD-OPSWTW within a short computation time. Efficient pre-processing methods were also developed to create the required travel time information.
5. Analysis and comparison of the results of the proposed solution methods.
6. Demonstrating that the presented problems and solution methods can be applied in practice by showcasing two case studies. Moreover, these case studies also showed the negative impact on the solution quality when time-dependency was not taken into account.

Introducing and solving the TD-OP and its variants such as the TD-OPTW and TD-OPSWTW is certainly an added value to the literature of the field of routing problems. It is a challenging optimization problem and practically relevant. From our experience, the TD-OP inherits characteristics of the OP but due to the time-dependency an even greater effort is required when solving the instances. Moreover, in any proposed algorithm dealing with this problem, one should consider that in looking for a good quality solution there is a constant need to update

the solution. A small change to the solution at one specific part requires that the arrival times for visits at customers later on are also updated. This requirement makes for a cumbersome solution procedures if not well designed. Firstly, on execution of the local search moves, preselecting interesting changes based on time-independent travel times is a must. Subsequently, when no local evaluation metric is present a full evaluation of the solution is always necessary to detect if the change means an improvement to the objective function. However, evaluating the effect of a move first locally, and if positive globally, decreases the required computation time.

As stated in the results sections, an effective algorithm designed for the time-independent variants of the OP, which does not consider the time-dependency is not expected to efficiently solve orienteering problems with time-dependent travel times, even with minor modifications.

7.3 Future research

Further research could focus on developing exact solution methods that are able to solve medium sized TD-OP and TD-OPTW within a reasonable amount of time. Advanced cuts will need to be developed and applied in order to successfully complete this line of research. Recently, the pulse framework was able to solve large OPTW problem instances within a reasonable amount of computation time. Therefore this framework offers interesting opportunities for future research on the time-dependent variant.

Secondly, in this dissertation, we ignored the fact that customers not served today must be served in the near future. Including this consideration would create an interesting extension and could have a big impact on which customers are selected on a given day.

Thirdly, concerning the multi-constrained TD-TOPTW the current modelling of the driving hours regulation is too simplistic. Recall that in this practical application a break needed to be planned between a predetermined start and end time. The duration of the break was equal to 45 minutes and the start time was in one of the instances equal to the departure time at the start depot plus 4.5 hours. These numbers referred to the legislation for truck drivers in Europe. However, this legislation is more complex as regulation (EC) 561/2006 requests the following requirements for each truck driver:

1. A period between two breaks of at least 45 minutes is called a driving period. The accumulated driving time in a driving period may not exceed 4.5 hours.
2. The break that ends a driving period may be reduced to 30 minutes if an additional break of at least 15 minutes is taken anywhere during that driving

period. The driving hours regulations do not allow service times at customers to be considered as break time.

3. The total accumulated driving time may not exceed 9 hours.
4. The total accumulated duty time may not exceed 13 hours.

Currently, we only partially modelled the first item in the problem formulation of the multi-constrained TD-TOPTW. Taking this European regulation into account would be an interesting future research opportunity.

Furthermore, the experience and the knowledge obtained from this study can be used as a pilot to develop new algorithms for the more general class of vehicle routing problems with time-dependent travel times.

References

- [1] Abbaspour, A., Samadzadegan, F.. Time-dependent personal tour planning and scheduling in metropolises. *Expert Systems with Applications* 2011;38:12439–12452.
- [2] Aghezzaf, E.H., Zhong, Y., Raa, B., Mateo, M.. Analysis of the single-vehicle cyclic inventory routing problem. *International Journal of Systems Science* 2012;43(11):2040–2049.
- [3] Ai, T.J., Pribadi, J.S., Ariyon, V.. Solving the Team Orienteering Problem with Particle Swarm Optimization. *Industrial Engineering & Management Systems* 2013;12:198–206.
- [4] Ando, N., Taniguchi, E.. Travel time reliability in vehicle routing and scheduling with time windows. *Networks & Spatial Economics* 2006;6(3-4):293–311.
- [5] Angelelli, E., Archetti, C., Vindigni, M.. The Clustered Orienteering Problem. *European Journal of Operational Research* 2014;238(2):404–414.
- [6] de Aragão, M.V.S.P., Barboza, F.H., de Freitas, V.E.U.. Team orienteering problem: Formulations and branch-cut and price. *Monografias em Ciência da Computação* 2010;13/10.
- [7] Archetti, C., Bianchessi, N., Speranza, M.G.. The capacitated team orienteering problem with incomplete service. *Optimization letters* 2013;7(7, SI):1405–1417.
- [8] Archetti, C., Bianchessi, N., Speranza, M.G., Hertz, A.. The Split Delivery Capacitated Team Orienteering Problem. *Networks* 2014;63(1):16–33.
- [9] Archetti, C., Feillet, D., Hertz, A., Speranza, M.G.. The capacitated team orienteering and profitable tour problems. *Journal of the Operational Research Society* 2009;60(6):831–842.
- [10] Archetti, C., Hertz, A., Speranza, M.G.. Metaheuristics for the team orienteering problem. *Journal of Heuristics* 2007;13(1):49–76.

