KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Multi-agent systems for dynamic logistics

Systematic evaluation and bio-inspired optimization

Rinde Roelf Sebastiaan van Lon

Supervisor:
Prof. dr. Tom Holvoet

Dissertation presented in partial
fulfillment of the requirements for the
degree of Doctor of Engineering Science (PhD):
Computer Science

June 2017

Multi-agent systems for dynamic logistics

Systematic evaluation and bio-inspired optimization

Rinde Roelf Sebastiaan VAN LON

Examination committee:
Prof. dr. ir. Joos Vandewalle, chair
Prof. dr. Tom Holvoet, supervisor
Prof. dr. Tom Wenseleers
Prof. dr. ir. Greet Vanden Berghe
Dr. ir. Paul Valckenaers
Prof. dr. ir. Dirk Cattrysse
Prof. dr. Danny Weyns

Prof. dr. Juergen Branke
(The University of Warwick, United Kingdom)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

June 2017

Preface

Pursuing a PhD is a fun and sometimes very challenging journey. I am greatly indebted to all the people that have helped me over the years and without whom it would have been much harder, if not impossible, to complete this text.

First and foremost I would like to thank my promotor, Tom Holvoet, for being such a great supervisor. Tom, thank you for placing your confidence in me from day one, for the freedom you gave me to pursue my own interests, for being so generous with your time, and for having my back when I needed it the most. I will always look back at our collaboration with fondness and cherish the memories of the good times we shared together!

I would like to thank the members of the BioCo3 project, Greet Vanden Berghe, Tom Wenseleers, and Paul Valckenaers for organizing such a nice project. I've found the discussions that we had in the context of this project very valuable.

Juergen Branke, I'm still very happy that I met you all those years ago at that conference in Cluj-Napoca, Romania. It's been a great honor and pleasure to collaborate with you. Thank you for sharing your ideas and expertise!

I'm grateful for the time and effort that Danny Weyns and Dirk Cattrysse put in reading my thesis and many thanks for the valuable feedback. I also would like to thank Joos Vandewalle for being the chair of my jury.

My colleagues from the AgentWise task force have been a great help and support over the years, Stijn, Rutger, Robrecht, Mario, Kristof, Jonathan, and Tung thank you so much! Also, I would like to thank Bartosz for helping with the development of RinSim in the early days and for pitching the idea of using RinSim in the MAS course to Tom!

Many thanks to all imec-DistriNet members with whom I've shared so many lunches over the years, shared DRADS experiences, learned a lot from, and generally had a very good time! In particular, Emad, Alex, Gijs, Willem, Dries, Jan Smans, Job, Fan, Bob, Wilfried, Jesper, Raoul, Thomas Heyman, Nayyab,

Amin, and José, thank you!

I would like to thank imec-DistriNet for being such a professional research group that values its people over procedures and bureaucracy. I've unfortunately experienced that this is far from the default practice in academia, and I want to thank Wouter Joosen in particular for setting such a great example of leading a research group. I want to thank Ghita, Annick, and Katrien for helping me out so many times and for organizing all these things that make our group better. It's been a joy and honor to be part of imec-DistriNet!

I also want to thank the wonderful people from the secretariat, in particular Marleen, Esther, Karen, and Anne-Sophie, for helping me organize all my conference trips and for our great conversations at the coffee machine!

Thanks to my colleagues from the CODeS research group in Ghent for being such a nice and welcoming group, thank you Tony, Joris, Eline, Mustafa, Wim, Túlio, and Jan. I also would like to Erik Van Achter for helping me improve my writing of the EJOR paper, I've learned a lot from you!

I would like to thank Eliseo and Ali for joining me with writing the EJOR paper and for having such nice discussions with me.

I also would like to take the time to thank the hidden heroes of our department, the sysadmins from the systeemgroep, for developing and maintaining such an awesome computer system. Without the great computer setup that you maintain I know that the hundreds of millions of simulations that I did for the work described in this dissertation would not have been possible. Thank you Bart, Anita, Greg, Kris, and Steven!

Thanks to the CW football team for being such a nice group of people, and for letting me score so often ☺. Thank you Toon, Geert, Antoine, Keith, Tuur, Adrian, Matthias, Pedro, Roel, Vincent, Niraj, Adrian, Marco, Dominique, Joris, Vladimir, Laurens, Stelios, Alex, Jan, Elias, Francesco, Koen, and Jannis.

Ik wil natuurlijk ook mijn goede vrienden uit Nederland bedanken, An, Bastijn, Jaap, Jordi, Judith, Kirsten, Lisette, Melissa, Onno, en Paul. Ik vond het soms moeilijk dat ik dingen moest missen omdat ik verder weg was, maar gelukkig lukte het me toch nog om jullie regelmatig te zien, veel dank voor alle steun die jullie me door de jaren hebben gegeven.

Ik wil mijn hele familie bedanken voor de onvoorwaardelijke steun die ik heb gekregen. In het bijzonder wil ik mijn ouders, Anke en Jan Roelf, en mijn broers, Jonne en Jip, bedanken voor jullie warmte en het geloof dat jullie in mij hebben! Ik wil ook Theo en Marion bedanken voor hun steun door de jaren, en Theo wil ik bedanken als meest trouwe non-academische lezer van mijn werk.

Tot slot wil ik Anouk bedanken voor de ongeloofelijke hoeveelheid steun en vertrouwen die je me schenkt. Er zijn geen woorden voor hoeveel ik aan je te danken heb, hoe vaak je een luisterend oor was als 'die computers' weer eens niet deden wat ik wilde, als ik weer eens een onmogelijk bug had gevonden, als dingen tegenzaten, als ik geen zin meer had. Je hebt me altijd weer gestimuleerd om verder te gaan en om het beste uit mezelf te halen ♡.

Abstract

Multi-agent system (MAS) literature often assumes decentralized MAS to be especially suited for dynamic and large scale problems. This assumption is based on the decentralized nature of MAS that allows for partitioning of the problem space such that individual agents can autonomously solve parts of the problem. This enables agents to quickly respond to changes in the problem. There is, however, little scientific evidence to support this assumption. Additionally, the prevailing paradigm in operational research is the use of centralized algorithms.

The main problem addressed in this dissertation is the lack of a systematic evaluation of decentralized MAS and centralized algorithms on dynamic and large scale logistics problems. The problem under consideration is the dynamic pickup-and-delivery problem with time windows (dynamic PDPTW). There are four requirements of such an evaluation. First, exact definitions of dynamism and scale are needed such that these properties can be varied independently. Second, a dataset of dynamic PDPTW instances with varying levels of dynamism and scale is required. Third, there is a need for a realistic and fair simulation platform that allows real-time simulation of dynamic PDPTW and supports centralized as well as decentralized algorithms. Fourth, representative centralized and decentralized implementations are required.

Reproducibility is one of the main principles of the scientific method. Therefore, all components needed for a systematic evaluation should be made open source. A sub-problem also addressed in this dissertation is that of optimizing MAS.

This dissertation comprises five major contributions. Dynamism, urgency, and scale are formally defined in the context of dynamic PDPTWs (contribution 1). Existing definitions of dynamism often mix the concept of dynamism with that of urgency. An empirical evaluation of our definitions of dynamism and urgency shows that the degree of dynamism is negatively correlated with operating costs while more urgent scenarios are correlated with significantly higher operating costs. This justifies the conceptual separation of dynamism and urgency. We

further define scale as a multiplier applied to the number of vehicles and orders in a problem. These three formal definitions enable experiments that investigate the influence of one property on operational costs independently of other properties.

Based on the formal definitions of dynamism, urgency, and scale, a benchmark dataset and problem instance generator of dynamic PDPTWs are constructed (contribution 2). The generated benchmark dataset allows systematic comparison of algorithms subject to varying problem properties.

For performing real-time experiments, a new open-source logistics simulator, RinSim, is developed (contribution 3). The simulator has support for both decentralized MAS as well as centralized algorithms and supports the dataset with different levels of dynamism, urgency, and scale. Both the centralized as well as the decentralized interface of RinSim provide the same software limitations and hardware constraints, thereby providing a fair environment for comparing performance.

A MAS based on the dynamic contract-net protocol (DynCNET) and a centralized tabu search algorithm, based on the OptaPlanner optimization library, are implemented. Using the measures, dataset, and RinSim, a systematic evaluation of these two implementations is conducted (contribution 4). This evaluation experiment is the first of its kind to compare the influence of dynamism, urgency, and scale on the performance of two classes of algorithms in such a systematic, thorough, and fair manner. The results of the comparison show that the solutions found by the centralized algorithm cost, on average, only 94.2% of the operating cost of the solutions found by the DynCNET MAS. This indicates that the centralized algorithms generally perform better compared to the MAS. However, for scenarios that are medium to very dynamic, very urgent, and medium to large scale, the average relative operating cost of the centralized algorithm is 112.3%, indicating that under these circumstances, the MAS performs better compared to the centralized algorithm. When assessing the performance of the algorithms individually per scenario property, there is not one algorithm that generally outperforms the other on that dimension.

To investigate whether the performance of agents can be optimized, the use of hyper-heuristics, more specifically genetic programming (GP), for MAS development is investigated (contribution 5). The heuristic that is evolved by GP is used as agent bid function in the auction process of CNET. The results show that our hyper-heuristic outperforms a reference algorithm, based on OptaPlanner, in all scenarios. In addition, the decentralized hyper-heuristic approach even outperforms the centralized reference algorithm in most situations.

Beknopte samenvatting

In literatuur over multi-agent systemen (MAS'en) wordt vaak aangenomen dat decentrale MAS'en erg geschikt zijn voor dynamische en grootschalige problemen. Deze aanname is gebaseerd op de decentrale eigenschappen van MAS'en die toelaten om problemen zodanig te splitsen dat agenten autonoom deelproblemen kunnen oplossen. Dit stelt agenten in staat om snel te reageren op veranderingen. Er is echter weinig wetenschappelijk bewijs voor deze aanname. Daarnaast zijn juist centrale algoritmen dominant in operationeel onderzoek.

Het voornaamste probleem dat wordt behandeld in deze dissertatie is het gebrek aan een systematische evaluatie van decentrale MAS'en en centrale algoritmen op dynamische en grootschalige logistieke problemen. Het logistieke probleem dat wordt bestudeerd is het dynamische *pickup-and-delivery* probleem met tijdvensters (dynamisch PDPTW). Er zijn vier vereisten voor een goede evaluatie. Ten eerste zijn er exacte definities van dynamiek en schaal vereist zodanig dat deze eigenschappen onafhankelijk gevarieerd kunnen worden. Ten tweede is er een dataset van dynamische PDPTW instanties met verschillende gradaties van dynamiek en schaal benodigd. De derde vereiste is een realistisch en onbevooroordeeld simulatie platform dat een dynamisch PDPTW in realtime kan simuleren en ondersteuning biedt voor zowel centrale als decentrale algoritmen. Ten vierde zijn er representatieve implementaties van centrale en decentrale algoritmes nodig.

Reproduceerbaarheid is een van de belangrijkste principes van de wetenschappelijke methode. Daarom moeten alle bovengenoemde componenten open source beschikbaar worden gemaakt. Een probleem dat ook wordt behandeld in deze dissertatie is het optimaliseren van MAS'en.

Deze dissertatie beschrijft vijf grote contributies. Dynamiek, urgentie, en schaal zijn formeel gedefinieerd in de context van dynamische PDPTW (contributie 1). Bestaande definities van dynamiek koppelen dynamiek vaak aan urgentie. Een empirische evaluatie van onze definities wijst uit dat de mate van dynamiek

negatief gecorreleerd is met operationele kosten terwijl meer urgente scenario's gecorreleerd zijn met significant hogere operationele kosten. Dit rechtvaardigt de conceptuele scheiding van dynamiek en urgentie. We definiëren schaal als een factor toegepast op het aantal voertuigen en bestellingen in een probleem. Deze drie definities maken het mogelijk om de effecten van één eigenschap op de operationele kosten te meten onafhankelijk van andere eigenschappen.