- [11] Arkin, E., Mitchell, J., Narasimhan, G.. Resource-constrained geometric network optimisation. In: Proceedings of the 14th ACM Symposium on Computational Geometry. New York: ACM; 1998. p. 307–316.
- [12] Balseiro, S., Loisea, I., Ramonet, J.. An ant colony algorithm hybridized with insertion heuristic for the time dependent vehicle routing problem with time windows. *Computers & Operations Research* 2011;38(6):954–966.
- [13] Barros, A., Evers, L.. The stochastic scouting problem. In: Proceedings of the Congreso Latino-Iberoamericano de Investigacion Operative - Simposio Brasileiro de Pesquisa Operacional CLAIO-SBPO. 2012. .
- [14] Be-Mobile, . Be-mobile: Floating vehicle data. <http://www.be-mobile.be>; 2014.
- [15] Berry, D.S., Belmont, D.M.. Distribution of vehicle speeds and travel times. In: Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability. Berkeley, Calif.: University of California Press; 1951. p. 589–602.
- [16] Bickel, P., Lake, L.W., Lehman, J.. Discretization, simulation, and swan-son’s (inaccurate) mean. *SPE Economics & Management* 2011;3:128–140.
- [17] Blum, A., Chawla, S., Karger, D.R., Lane, T., Meyerson, A., Minkoff, M.. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM Journal of Computing* 2007;37(2):653–670.
- [18] Boyce, D.E., Nie, Y., Bar-Gera, H., Liu, Y., Hu, Y.. Field Test of a Method for Finding Consistent Route Flows and Multiple-Class Link Flows in Road Traffic Assignments. Technical Report; Transportation Center, McCormick School of Engineering and Applied Science; 2010.
- [19] Bullnheimer, B., Hartl, R.F., Strauss, C.. Applying the ant system to the vehicle routing problem. In: *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. 1999. p. 285–296.
- [20] Butt, S., Cavalier, T.. A heuristic for the multiple tour maximum collection problem. *Computers & Operations Research* 1994;21:101–111.
- [21] Campbell, A.M., Gendreau, M., Thomas, B.W.. The orienteering problem with stochastic travel and service times. *Annals of Operations Research* 2011;186(1):61–81.
- [22] Campos, V., Marti, R., Sanchez-Oro, J., Duarte, A.. GRASP with path relinking for the orienteering problem. *Journal of the Operational Research Society* 2014;65(12):1800–1813.

- [23] Chao, I., Golden, B., Wasil, E.. Theory and methodology - the team orienteering problem. *European Journal of Operational Research* 1996;88:464–474.
- [24] Chao, I., Golden, B., Wasil, E.. Theory and methodology a fast and effective heuristic for the orienteering problem. *European Journal of Operational Research* 1996;88:475–489.
- [25] Chekuri, C., Korula, N., Pal, M.. Improved Algorithms for Orienteering and Related Problems. *ACM Transactions on Algorithms* 2012;8(3).
- [26] Chen, H., Hsueh, C., Chang, M.. The real-time time-dependent vehicle routing problem. *Transportation Research Part E* 2006;42:383–408.
- [27] Clark, S., Watling, D.. Modelling network travel time reliability under stochastic demand. *Transportation Research Part B: Methodological* 2005;39(2):119 – 140.
- [28] Corthout, R., Himpe, W., Viti, F., Frederix, R., Tampère, C.M.. Improving the efficiency of repeated dynamic network loading through marginal simulation. *Transportation Research Part C: Emerging Technologies* 2014;41(0):90 – 109.
- [29] Cura, T.. An artificial bee colony algorithm approach for the team orienteering problem with time windows. *Computers and Industrial Engineering* 2014;74:270–290.
- [30] Dabia, S., Ropke, S., van Woensel, T., De Kok, T.. Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation Science* 2013;47(3):380–396.
- [31] Dang, D.C., Guibadj, R.N., Moukrim, A.. An effective PSO-inspired algorithm for the team orienteering problem. *European Journal of Operations Research* 2013;229(2):332–344.
- [32] Demeyer, S., Audenaert, P., Pickavet, M., Demeester, P.. Dynamic and stochastic routing for multimodal transportation systems. *IET Intelligent Transport Systems* 2014;8(2):112–123.
- [33] Dijkstra, E.. A note on two problems in connexion with graphs. *Numerische Mathematik* 1959;1(1):269–271.
- [34] Divsalar, A., Vansteenwegen, P., Cattrysse, D.. A variable neighborhood search method for the orienteering problem with hotel selection. *International journal production economics* 2013;145:150–160.

- [35] Divsalar, A., Vansteenwegen, P., Sorensen, K., Cattrysse, D.. A memetic algorithm for the orienteering problem with hotel selection. *European Journal of Operations Research* 2014;237(1):29–49.
- [36] Donati, A., Montemanni, R., Casagrande, N., Rizzoli, A., Gambardella, L.. Time dependent vehicle routing problem with a multi ant colony system. *European Journal of Operational Research* 2008;185:1174–1191.
- [37] Duque, D., Lozano, L., Medaglia, A.L.. Solving the Orienteering Problem with Time Windows via the Pulse Framework. *Computers and Operations Research* 2015;54:168–176.
- [38] Erdogan, G., Laporte, G.. The orienteering problem with variable profits. *Networks* 2013;61(2, SI):104–116.
- [39] Evers, L., Barros, A.I., Monsuur, H., Wagelmans, A.. Online stochastic UAV mission planning with time windows and time-sensitive targets. *European Journal of Operational Research* 2014;238(1):348 – 362.
- [40] Evers, L., Dollevoet, T., Barros, A., Monsuur, H.. Robust UAV mission planning. *Annals of Operations Research* 2014;222:293–315.
- [41] Evers, L., Glorie, K., van der Ster, S., Barros, A., Monsuur, H.. The Orienteering Problem under Uncertainty Stochastic Programming and Robust Optimization compared. Technical Report Econometric Institute Report EI 2012-21; Erasmus University, Econometric Institute; 2012.
- [42] Evers, L., Glorie, K., van der Ster, S., Barros, A.I., Monsuur, H.. A two-stage approach to the orienteering problem with stochastic weights. *Computers & Operations Research* 2014;43(0):248 – 260.
- [43] Fan, Y., Kalaba, R., Moore, J.. Arriving on time. *Journal of Optimization Theory and Applications* 2005;127(3):497–513.
- [44] Feillet, D., Dejax, P., Gendreau, M.. Traveling salesman problems with profits. *Transportation Science* 2005;39(2):188–205.
- [45] Figliozzi, M.A.. The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics. *Transportation Research Part E: Logistics and Transportation Review* 2012;48(3):616 – 636.
- [46] Fischetti, M., Salazar, J., Toth, P.. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing* 1998;10:133–148.