Gebaseerd op de formele definities van dynamiek, urgentie, en schaal is een dataset en probleem instantie generator van dynamische PDPTWs geconstrueerd (contributie 2). Deze benchmark dataset faciliteert systematische vergelijkingen van algoritmen onderhevig aan variërende probleem eigenschappen.

RinSim is een nieuwe open-source simulator specifiek ontwikkeld voor realtime logistieke experimenten (contributie 3). RinSim ondersteunt decentrale MAS'en, centrale algoritmen, en de dataset met verschillende gradaties van dynamiek, urgentie, en schaal. Zowel de centrale als de decentrale interface van RinSim hebben dezelfde soft- en hardware beperkingen. Op deze manier biedt RinSim een onbevooroordeelde omgeving voor het vergelijken van algoritme prestaties.

Een MAS, gebaseerd op het dynamisch contract-net protocol (DynCNET), en een gecentraliseerd taboe zoek algoritme, gebaseerd op de OptaPlanner optimalisatie bibliotheek, zijn geïmplementeerd. Met behulp van de formele definities, dataset, en RinSim zijn deze implementaties systematisch geëvalueerd (contributie 4). Dit evaluatie-experiment is het eerste in zijn soort dat de invloed van dynamiek, urgentie, en schaal, op de prestaties van twee categorieën algoritmen op een dusdanig systematische, grondige, en onbevooroordeelde manier vergelijkt. De resultaten van deze vergelijking tonen aan dat de kosten van de oplossingen die gevonden worden door het centrale algoritme gemiddeld genomen maar 94.2% beslaan ten opzichte van de operationele kosten die gevonden worden door het MAS. Dit geeft aan dat dit centrale algoritme over het algemeen beter presteert ten opzichte van het MAS. Maar, voor scenario's die gemiddeld tot erg dynamisch zijn, erg urgent, en een gemiddelde tot grote schaal hebben, zijn de gemiddelde relatieve operationele kosten van het centrale algoritme 112.3%, het MAS werkt onder deze omstandigheden dus beter. Uit het beoordelen van de prestaties van de algoritmen per probleem eigenschap blijkt dat geen van de algoritmen in alle gevallen beter presteert dan de ander.

Om te onderzoeken of de prestaties van MAS'en kunnen worden geoptimaliseerd, bestuderen we hyper-heuristieken, meer specifiek genetisch programmeren (GP) (contributie 5). De heuristiek die wordt geëvolueerd door GP wordt gebruikt als biedingsfunctie in het veiling proces van CNET. De resultaten laten zien dat onze hyper-heuristiek in alle gevallen beter presteert dan het referentie algoritme dat is gebaseerd op OptaPlanner. De decentrale hyper-heuristiek presteert in de meeste gevallen zelfs beter dan het centrale referentie algoritme.

Abbreviations

CNET contract-net protocol

COP centralized OptaPlanner

DARP dial-a-ride-problem

DGP decentralized genetic programming

DIC decentralized insertion cost

DOP decentralized OptaPlanner

DynCNET dynamic contract-net protocol

ES evolution strategy

GP genetic programming

JVM Java virtual machine

MAS multi-agent system

MIP mixed integer program

NEAT neuroevolution of augmenting topologies

OR operational research

PDP pickup and delivery problem

PDPTW pickup and delivery problem with time windows

TSP traveling salesman problem

VRP vehicle routing problem

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computational demands of most optimization problems grow at a superlinear rate when their scale and dynamism increase. This dissertation constitutes research on multi-agent systems (MAS's) for dynamic optimization problems in the domain of logistics. MAS's are, due to their decentralized nature, often assumed to be especially suited for large scale and dynamic problems. An example dynamic optimization problem is the dynamic pickup and delivery problem with time windows (PDPTW), which is an NP-hard logistics problem (Savelsbergh & Sol, 1995). The prevailing paradigm for solving logistics problems is the use of centralized optimization algorithms. There is currently little scientific evidence that indicates that the centralized paradigm is superior to the decentralized paradigm, or vice versa[1]. The main problem addressed in this dissertation is the lack of an extensible and systematic comparison between decentralized MAS's and centralized algorithms in dynamic optimization problems.

This dissertation comprises five major contributions, written as four separate articles (van Lon et al., 2016, 2017; van Lon & Holvoet, 2015, 2017)[2]. Dynamism, urgency, and scale are formally defined in the context of dynamic PDPTWs (contribution 1). Using these definitions, a dataset with varying levels of dynamism, urgency, and scale is constructed (contribution 2). For performing experiments, a new logistics simulator, RinSim, was developed. RinSim constitutes a technical contribution that provides a simulation platform for comparing algorithms in real-time (contribution 3). Together, RinSim and

[1] In the remainder of this dissertation, references to 'centralized versus decentralized' should always be interpreted as being in the context of dynamic PDPTW.

[2] Portions of the abstract, introduction, and conclusion are reused from these articles.

the dataset enable the systematic evaluation of a decentralized MAS and a centralized algorithm. The performance of both approaches is compared on varying levels of dynamism, urgency, and scale (contribution 4). In this context, performance is defined as minimizing operating costs, a combination of customer waiting times and vehicle travel times. The results show that the centralized approach performs better than the decentralized MAS in most cases, except when the problem is medium to very dynamic, very urgent, and medium to large scale. To investigate whether the performance of agents can be optimized, the use of hyper-heuristics for MAS development is investigated (contribution 5). The results of this experiment show that hyper-heuristics improve agent performance significantly and even outperform the reference centralized algorithm in most situations.

The remainder of this chapter discusses the context (Section 1.1), the problem statement (Section 1.2), and the contributions (Section 1.3), in more detail.

1.1 Context

Multi-agent systems (Weiss, 1999; Wooldridge, 2002) make a broad research area involving autonomous software entities, called agents, that typically have a local view of the world. Areas include decentralized control systems, agent based simulation, game theory, trust & reputation, negotiation, etc. In the present dissertation we use MAS's as a paradigm for designing decentrally controlled systems.