- [47] Fleischmann, B., Gietz, M., Gnutzmann, S.. Time-varying travel times in vehicle routing. *Transportation Science* 2004;38:160–173.
- [48] Fomin, F., Lingas, A.. Approximation algorithms for time-dependent orienteering. *Information Processing Letters* 2002;83:57–62.
- [49] Forgy, E.W.. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics* 1965;21:768–769.
- [50] Franceschetti, A., Honhon, D., Woensel, T.V., Bekta, T., Laporte, G.. The time-dependent pollution-routing problem. *Transportation Research Part B: Methodological* 2013;56:265 – 293.
- [51] Fu, L., Rilett, L.. Expected shortest paths in dynamic and stochastic traffic networks. *Transportation Research Part B: Methodological* 1998;32(7):499–516.
- [52] Garcia, A., Arbelaiz, O., Vansteenwegen, P., Souffriau, W., Linaz, M.. Hybrid Artificial Intelligence Systems; Springer Verlag Berlin Heidelberg; volume Volume 6077/2010 of *Lecture Notes in Computer Science*. p. 151–158.
- [53] Garcia, A., Vansteenwegen, P., Arbelaiz, O., Souffriau, W., Linaza, M.. Integrating Public Transportation in Personalised Electronic Tourist Guides. *Computers & Operations Research* 2013;40:758–774.
- [54] Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G.. A survey on algorithmic approaches for solving tourist trip design problems. *Journal of Heuristics* 2014;20(3):291–328.
- [55] Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., Vathis, N.. Efficient heuristics for the time dependent team orienteering problem with time windows. In: Gupta, P., Zaroliagis, C., editors. *Applied Algorithms*. Springer International Publishing; volume 8321 of *Lecture Notes in Computer Science*; 2014. p. 152–163.
- [56] Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., Vathis, N.. Heuristics for the time dependent team orienteering problem: Application to tourist route planning. *Computers & Operations Research* 2015;62(0):36 – 50.
- [57] Geem, Z., Tseng, C.L., Park, Y.. Harmony search for generalized orienteering problem: Best touring in china. In: Wang, L., Chen, K., Ong, Y., editors. *Advances in Natural Computation*. Springer Berlin Heidelberg; volume 3612 of *Lecture Notes in Computer Science*; 2005. p. 741–750.

- [58] Gendreau, M., Ghiani, G., Guerriero, E.. Time-dependent routing problems: A review. *Computers & Operations Research* 2015;64:189 – 197.
- [59] Gendreau, M., Laporte, G., Semet, F.. A branch-and-cut algorithm for the undirected selective travelling salesman problem. *Networks* 1998;32:263–273.
- [60] Gendreau, M., Laporte, G., Semet, F.. A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research* 1998;106 (2-3):539–545.
- [61] Golden, B., Levy, L., Vohra, R.. The orienteering problem. *Naval Research Logistics* 1987;34:307–318.
- [62] Haghani, A., Jung, S.. A dynamic vehicle routing problem with time-dependent travel times. *Computers & Operations Research* 2005;32:2959–2986.
- [63] Hashimoto, H., Yagiurab, M., Ibaraki, T.. An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *Discrete Optimization* 2008;5:434–456.
- [64] He, R.R., Liu, H.X., Kornhauser, A.L., Ran, B.. Temporal and Spatial Variability of Travel Time. Technical Report; Center for Traffic Simulation Studies. UC Irvine: Center for Traffic Simulation Studies.; 2002.
- [65] Hu, Q., Lim, A.. An iterative three-component heuristic for the team orienteering problem with time windows. *European Journal of Operations Research* 2014;232(2):276–286.
- [66] Ichoua, S., Gendreau, M., Potvin, J.Y.. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research* 2003;144:379–396.
- [67] Ilhan, T., Iravani, S.M.R., Daskin, M.S.. The orienteering problem with stochastic profits. *IIE Transactions* 2008;40(4):406–421.
- [68] Inrix, . Traffic scorecard. <http://inrix.com/scorecard/>; 2015. Accessed: 2015-05-19.
- [69] Jensen, P.A.. Operations research models and methods. 2004.
- [70] Kantor, M., Rosenwein, M.. The orienteering problem with time windows. *Journal of Operational Research Society* 1992;43 (6):629–635.

- [71] Kaparias, I., Bell, M.G.H., Belzner, H.. A New Measure of Travel Time Reliability for In-Vehicle Navigation Systems. *Journal of Intelligent Transportation Systems* 2008;12(4):202–211.
- [72] Karbowska-Chilinska, J., Zabielski, P. Genetic Algorithm with Path Re-linking for the Orienteering Problem with Time Windows. *Fundamenta Informaticae* 2014;135(4):419–431.
- [73] Kataoka, S., Morito, S.. An algorithm for the single constraint maximum collection problem. *Journal of the Operational Research Society of Japan* 1988;31:515–530.
- [74] Ke, L., Archetti, C., Feng, Z.. Ants can solve the team orienteering problem. *Computers and Industrial Engineering* 2008;54:648–665.
- [75] Kek, A.G.H., Cheu, R.L., Meng, Q.. Distance-constrained capacitated vehicle routing problems with flexible assignment of start and end depots. *Mathematical and Computer Modelling* 2008;47(1-2):140–152.
- [76] Kharoufeh, J.P., Gautam, N.. Deriving link travel-time distributions via stochastic speed processes. *Transportation Science* 2004;38(1):97–106.
- [77] Kim, B.I., Li, H., Johnson, A.L.. An augmented large neighborhood search method for solving the team orienteering problem. *Expert Systems with Applications* 2013;40(8):3065–3072.
- [78] Kok, A.L., Hans, E.W., Schutten, J.M.J.. Vehicle routing under time-dependent travel times: The impact of congestion avoidance. *Computers & Operations Research* 2012;39 (5):910–918.
- [79] Kok, A.L., Hans, E.W., Schutten, J.M.J., Zijm, W.H.M.. A dynamic programming heuristic for vehicle routing with time-dependent travel times and required breaks. *Flexible Service and Manufacturing Journal* 2010;22(1-2, SI):83–108.
- [80] Kritzingner, S., Tricoire, F., Doerner, K., Hartl, R.. Variable neighborhood search for the time-dependent vehicle routing problem with soft time windows. In: Coello, C., editor. *Learning and Intelligent Optimization*. Springer Berlin Heidelberg; volume 6683 of *Lecture Notes in Computer Science*; 2011. p. 61–75.
- [81] Kuo, Y.. Using simulated annealing to minimize fuel consumption for the time-dependent vehicle routing problem. *Computers & Industrial Engineering* 2010;59(1):157 – 165.