In pickup and delivery problems (PDPs) a fleet of vehicles is tasked with the pickup and delivery of customers or goods. Central to this dissertation is the dynamic PDPTW (Berbeglia et al., 2010). The objective in dynamic PDPTW is to serve all customers while minimizing fuel costs and time window violations. Dynamism is often caused by the arrival of new orders (Pillac et al., 2013), depending on their time windows, some of these orders may need to be serviced urgently. In this dissertation, *purely* dynamic PDPTWs are considered. No information about orders is known ahead of time, implying that algorithms designed for PDPTWs are unable to plan ahead, all computations have to be done online. In general, there are three different centralized approaches to the PDP: exact methods, (meta)heuristics, and stochastic modeling or sampling. Exact methods are known to be less scalable than non-exact methods (Pillac et al., 2013). And, because of the NP-hard nature of PDP, exact methods quickly become infeasible to use. Stochastic modeling or sampling assumes knowledge of a priori information about the future, a possibility that is not considered in this dissertation. Because of their capability of rapidly finding

(sub-optimal) solutions, (meta)heuristics are used as a centralized reference algorithm.

Agents can be applied to dynamic PDPTWs in a natural way. For example, Fischer et al. (1995) use a truck agent that is responsible for operating a vehicle and a shipping company agent responsible for handing out new tasks. These agents can then participate in a dynamic version of contract-net protocol (CNET), first introduced by Smith (1980). CNET is a decentralized auction protocol for the task assignment problem. In CNET, agents can tender a task to which potential contractors can respond by sending a bid. The task is assigned to the agent with the best bid. Decentralized MAS's are often considered to be a good fit for large scale and dynamic problems because of their ability to make swift local decisions (Fischer et al., 1995; Glaschenko et al., 2009; Weyns et al., 2006). Collectively, the local decisions made by all agents aim to solve the global optimization problem. Agents can make these local decisions using two different approaches: 1) explicitly searching through the space of possible schedules using an (exact or heuristic) optimization procedure, or, 2) using a heuristic, a rule of thumb, that guides the agent by assigning priorities to actions without explicitly searching the space of schedules. Creating a MAS using the first approach can be done by using an algorithm or software library for the PDPTW, such as OptaPlanner (De Smet et al., 2016). The second approach, designing a heuristic that governs a single agent, is a challenging task. Local decisions made by agents can have far reaching global consequences. However, hyper-heuristics, a recent technique for the automatic design of heuristics, is a promising alternative to human constructed heuristics.

Hyper-heuristics is a branch of optimization literature concerned with the automatic design of heuristics (Burke et al., 2013). Genetic programming (GP) is a subfield of evolutionary computing (Eiben & Smith, 2007), capable of generating heuristics of arbitrary complexity. Just like any evolutionary algorithm, GP maintains a population of solution candidates whom are repeatedly recombined to increase their fitness using natural selection. GP can be an effective hyper-heuristic for the design of a MAS because the effectiveness of the entire MAS (global level) can be used as fitness indicator to evolve a heuristic at the local level. This principle has been demonstrated before by Beham et al. (2009) and van Lon et al. (2012).

1.2 Problem statement

The main problem addressed in this dissertation is the lack of a systematic evaluation comparing decentralized MAS's and centralized algorithms on

logistics problems.

There are four requirements for a systematic evaluation of centralized and decentralized algorithms with varying levels of dynamism, urgency, and scale. First, measures for the problem properties under investigation are required. These measures should be conceptually orthogonal, meaning that a measure for one concept may not be (partially) mixed with aspects of other concepts. Such a measure allows for independent variation of one problem property and study its effect on optimization algorithms. Second, a problem instance generator with varying levels of dynamism, urgency, and scale, that is based on these measures, is required. With this problem instance generator a dataset can be constructed that can serve as a benchmark. Third, a realistic and fair logistics simulation platform is required that allows simulating problem instances from the dataset while supporting centralized and decentralized algorithms. To be able to investigate time related properties such as dynamism and urgency, the simulator must be able to simulate in real-time. Fairness of the simulator should be ensured by subjecting both algorithm paradigms to the same constraints with respect to hardware resources and software limitations. Fourth, representative centralized and decentralized implementations are required.

Additionally, all source code, datasets, and results produced in such an evaluation should be made publicly available. It has been argued before by Ince et al. (2012) and van Lon & Holvoet (2013) that opening all relevant artifacts would aid reproducibility, accountability, and extensibility of research.

A sub-problem addressed in this dissertation is that of optimizing MAS for logistics. This is a challenging task due to the decentralized nature of MAS's, at algorithm design time it is hard to foresee the consequences that an action of a single agent has on the collective.

1.3 Summary of contributions and outline

The five contributions of this dissertation are summarized in this section. For each contribution, the relevant content chapters that describe the contribution are indicated. The four content chapters, chapters 2 to 5, are papers[3] that are included verbatim, with the following exceptions: typographical errors are corrected, references between the papers have been changed into their corresponding chapter names, and 'paper' has been changed to 'chapter' when appropriate. The dissertation is concluded in Chapter 6.

[3]Chapters 2, 3, and 5 are published papers. At the time of writing, Chapter 4 is under review, the submitted version is included in this dissertation.

1.3.1 Measures of dynamic pickup-and-delivery problems

Formal definitions of dynamism and urgency (Chapter 2), and scale (Chapter 3) are put forward. Existing definitions of dynamism often mix the concept of dynamism with that of urgency. An empirical evaluation of the definitions of dynamism and urgency shows that changes in dynamism and urgency have a different influence on the solution quality in dynamic logistic problems. This justifies the conceptual separation of dynamism and urgency. We further define scale as a multiplier applied to the number of vehicles and orders in a problem. These three formal definitions enable experiments that investigate the influence of one property on solution quality independently of other properties.

1.3.2 Dataset

Based on the formal definitions of dynamism, urgency, and scale, a benchmark dataset and problem instance generator are presented (Chapter 3). To avoid any interactions between the variables, the dataset generator is created meticulously. The generated benchmark dataset allows systematic comparison of algorithms. By open sourcing the dataset generator, other researchers are enabled to create their own datasets and conduct new investigations.