- [82] Kwon, J., Coifman, B., Bickel, P.. Day-to-day travel-time trends and travel-time prediction from loop-detector data. *Transportation Research Record: Journal of the Transportation Research Board* 2000;1717:120–129.
- [83] Labadie, N., Mansini, R., Melechovský, J., Calvo, R.W.. The team orienteering problem with time windows: An lp-based granular variable neighborhood search. *European Journal of Operational Research* 2012;220(1):15 – 27.
- [84] Labadie, N., Melechovsky, J., Calvo, R.W.. Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. *Journal of Heuristics* 2012;17(6):729–753.
- [85] Laporte, G., Martello, S.. The selective travelling salesman problem. *Discrete applied Mathematics* 1990;26:193–207.
- [86] Lau, H.C., Yeoh, W., Varakantham, P., Nguyen, D.T., Chen, H.. Dynamic stochastic orienteering problems for risk-aware applications. *CoRR* 2012;abs/1210.4874:1–11.
- [87] Lecluyse, C., Sörensen, K., Peremans, H.. A network-consistent time-dependent travel time layer for routing optimization problems. *European Journal of Operational Research* 2013;226:395–413.
- [88] Lecluyse, C., Van Woensel, T., Peremans, H.. Vehicle routing with stochastic time-dependent travel times. *4OR: Quarterly journal of Operational Research* 2009;7:363–377.
- [89] Li, J.. Model and algorithm for time-dependent team orienteering problem. In: Lin, S., Huang, X., editors. *Advanced Research on Computer Education, Simulation and Modeling*. Springer Berlin Heidelberg; volume 175 of *Communications in Computer and Information Science*; 2011. p. 1–7.
- [90] Li, J., Wu, Q., Li, X., Zhu, D.. Study on the time-dependent orienteering problem. In: *International Conference on E-Product E-Service and E-Entertainment (ICEEE)*. 2010. p. 1 – 4.
- [91] Li, X., Tian, P., Leung, S.. Vehicle routing problems with time windows and stochastic travel and service times: Models and algorithm. *International Journal of Production economics* 2010;125:137–145.
- [92] Liang, Y.C., Kulturel-Konak, S., Smith, A.. Meta heuristics for the orienteering problem. In: *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on Computational Intelligence*. volume 1; 2002. p. 384–389.

- [93] Lin, S.W.. Solving the team orienteering problem using effective multi-start simulated annealing. *Applied Soft Computing* 2013;13(2):1064–1073.
- [94] Lin, S.W., Yu, V.F.. A simulated annealing heuristic for the team orienteering problem with time windows. *European Journal of Operations Research* 2012;217(1):94–107.
- [95] Liu, X., Chien, S., Kim, K.. Evaluation of floating car technologies for travel time estimation. *Journal of Modern Transportation* 2012;20(1):49–56.
- [96] Luo, Z., Cheang, B., Lim, A., Zhu, W.. An adaptive ejection pool with toggle-rule diversification approach for the capacitated team orienteering problem. *European Journal of Operations Research* 2013;229(3):673–682.
- [97] Maden, W., Eglese, R., Black, D.. Vehicle routing and scheduling with time-varying data: A case study. *Journal of the Operational Research Society* 2010;61(3, SI):515–522.
- [98] Malandraki, C., Daskin, M.. Time dependent vehicle routing problems: formulations, properties and heuristic algorithms. *Transportation Science* 1992;26:185–200.
- [99] Miller, C., Tucker, A., Zemlin, R.. Integer programming formulations and travelling salesman problems. *Journal of the ACM* 1960;7:326–329.
- [100] Montemanni, R., Gambardella, L.. An ant colony system for team orienteering problems with time windows. *Foundation Of Computing And Decision Sciences* 2009;.
- [101] Mufalli, F., Batta, R., Nagi, R.. Simultaneous sensor selection and routing of unmanned aerial vehicles for complex mission plans. *Computers & Operations Research* 2012;39(11):2787 – 2799.
- [102] Nagarajan, V., Ravi, R.. The directed orienteering problem. *Algorithmica* 2011;60(4):1017–1030.
- [103] Nguyen, P.K., Crainic, T.G., Toulouse, M.. A tabu search for Time-dependent Multi-zone Multi-trip Vehicle Routing Problem with Time Windows. *European Journal of Operational Research* 2013;231(1):43–56.
- [104] Noland, R.B., Polak, J.W.. Travel time variability: A review of theoretical and empirical issues. *Transport Reviews* 2002;22(1):39–54.
- [105] Osvald, A., Stirn, L.Z.. A vehicle routing algorithm for the distribution of fresh vegetables and similar perishable food. *Journal of Food Engineering* 2008;85(2):285 – 295.

- [106] Papapanagiotou, V., Montemanni, R., Gambardella, L.. Objective function evaluation methods for the orienteering problem with stochastic travel and service times. *Journal of applied Operational research* 2014;6(1):16–29.
- [107] Papapanagiotou, V., Weyland, D., Montemanni, R., Gambardella, L.. A sampling-based approximation of the objective function of the orienteering problem with stochastic travel and service times. *Lecture Notes in Management Science* 2013;5:143–152.
- [108] Pietz, J., Royset, J.O.. Generalized orienteering problem with resource dependent rewards. *Naval Research Logistics* 2013;60(4):294–312.
- [109] Potvin, J., Xu, Y., Benyahia, I.. Vehicle routing and scheduling with dynamic travel times. *Computers & Operations research* 2006;33:1129–1137.
- [110] Rahmani, M., Jenelius, E., Koutsopoulos, H.N.. Non-parametric estimation of route travel time distributions from low-frequency floating car data. *Transportation Research Part C: Emerging Technologies* 2015;58(B, SI):343–362.
- [111] Ramesh, R., Brown, K.. An efficient four-phase heuristic for the generalized orienteering problem. *Computers & Operations Research* 1991;18:151–165.
- [112] Righini, G., Salani, M.. Dynamic programming for the orienteering problem with time windows. *Technical Report; Dipartimento di Tecnologie dell'Informazione*; 2006.
- [113] Righini, G., Salani, M.. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks* 2008;51(3):155–170.
- [114] Righini, G., Salani, M.. Incremental state space relaxation strategies and initialization heuristics for solving the Orienteering Problem with Time Windows with dynamic programming. *Computers and Operations Research* 2009;36(4):1191–1203.
- [115] Royset, J.O., Reber, D.N.. Optimized routing of unmanned aerial systems for the interdiction of improvised explosive devices. *Military Operations Research* 2009;14(4):5–19.
- [116] Russell, R.A., Urban, T.L.. Vehicle routing with soft time windows and erlang travel times. *The Journal of the Operational Research Society* 2008;59(9):1220–1228.