1.3.3 A realistic simulation platform

The real-time logistics simulator, RinSim, is presented as a technical contribution (Chapter 4). The simulator has support for both decentralized MAS's as well as centralized algorithms and supports the dataset with different levels of dynamism, urgency, and scale. Both the centralized as well as the decentralized interface of RinSim provide the same software limitations and hardware constraints, thereby providing a fair environment for comparing performance. RinSim is entirely open-source to support reproducibility of all experiments and to allow extensibility of all components.

1.3.4 Systematic evaluation of centralized algorithms and decentralized multi-agent systems

A MAS based on CNET and a centralized tabu search algorithm are implemented (Chapter 4). Using the measures, dataset, and RinSim, a systematic evaluation of these two implementations is conducted. This evaluation experiment is the first of its kind to compare the influence of dynamism, urgency, and scale on

the performance of two classes of algorithms in such a systematic, thorough, and fair manner. The code of the algorithms as well as the experiment results data are published to allow complete reproducibility of the evaluation. Because all components are completely open source, this evaluation provides a baseline of performance comparisons between centralized and decentralized algorithms.

1.3.5 Genetic programming of multi-agent systems

Based on the evaluation of centralized and decentralized algorithms, an investigation on optimizing MAS is conducted (Chapter 5). The main hypothesis is that hyper-heuristics, more specifically GP, can be used to improve agents decentrally coordinated via CNET. The heuristic that is evolved by GP is used as agent bid function in the auction process of CNET. This contribution, the combination of hyper-heuristics and MAS, provides a first step towards the automatic design of MAS's.

Chapter 2

Measures of dynamism and urgency in logistics

In order to evaluate the performance of algorithms on differing degrees of dynamism, a formal definition of dynamism is required. This chapter contains the paper:

> van Lon, R. R. S., Ferrante, E., Turgut, A. E., Wenseleers, T., Vanden Berghe, G., & Holvoet, T. (2016). Measures of dynamism and urgency in logistics. *European Journal of Operational Research*, 253(3), 614–624. doi:10.1016/j.ejor.2016.03.021

The definitions proposed in this chapter are the result of discussions led by Rinde R.S. van Lon with all authors. Rinde R.S. van Lon did all the programming and most of the writing, the other authors gave feedback on the writing and Eliseo Ferrante contributed an early version of the related work section.

Abstract

Dynamism was originally defined as the proportion of online versus offline orders in the literature on dynamic logistics. Such a definition however, loses meaning when considering purely dynamic problems where all customer requests arrive dynamically. Existing measures of dynamism are limited to either 1) measuring the proportion of online versus offline orders or 2) measuring urgency, a concept

that is orthogonal to dynamism, instead. This chapter defines separate and independent formal definitions of dynamism and urgency applicable to purely dynamic problems. Using these formal definitions, instances of a dynamic logistic problem with varying levels of dynamism and urgency were constructed and several route scheduling algorithms were executed on these problem instances. Contrary to previous findings, the results indicate that dynamism is positively correlated with route quality; urgency, however, is negatively correlated with route quality. This chapter contributes the theory that dynamism and urgency are two distinct concepts that deserve to be treated separately.

2.1 Introduction

Logistic optimization problems aim at minimizing costs while serving customers' transportation requests. The most common problem formalization is the vehicle routing problem (VRP) (Dantzig & Ramser, 1959). Roads are treated as edges of a graph and a traveling salesman problem (TSP) is solved for one or more vehicles represented in such a graph (Flood, 1956). In practice, vehicle schedules are devised offline, after all customer requests have been received, and are applied later on without the possibility to modify the schedules once the vehicles have started servicing.

A number of technological advances have fostered new interest and transformed problems in the domain of logistics. Such advances are the introduction of the Global Positioning System (GPS) in 1996, the increasing accuracy of Geographic Information Systems (GIS), and more recently the development and spread of tablets and smart phones with high-bandwidth Internet. Online changing of routes or devising completely new routes is now possible due to the availability of accurate information on the position of all vehicles. These developments open new avenues for increasing customer satisfaction (i.e. relatively fast shipping of goods, even at the day of ordering), while operational costs and environmental impact can be further decreased. In the dynamic variant, the typical dynamic aspect is the arrival time of the request containing the useful information needed to compute optimal routes for the vehicles (Pillac et al., 2013).

Dynamic logistics is a well researched topic continuing to receive widespread attention (Berbeglia et al., 2010; Parragh et al., 2008; Pillac et al., 2013). Psaraftis (1995) and later Eksioglu et al. (2009) devised taxonomies for the (dynamic) VRP, but did not formally define dynamism as such. Pillac et al. (2013) suggested that a better formalization of the dynamics would allow more precise classification of problem instances. Based on such a classification, it would be possible to scientifically assess the quality of algorithms for dynamic

logistic problems in different circumstances. For instance, datasets such as those presented in (Gendreau et al., 2006; Li & Lim, 2001; Mitrović-Minić & Laporte, 2004) could be classified and compared quantitatively and it would be possible to find specific dynamic properties within dynamic logistics where one class of algorithms performs better than others. The cornerstone of a formalization of dynamics in logistics is a formal definition of dynamism. Intuition suggests that the frequency of change should be part of such a definition of dynamism. A more dynamic problem is characterized by a more continuous distribution of request arrivals. Static problems, on the other hand, have all requests available at the same time or, alternatively, become available in bursts and thus have a more varying request arrival frequency. Furthermore, different optimization algorithms likely differ in their ability to find near-optimal solutions for highly dynamic problems. When information is clustered together, the available time can be used for devising a good schedule, contrastingly, frequent changes of the problem definition make scheduling in advance almost useless and favor a completely reactive strategy instead.