- [117] Salazar-Aguilar, M., Langevin, A., Laporte, G.. The multi-district team orienteering problem. *Computers and Operations Research* 2014;41:76–82.
- [118] Schilde, M., Doerner, K., Hartl, R., Kiechle, G.. Metaheuristics for the biobjective orienteering problem. *Swarm Intelligence* 2009;3:179–201.
- [119] Setak, M., Habibi, M., Karimi, H., Abedzadeh, M.. A time-dependent vehicle routing problem in multigraph with fifo property. *Journal of Manufacturing Systems* 2015;35(0):37 – 45.
- [120] Sevkli, Z., Sevilgen, F.E.. Variable neighborhood search for the orienteering problem. In: Levi, A and Savas, E and Yenigun, H and Balciory, S and Saygin, Y, , editor. *Computer and Information Sciences - IS-CIS 2006, Proceedings*. Sabanci Univ, Fac Engn & Nat Sci; Sci & Technol Res Council Turkey; Sabanci Univ; Inst Elect & Elect Engineers, Turkey Sect; IFIP; volume 4263 of *Lecture Notes in Computer Science*; 2006. p. 134–143. 21st International Symposium on Computer and Information Sciences (ISCIS 2006), Istanbul, TURKEY, NOV 01-03, 2006.
- [121] Soler, D., Albiach, J., Martinez, E.. A way to optimally solve a time-dependent vehicle routing problem with time windows. *Operations Research Letters* 2009;37:37–42.
- [122] Sörensen, K., Sevaux, M., Schittekat, P.. Adaptive and Multilevel Metaheuristics; London: Springer; volume 136 of *Lecture Notes in Economics and Mathematical Systems*. p. 239–253.
- [123] Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., Van Oudheusden, D.. Path Relinking Approach for the Team Orienteering Problem. *Computers & Operations Research* 2010;37:1853–1859.
- [124] Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., Van Oudheusden, D.. The planning of Cycle Trips in the Province of East Flanders. *OMEGA: International Journal of Management Science* 2011;39:209–213.
- [125] Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., Van Oudheusden, D.. The multiconstraint team orienteering problem with multiple time windows. *Transportation Science* 2013;47 (1):53–63.
- [126] Souffriau, W., Vansteenwegen, P., Vertommen, J., Vanden Berghe, G., Van Oudheusden, D.. A personalised tourist trip design algorithm for mobile tourist guides. *Applied Artificial Intelligence* 2008;22:964–985.
- [127] Steven, L., Filip, V.. Het Belgische verkeer in cijfers. Technical Report; Transport & Mobility Leuven; 2004.

- [128] Sun, S., Duan, Z., Yang, D.. Urban Freight Management with Stochastic Time-Dependent Travel Times and Application to Large-Scale Transportation Networks. *Discrete Dynamics in Nature and Society* 2015;;10.
- [129] Taş, D., Dellaert, N., van Woensel, T., de Kok, T.. Vehicle routing problem with stochastic travel times including soft time windows and service costs. *Computers & Operations Research* 2013;40(1):214 – 224.
- [130] Taş, D., Dellaert, N., van Woensel, T., de Kok, T.. The time-dependent vehicle routing problem with soft time windows and stochastic travel times. *Transportation Research Part C: Emerging Technologies* 2014;48(0):66 – 83.
- [131] Taş, D., Gendreau, M., Dellaert, N., van Woensel, T., de Kok, A.G.. Vehicle routing with soft time windows and stochastic travel times: A column generation and branch-and-price solution approach. *European Journal of Operations Research* 2014;236(3, SI):789–799.
- [132] Tae, H., Kim, B.I.. A Branch-and-Price Approach for the Team Orienteering Problem with Time Windows. *International Journal of Industrial Engineering- Theory Applications and Practice* 2015;22(2):243–251.
- [133] Talbi, E.G.. *Metaheuristics: From design to implementation*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2009.
- [134] Tang, H., Miller-Hooks, E.. Algorithms for a stochastic selective traveling salesperson problem. *The Journal of the Operational Research Society* 2005;56(4):439–452.
- [135] Taniguchi, E., Thompson, R., Yamada, T., van Duin, J.. *City Logistics: Network Modelling and Intelligent Transport Systems*. Pergamon, 2001.
- [136] Tarantilis, C.D., Stavropoulou, F., Repoussis, P.P.. The Capacitated Team Orienteering Problem: A Bi-level Filter-and-Fan method. *European Journal of Operations Research* 2013;224(1):65–78.
- [137] Tasgetiren, M.F.. A genetic algorithm with an adaptive penalty function for the orienteering problem. *Journal of Economic & Social Research* 2001;4(2).
- [138] Teng, S.Y., Ong, H.L., Huang, H.C.. An integer l-shaped algorithm for time-constrained traveling salesman problem with stochastic travel and service times. *Asia-Pacific Journal of Operational Research* 2004;21(02):241–257.