Lund et al. (1996) proposed the first formal measure for quantifying dynamism in logistic problems. They define dynamism as the proportion of requests known after the scheduling phase (i.e. when vehicles are already shipping) with respect to the total number of requests. Their measure considers a problem where all requests arrive during shipping as 100% dynamic. Contrary to our intuition, the relative timing of the requests does not influence the value of this dynamism measure. Larsen et al. (2002) recognized the limitation of the measure by Lund et al. and aimed at fixing it by taking into account the urgency of a request. Larsen et al.'s measure considers a request to be more dynamic when announced closer to its deadline. However, this approach fails to measure what intuitively could be considered dynamism, since it does not measure the relative distribution of request announcements. On closer inspection, the concept of urgency is included in the degree of dynamism considered by Larsen et al. Moreover, Larsen et al. showed that for problems with a high dynamism value, the algorithms tested produced a lower quality schedule. Based on their experimental setup, concluding whether the negative correlation between their measure and schedule quality is the result of dynamism, urgency or a combination thereof is nearly impossible.

The present chapter investigates whether the experimental observations reported by Larsen et al. are caused by dynamism, urgency or both. We analyze whether splitting urgency and dynamism into separate concepts is desirable. To conduct a sound scientific evaluation, we need to be able to formally define both dynamism and urgency as two separate concepts and to develop the tools for classifying logistic scenarios. These tools enable generating instances of logistic problems with varying levels of dynamism and urgency. The instances are realistic,

while capable of sharing common characteristics, excepting differing levels of dynamism and urgency. The dataset thus generated contains instances of the dynamic PDPTW, a special case of the VRP that is sufficiently relevant to allow general claims. Further, the dataset, the simulator and all code is available online to allow reproducibility of all results.

The chapter is organized as follows. First, the relevant literature is discussed (Section 2.2). Second, dynamic PDPs are formally defined and dynamism and urgency are explained intuitively (Section 2.3). The novel measures which form the main contribution of the chapter are explained (Section 2.4) and the empirical evaluation is discussed (Section 2.5). Finally, the conclusions based on the experimental evaluation are presented and the usefulness of the proposed measures to advance the field of dynamic logistics and beyond is discussed (Section 2.6).

2.2 Related work

The VRP was first introduced (Dantzig & Ramser, 1959) as a generalization of the TSP (Flood, 1956). A dynamic version of VRP was first studied considering a dynamic version of a special case of VRP transportation of people (Wilson & Colvin, 1977): the dial-a-ride-problem (DARP) (Cordeau & Laporte, 2003). The customer requests (trips from a source to a destination) in a DARP appear dynamically. These type of requests were later formally defined in (Psaraftis, 1980) as *immediate requests*, distinguished from *advanced requests* that are received before the beginning of the planning horizon.

In this section, we review the existing literature on previously proposed dynamism measures. We also briefly review the state of the art on the dynamic PDPTW.

2.2.1 Dynamism and measures

The first dynamism measure was introduced by Lund et al. (1996) and later refined by Larsen et al. (2002). Section 2.4.2 discusses these measures in detail after an intuitive definition of dynamism is presented. Larsen (2000) proposed a framework that distinguishes between weakly, moderately and strongly dynamic systems. The intention of this framework is to quickly find an appropriate algorithm based on the problem's classification.

Beside these works, we have no knowledge of any work that defines measures for dynamism within the field of operations research. Nevertheless, several authors make interesting observations related to dynamism in logistics.

A first observation, by Kilby et al. (1998), is that the arrival rate of new tasks in a dynamic VRP is important. If the problem updates constantly, an algorithm will require more restarts than in the case where requests arrive in widely separated bursts. Similarly, Pillac et al. (2013) note that the frequency of updates in problem information have a dramatic impact on the time available for optimization. The statements made by Kilby et al. and Pillac et al. align with what intuitively could be considered dynamism since the arrival rate of requests is similar to the relative distribution of request announcements.

A second observation, also by Kilby et al. (1998), is about the time at which a commitment to serve a customer at a particular time must be made. The time of the commitment is one of the fundamental questions in dynamic routing. Kilby et al. define a dynamism-related measure called the *commit horizon*, which denotes the period where the schedule is fixed before the latest possible commit time. The latter is problem-dependent but is often defined as the operation's starting time. Although we did not consider the commit horizon in our study, it may be an interesting property to investigate related to dynamism.

A third noteworthy observation about dynamism in logistics is made by Borndörfer et al. (1999). In the static DARP, the computed schedule and the schedule executed on the next day often differ significantly because of cancellations of requests, spontaneous requests, vehicle breakdowns and other unpredictable events. This observation suggests that static DARPs are exceptional in practice.

2.2.2 Literature review on the dynamic PDPTW

Gendreau & Potvin (1998) discussed application domains in which dynamic vehicle routing problems occur, such as dial-a-ride (taxi) problems and courier and repair services. Berbeglia et al. (2010) presented an extensive overview of variants of dynamic PDPs. The dynamic PDPTW is a special case of the dynamic VRP. It should be noted that the dynamic PDPTW is often seen as a stochastic problem, in which some knowledge about the nature of the arrivals is known in advance in a stochastic way, while the actual requests become known only during the operation day (Fu, 2002; Pillac et al., 2013). Psaraftis (1995) remarked, without formally defining near-term, that in dynamic vehicle routing near-term events are more important than long-term events. Research on the dynamic PDPTW has mainly concentrated on algorithm development, in this

section, we only review work in which the dynamic PDPTW is seen from a completely dynamic perspective, without any a priori knowledge.

Gendreau et al. (2006) presented a dynamic version of tabu search with a neighboring structure based on ejection chains. The optimization procedure is run while the environment is static. When new requests arrive, or when a vehicle has finished pickup or delivery, the algorithm performs insertion and ejection moves. Madsen et al. (1995) developed an insertion heuristic to tackle the dynamic DARP with time windows for moving elderly and disabled people in Denmark.