- [139] Thomadsen, T., Stidsen, T.. The Quadratic Selective Travelling Salesman Problem. Technical Report; Informatics and Mathematical Modelling, Technical University of Denmark, DTU; Richard Petersens Plads, Building 305, DK-2800 Kgs. Lyngby; 2003.
- [140] Thomas, B.W., Manni, E.. Scheduled penalty Variable Neighborhood Search. *Computers and Operations Research* 2014;52(B, SI):170–180.
- [141] Tricoire, F., Romauch, M., Doerner, K.F., Hartl, R.F.. Heuristics for the multi-period orienteering problem with multiple time windows. *Computers and Operations Research* 2010;37(2):351–367.
- [142] Tsiligrirides, T.. Heuristic methods applied to orienteering. *Journal of the Operational Research Society* 1984;35 (9):797–809.
- [143] Van Woensel, T., Kerbache, L., Peremans, H., Vandaele, N.. Vehicle routing with dynamic travel times: A queueing approach. *European Journal of Operational Research* 2008;186:990–1007.
- [144] Vansteenwegen, P.. Planning in tourism and public transportations - attraction selection by means of a personalised electronic tourist guide and train transfer scheduling. Ph.D. thesis; Katholieke Universiteit Leuven, Centre for Industrial Management, Belgium; 2008.
- [145] Vansteenwegen, P., Souffriau, W., Van Oudheusden, D.. The orienteering problem: a survey. *European Journal of Operational Research* 2011;209:1–10.
- [146] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.. A Guided Local Search Metaheuristic for the Team Orienteering Problem. *European Journal of Operational Research* 2009;196(1):118–127.
- [147] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.. Iterated Local Search for the Team Orienteering Problem with Time Windows. *Computers & Operations Research* 2009;36:3281–3290.
- [148] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.. Metaheuristics for tourist trip planning. In: Sörensen, K., Sevaux, M., Habenicht, W., Geiger, M.J., editors. *Metaheuristics in the Service Industry*. Springer Berlin Heidelberg; volume 624 of *Lecture Notes in Economics and Mathematical Systems*; 2009. p. 15–31.
- [149] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.. The City Trip Planner: A Tourist Expert System. *Expert Systems with Applications* 2011;38:6540–6546.

- [150] Vansteenwegen, P., Van Oudheusden, D.. The mobile tourist guide: An OR opportunity. *OR Insights* 2007;20 (3):21–27.
- [151] Verbeeck, C., Sörensen, K., Aghezzaf, E.H., Vansteenwegen, P.. A fast solution method for the time-dependent orienteering problem. *European Journal of Operational Research* 2014;236 (2):419–432.
- [152] Verbeeck, C., Vansteenwegen, P., Aghezzaf, E.H.. An extension of the arc orienteering problem and its application to cycle trip planning. *Transportation Research Part E* 2014;68:64–78.
- [153] Wang, Q., Sun, X., Golden, B., Jia, J.. Using artificial neural networks to solve the orienteering problem. *Annals of Operations Research* 1995;61:111–120.
- [154] Wang, X., Golden, B., Gulczynski, D.. A worst-case analysis for the split delivery capacitated team orienteering problem with minimum delivery amounts. *Optimization letters* 2014;8(8):2349–2356.
- [155] Wang, X., Golden, B., Wasil, E.. Using a genetic algorithm to solve the generalized orienteering problem. In: Golden, B., Raghavan, S., Wasil, E., editors. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer US; volume 43 of *Operations Research/Computer Science Interfaces*; 2008. p. 263–274.
- [156] Wen, L., Eglese, R.. Minimum cost VRP with time-dependent speed data and congestion charge. *Computers & Operations Research* 2015;56(0):41 – 50.
- [157] Yildirimoglu, M., Geroliminis, N.. Experienced travel time prediction for congested freeways. *Transportation Research Part B: Methodological* 2013;53:45 – 63.
- [158] Zhang, S., Ohlmann, J.W., Thomas, B.W.. A priori orienteering with time windows and stochastic wait times at customers. *European Journal of Operational Research* 2014;239(1):70 – 79.
- [159] Zhang, T., Chaovalitwongse, W.A., Zhang, Y.. Integrated Ant Colony and Tabu Search approach for time dependent vehicle routing problems with simultaneous pickup and delivery. *Journal of Combinatorial Optimization* 2014;28(1, SI):288–309.
- [160] Zheng, F., Van Zuylen, H.. Urban link travel time estimation based on sparse probe vehicle data. *Transportation Research Part C: Emerging Technologies* 2013;31:145 – 157.

List of publications

Articles in internationally reviewed academic journals

- Verbeeck, C., Vansteenwegen, P., Aghezzaf, E.-H. (2015). Solving the Stochastic Time-Dependent Orienteering Problem with Time Windows, submitted to *European Journal of Operational Research*
- Verbeeck, C., Van Peteghem, V., Vanhoucke, M., Vansteenwegen, P., Aghezzaf, E.-H. (2015). A Metaheuristic Solution Approach for the Time-constrained Project Scheduling Problem, submitted to *OR Spectrum*
- Verbeeck, C., Aghezzaf, E.-H., Vansteenwegen, P. (2015). A fast solution method for the Time-Dependent Orienteering problem with Time Windows, submitted to *Annals of Operations Research*
- Verbeeck, C., Sörensen, K., Aghezzaf, E.-H., Vansteenwegen, P. (2014). A fast solution method for the time-dependent orienteering problem, *European Journal of Operational Research*, 236 (2), 419-432
- Verbeeck, C., Aghezzaf, E.-H., Vansteenwegen, P. (2014). An extension of the arc orienteering problem and its application to cycle trip planning, *Transportation Research Part E*, 68, 64-78

Papers at international scientific conferences and symposia, published in full in proceedings

- Verbeeck, C., Aghezzaf, E.-H., Vansteenwegen, P. (2015). Solving the stochastic time-dependent orienteering problem with time windows, *6th International Workshop on Freight Transportation and Logistics (Odysseus 2015)*, Ajaccio, France, 31 May-6 June 2015
- Verbeeck, C., Aghezzaf, E.-H., Vansteenwegen, P. (2014). Solving the stochastic time-dependent orienteering problem, *10ème Conférence Internationale Francophone de Modélisation et Simulation (MOSIM '14)*, Nancy, France, 5-7 November 2014
- Van Peteghem, V., Verbeeck, C. (2014) A metaheuristic solution approach for the time-constrained project scheduling problem, *14th International Conference on Project Management and Scheduling (PMS 2014)*, Munich, Germany, 30 March-2 April 2014
- Verbeeck, C., Vansteenwegen, P. (2013), Solution methods for the time-dependent orienteering problem, *Eighth Triennial Symposium on Transportation Analysis (TRISTAN 2013)*, San Pedro de Atacama, Chile, 9-14 June 2013

- Verbeeck, C., Aghezzaf, E.-H., Vansteenwegen, P. (2013). A fast solution method for the time-dependent orienteering problem with time windows, *6th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA-2013)*, Ghent, Belgium, 27-30 August 2013

Meeting abstracts, presented at international scientific conferences and symposia, published or not published in proceedings or journals

- Verbeeck, C., Vansteenwegen, P., Aghezzaf, E.-H. (2014). Solution methods for the cycle trip planning problem, *28th Annual conference of the Belgian Operations Research Society (ORBEL)*, Mons, Belgium, 30-31 January 2014
- Verbeeck, C., Vansteenwegen, P., Aghezzaf, E.-H. (2014). The orienteering problem with time-dependent stochastic travel times, *Third meeting of the EURO Working Group on Vehicle Routing and Logistics Optimization (VeRoLog 2014)*, Oslo, Norway, 22-25 June 2014
- Verbeeck, C., Vansteenwegen, P. (2012), Comparing metaheuristics for the time-dependent orienteering problem, *25th European Conference on Operational Research (EURO)*, Vilnius, Lithuania, 8-11 July 2012
- Verbeeck, C., Vansteenwegen, P. (2012), Congestion avoidance: optimization of vehicle routing planning for the logistics industry, *26th Annual Conference of the Belgian Operations Research Society (ORBEL)*, Brussels, Belgium, 5-14 February 2012
- Verbeeck, C., Vansteenwegen, P. (2012), Metaheuristics for the time-dependent orienteering problem, *First Annual Conference of the EURO Working Group on Vehicle Routing and Logistics Optimization (VeRoLog 2012)*, Bologna, Italy, 18-20 June 2012

Appendix

A.1 Width of departure time distribution spanning multiple time slots

If $D \sim \mathcal{N}(\mu_0, \sigma_0^2)$ represents the departure time and $g > 0$ the start of a time slot that divides the departure time distribution into two parts. Furthermore, let $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ be the travel time corresponding to the first time slot and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ the travel time corresponding to the second time slot. Then, the random variable A represents the arrival time and is equal to the following piecewise function:

$$A = \begin{cases} D + X_1 & \text{if } D \leq g \\ D + X_2 & \text{if } D > g. \end{cases} \quad (\text{A.1})$$

To find the probability density function of A , the following help functions need to be calculated:

- $h_1(z)$ which is the probability density function of $(D + X_1)|(D \leq g)$
- $h_2(z)$ which is the probability density function of $(D + X_2)|(D > g)$

The probability density function of $(D + X_1)|(D \leq g)$ is the joint product of the individual probability density functions or:

$$\phi_{f_1} = \frac{\phi_D}{\mathcal{P}(D \leq g)} * \phi_{x_1} \quad (\text{A.2})$$

Note that $\mathcal{P}$ stands for the probability. This function is inserted in Mathematica as follows:

```
f1 = (1/(Subscript[\[Sigma],0]*Sqrt[2*\[Pi]]))
*Exp[-(d - Subscript[\[Mu],0])^2/
(2 *Subscript[\[Sigma],0]^2)]/(1/2 (1 +
Erf[(g - Subscript[\[Mu],0])/
(Subscript[\[Sigma],0]*Sqrt[2])]))*
1/(Subscript[\[Sigma],1]*Sqrt[2*\[Pi]])*
Exp[-(Subscript[x,1] - Subscript[\[Mu],1])^2/
(2 *Subscript[\[Sigma],1]^2)];
domain[f1] = {{d,-Infinity,Infinity},{Subscript[x,1],
-Infinity,Infinity}} &&
{Element[Subscript[\[Mu],0],Reals],
Element[Subscript[\[Mu],1],Reals],
Subscript[\[Sigma],0] > 0,Subscript[\[Sigma],1] > 0,
g > 0};
```

Afterwards, we apply the following transformation on $f_1(D, X_1) \rightarrow (Z = D + X_1, V = X_1)$. The joint probability density function of (Z, V) is called $g_1(z, v)$. Note that the transformation equation $(Z = D + X_1, V = X_1)$ induces dependency between Z and V . In particular, since $Z = V + D$ and $D < g$, it follows that $Z < V + g$. We execute this transformation in Mathematica using the Transform function and a boole statement.

```
g1 = Transform[{z == d + Subscript[x,1],
v == Subscript[x,1]},f1] Boole[z < v + g]
domain[g1] = {{z, -Infinity, Infinity},
{v, -Infinity,Infinity}} &&
{Element[Subscript[\[Mu],0],Reals],
Element[Subscript[\[Mu],1], Reals],
Subscript[\[Sigma],0] > 0,
Subscript[\[Sigma],1] > 0, g > 0};
```

Finally, h_1 is the marginal probability density function of $Z = D + X_1$. This last procedure is performed in Mathematica using the Marginal function:

```
h1 = Marginal[z, g1]
```

Similarly, h_2 is calculated as follows: The probability density function of $(D + X_2)|(D > g)$ is the joint product of the individual probability density function or:

$$\phi_{f2} = \frac{\phi_D}{\mathcal{P}(D > g)} * \phi_{x_2} \quad (\text{A.3})$$