Mitrović-Minić et al. (2004) presented an approach based on two time horizons: a short time horizon aimed at achieving the short-term goal of minimizing the distance traveled, and a longer time horizon aimed at achieving the long-term goal of facilitating the insertion of future requests. Five more rolling horizon heuristics were considered and compared in (Yang et al., 2004). Mitrović-Minić & Laporte (2004) first considered two very simple heuristics: drive-first, that requires a vehicle to drive as soon as it is feasible according to the earliest departure time, and wait-first that instead requires the vehicle to wait at its current location as long as it is feasible. Two more waiting strategies aim to achieve a trade-off between the first two heuristic and at better handling waiting times in order to facilitate insertion of future requests.

Pureza & Laporte (2008) proposed two strategies, a waiting and a buffering strategy. Differently from (Mitrović-Minić & Laporte, 2004), the waiting strategy exploits extra information provided by the computation of the faster path. In order to minimize the earliness of a service at a location, the buffering strategy, instead, postpones the assignment of the least urgent new requests to the latest possible time. The idea underlying both strategies is to schedule requests in batches, retrieving as much information as possible to produce better schedules.

In search of a dispatching algorithm that could imitate a human being, Potvin et al. (1993) presented a learning system based on linear programming. The system is able to learn an optimal policy taking into account expert decisions in former situations. The same idea was implemented by Shen et al. (1995), who based an approach on neural networks and by Benyahia & Potvin (1998), where genetic programming was used.

An alternative approach to the dynamic PDPTW relies on a decentralized MAS instead of a centralized decision maker. Every vehicle is assumed to be able to perform some computation and to communicate with any other vehicle anytime. A first comparison between centralized and decentralized approaches was performed by Mes et al. (2007). They compared traditional heuristics developed in earlier work (van der Heijden et al., 2002) against novel distributed

MAS that use a Vickrey auction to bid for new pickup and delivery requests when they appear. It was shown that the performance of the MAS approach is often at least as good as traditional heuristics. Mes et al. (2010) further improved the performance of the MAS by introducing a look-ahead mechanism. Bidding uses value functions to estimate the expected future revenue of inserting a new order in an agent's schedule. Other look-ahead strategy combinations were proposed in (Mes et al., 2013).

2.3 Dynamic pickup-and-delivery problems

In PDPs, a fleet of vehicles deals with customer transportation requests. A request is handled when an item is transported from pickup to delivery location as requested by the customer. In dynamic PDPs, requests may arrive at any time during the fleet's operating hours, necessitating the maintenance of a flexible schedule. No prior information is available about the number of requests that may still arrive nor about their locations or time windows.

2.3.1 Formal definition

The definition of the dynamic PDP used throughout this chapter is based on (Gendreau et al., 2006). A *scenario*, which describes the unfolding of a dynamic PDP, is defined as a tuple:

$$\langle \mathcal{T}, \mathcal{E}, \mathcal{V} \rangle := \text{scenario},$$

where

$$\begin{aligned} [0, \mathcal{T}) &:= \text{time frame of the scenario}, & \mathcal{T} &> 0 \\ \mathcal{E} &:= \text{list of events}, & |\mathcal{E}| &\geq 2 \\ \mathcal{V} &:= \text{set of vehicles}, & |\mathcal{V}| &\geq 1 \end{aligned}$$

$[0, \mathcal{T})$ is the period in which the fleet of vehicles $\mathcal{V}$ have to handle all customer requests. The events represent customer requests. We distinguish between advance events and dynamic events. Advance events are known before time 0 of the time frame of the scenario. Dynamic events are instead revealed between time 0 and time $\mathcal{T}$ and describe new transportation requests, or can possibly introduce other new information. Each event $e_i \in \mathcal{E}$ is defined by the following

variables:

$$a_i := \text{announce time}$$

$$p_i := \left[p_i^L, p_i^R\right) = \text{pickup time window, } p_i^L < p_i^R$$

$$d_i := \left[d_i^L, d_i^R\right) = \text{delivery time window, } d_i^L < d_i^R$$

$$pst_i := \text{pickup service time span}$$

$$dst_i := \text{delivery service time span}$$

$$ploc_i := \text{pickup location}$$

$$dloc_i := \text{delivery location}$$

$$tt_i := \text{travel time from pickup location to delivery location}$$

Similar to (Larsen et al., 2002) we define reaction time as the length of the interval between the order arrival time a_i and the closing of the pickup time window p_i^R:

$$r_i := p_i^R - a_i = \text{reaction time} \tag{2.1}$$

The time window related variables of a transportation request are visualized in Figure 2.1.

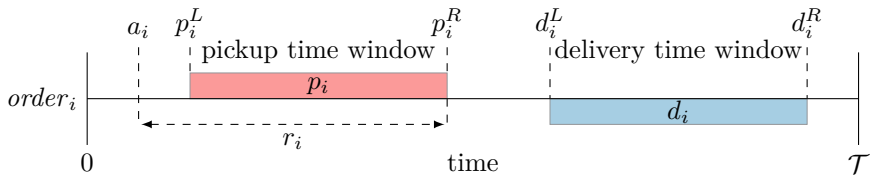


Figure 2.1: Visualization of the time related variables of a single order event $e_i \in \mathcal{E}$.

Furthermore we assume that:

- vehicles start at a depot and have to return after all orders are handled;
- the fleet of vehicles $\mathcal{V}$ is homogeneous;
- the cargo capacity of vehicles is infinite (e.g. courier service);
- the vehicle is either stationary or driving at a constant speed;
- contrastingly to (Gendreau et al., 2006), vehicle diversion is allowed, meaning that a vehicle can divert from its destination at any time;

- vehicle fuel is infinite and driver fatigue is not an issue;
- each location can be reached from any other location; and,
- the scenario is completed when all pickup and deliveries have been executed and all vehicles have returned to the depot.