Afterwards, we apply the following transformation $(D, X_2) \rightarrow (Z = D + X_2, V = X_2)$, the joint probability density function of (Z, V) is called $g_2(z, v)$. h_2 is the marginal probability density function of $Z = D + X_2$. These procedures are done in Mathematica as follows:

```
f2 = (1/(Subscript[\[Sigma],0]*Sqrt[2*\[Pi]]))*
Exp[-(d - Subscript[\[Mu],0])^2/
(2 *Subscript[\[Sigma],0]^2)]]/(1/2
(1 -Erf[(g - Subscript[\[Mu],0])/
(Subscript[\[Sigma],0]*Sqrt[2])]))*1/
(Subscript[\[Sigma],2]*Sqrt[2*\[Pi]]))
*Exp[-(Subscript[x,2] - Subscript[\[Mu],2])^2/(2
*Subscript[\[Sigma],2]^2)];
domain[f2] = {{d, -Infinity, Infinity},
{Subscript[x, 2], -Infinity,Infinity}} &&
{Element[Subscript[\[Mu],0],Reals],
Element[Subscript[\[Mu],2], Reals],
Subscript[\[Sigma],0] > 0,
Subscript[\[Sigma],2] > 0, g > 0};
g2 = Transform[{z == d + Subscript[x,2],
v == Subscript[x,2]},f2] Boole[z > v + g]
domain[g2] = {{z, -Infinity, Infinity},
{v, -Infinity,Infinity}}
&&{Element[Subscript[\[Mu],0], Reals],
Element[Subscript[\[Mu],2], Reals],
Subscript[\[Sigma],0]>0,Subscript[\[Sigma],2]>0,
g > 0};
h2 = Marginal[z, g2]
```

The probability density function of A is now equal to:

$$A = \mathcal{P}(D \leq g) * h_1(z) + \mathcal{P}(D > g) * h_2(z) \quad (\text{A.4})$$

This is calculated using the Prob function in Mathematica as follows:

```
Dep = 1/(Subscript[\[Sigma],0]*Sqrt[2*\[Pi]])*
Exp[-(d - Subscript[\[Mu],0])^2/
(2*Subscript[\[Sigma],0]^2)];
domain[Dep] = {d, -Infinity,
Infinity} && {Element[Subscript[\[Mu],0], Reals],
Subscript[\[Sigma],0] > 0};
prob = Prob[g, Dep]
Arr = h1*prob + (1 - prob)*h2 // Simplify
```

To calculate the mean and the variance let $f_3(d, x_1, x_2)$ denote the joint probability density function of (D, X_1, X_2) which, by independence, is the product of the 3 individual normal probability density functions. Together with the following function z , Mathematica can calculate the mean and standard deviation of A using respectively the Expect and Var function.

```
f3 = (1/(Subscript[\[Sigma],0]*Sqrt[2*\[Pi]]))*
Exp[-(d - Subscript[\[Mu],0])^2/(
2 *Subscript[\[Sigma],0]^2)]]*(1/(
Subscript[\[Sigma],1]*Sqrt[2*\[Pi]]))*
Exp[-(Subscript[x,1] - Subscript[\[Mu],1])^2/(
2 *Subscript[\[Sigma],1]^2)]]*(1/(
Subscript[\[Sigma],2]*Sqrt[2*\[Pi]]))*
Exp[-(Subscript[x,2] - Subscript[\[Mu],2])^2/(
2 *Subscript[\[Sigma],2]^2)]];
domain[f3] = {{d,-Infinity,Infinity},{Subscript[x,1],
-Infinity,Infinity},{Subscript[x,2],
-Infinity,Infinity}} && {Element[Subscript[\[Mu],0],
Reals], Element[Subscript[\[Mu],1], Reals],
Element[Subscript[\[Mu],2], Reals],
Subscript[\[Sigma],0] > 0, Subscript[\[Sigma],1] > 0,
Subscript[\[Sigma],2] > 0, g > 0};
z = Piecewise[{{d + Subscript[x,1], d <= g},
{d + Subscript[x,2], d > g}}];
meana= Expect[z,f3]
vara = Var[z, f3]
sda = Sqrt[vara]
```

A.2 Early arrival

Define normal arrival time function and help function z .

```
A = Exp[-(t - Subscript[\[Mu],a])^2/
(2 Subscript[\[Sigma],a]^2)]/(Sqrt[2 Pi]
Subscript[\[Sigma],a]);
domain[A] = {t, -Infinity, Infinity} &&
Subscript[\[Mu],a] > 0,
Subscript[\[Sigma],a] > 0, o > 0, s > 0};
z = If[t <= o, o + s, t + s];
```

Calculation of the mean using the Expect function

```
meand = Expect[z,A]
```

Calculation of the standard deviation using the `Var` function

```
vard = Var[z,A]
sdd = Sqrt[vard]
```

Calculation of the continuous part of the probability density function using the `Prob` function

```
cdf = Prob[If[t < o, o + s, t + s] < d, A]
domain[cdf] = {d, -Infinity, Infinity}
&& {Subscript[\[Mu],a] > 0,
Subscript[\[Sigma],a] > 0, o > 0, s > 0};
pdf = D[cdf, d]
domain[pdf] = {d, -Infinity, Infinity}
&& {Subscript[\[Mu],a] > 0,
Subscript[\[Sigma],a] > 0, o > 0, s > 0};
```

A.3 Late arrival

Define normal arrival time function and help function `z`.

```
A = Exp[-(t - Subscript[\[Mu],a])^2/
(2 Subscript[\[Sigma],a]^2)]/(
Sqrt[2 Pi] Subscript[\[Sigma],a]);
domain[A] = {t, -Infinity, Infinity} &&
{Subscript[\[Mu],a] > 0,
Subscript[\[Sigma],a] > 0, c > 0, s > 0};
z = If[t <= c, t + s, t];
```

Calculation of the mean using the `Expect` function

```
meand = Expect[z, A]
```

Calculation of the standard deviation using the `Var` function

```
vard = Var[z, A]
sdd = Sqrt[vard]
```

Calculation of the probability density function using the `Prob` function

```
cdf = Prob[If[t < c, t + s, t] < d, A]
domain[cdf] = {d, -Infinity, Infinity}
&& {Subscript[\[Mu],a] > 0,
Subscript[\[Sigma],d] > 0, c > 0, s > 0};
pdf = D[cdf, d] // Simplify
domain[pdf] = {d, -Infinity, Infinity}
&& {Subscript[\[Mu],a] > 0,
Subscript[\[Sigma],a] > 0, c > 0, s > 0};
```

A.4 Time-independent solution solution case 1

A.5 Time-dependent solution solution case 1

A.6 Time-independent solution solution case 2

A.7 Time-dependent solution solution case 2