Vehicle schedules are subject to both hard and soft constraints. The openings of time windows are hard constraints and therefore:

$$sp_i \geq p_i^L \tag{2.2}$$

$$sd_i \geq d_i^L \tag{2.3}$$

sp_i is the start of the pickup operation of order event e_i by a vehicle; similarly, sd_i is the start of the delivery operation of order event e_i by a vehicle. The time windows closings (p_i^R and d_i^R) are soft constraints. They are incorporated into the objective function, which is defined similarly to (Gendreau et al., 2006) and needs to be minimized:

$$min := \sum_{j \in \mathcal{V}} \left(vtt_j + td\left\{bd_j, \mathcal{T}\right\}\right) + \sum_{i \in \mathcal{E}} \left(td\left\{sp_i, p_i^R\right\} + td\left\{sd_i, d_i^R\right\}\right) \tag{2.4}$$

where

$$td\left\{\alpha, \beta\right\} := max\left\{0, \alpha - \beta\right\} \;=\; \text{tardiness} \tag{2.5}$$

vtt_j is the total travel time of vehicle v_j; bd_j is the return time of vehicle v_j to the depot. The objective function computes the total vehicle travel time, the total tardiness of vehicles returning to the depot and the total pickup and delivery tardiness. The objective function determines the route cost of a solution, where a fleet of vehicles executes a scenario. A low route cost corresponds with a high quality route.

We further impose the following hard constraints on the construction of scenarios to ensure consistency and feasibility of individual orders:

$$r_i \geq 0 \tag{2.6}$$

$$d_i^R \geq p_i^R + pst_i + tt_i \tag{2.7}$$

$$d_i^L \geq p_i^L + pst_i + tt_i \tag{2.8}$$

These constraints are visualized in Figure 2.2. The reaction time constraint (eq. 2.6) ensures that an order is always announced before its due date. The time window constraints (eq. 2.7 and eq. 2.8) ensure that pickup and delivery time windows are compatible. These constraints ensure that a pickup operation

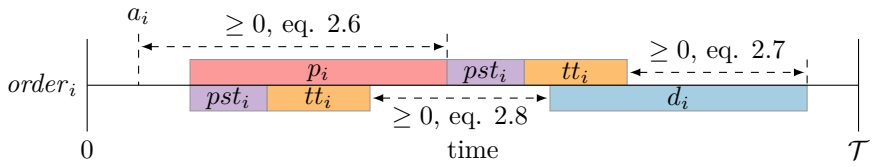


Figure 2.2: Time window constraints of an order event $e_i \in \mathcal{E}$.

started at any time within p_i allows a delivery within d_i, given that a vehicle is available and respecting vehicle capacity, service time and travel time constraints.

2.4 Measure design

Although we aim at measuring properties of dynamic PDPTWs, the concepts of dynamism and urgency are not limited to measuring properties of the problem class dynamic PDPTW. In general, properties of a series of events should be measured.

2.4.1 Intuitive definitions

Dynamism and urgency are abstract variables that capture two aspects of dynamic PDPTWs. We consider these variables to be problem related as opposed to algorithm related; the applied algorithm should have no influence on the value of the measures. However, the dynamism and urgency measures may assist in choosing an appropriate algorithm for a PDPTW instance. Further, the measures should be conceptually orthogonal, i.e. a measure for one concept should not be (partially) mixed with aspects of other concepts. Therefore, urgency and dynamism should not be correlated.

Dynamism

We base our notion of dynamism on the meaning of the word dynamic: "*marked by usually continuous and productive activity or change*" (Merriam Webster, 2014). Therefore we consider the degree of dynamism to be the *continuity* of change. A very dynamic scenario is one that changes continuously while a less dynamic scenario only changes occasionally. This is visualized in Figure 2.3. We further define a *change* to be an event that introduces additional information

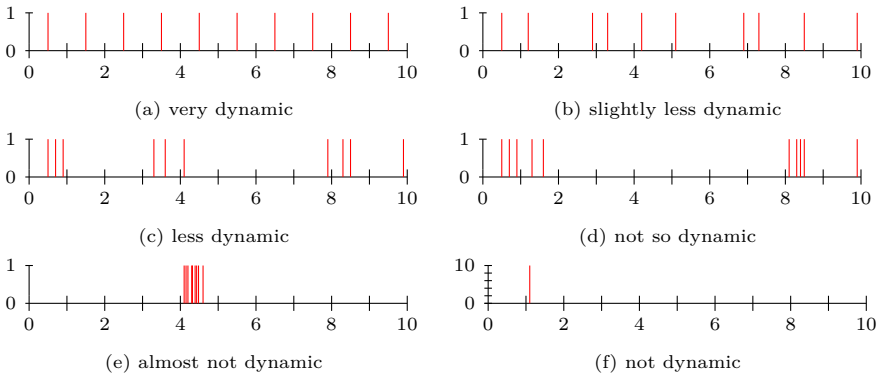


Figure 2.3: Visualization of order arrival times, each red bar indicates an event in which a new order is announced. The figures (a) to (f) are presented in decreasing order of dynamism. In (a) the events have equal interarrival times and are nicely distributed over the period, in (b) and (c) we see that changes occur less frequently. In (d) and (e) all events arrive in one or two batches making it less continuous and therefore less dynamic. In (f) all 10 events arrive at the same time resulting in a scenario with no dynamism.

to the problem, such as an order event as defined formally in Section 2.3.1. In our interpretation, knowing the dynamism of a problem does not give any extra information on the predictability of events.

Urgency

Urgency is an indicator of the reaction time available for responding to an incoming order. Urgency can be expressed in time units and defined as the difference between order arrival time and closing of the pickup time window as shown in Figure 2.4.

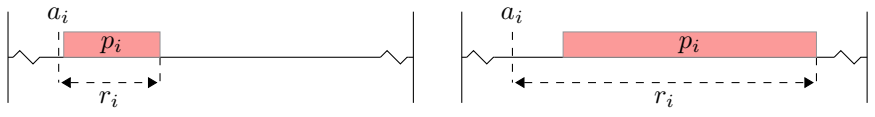


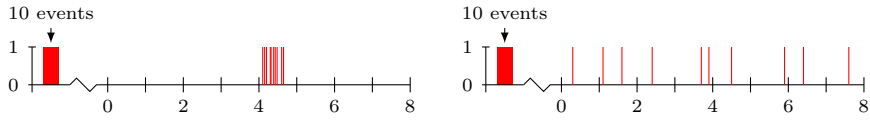
Figure 2.4: Visualization of events with different degree of urgency, a relatively urgent order (a), and a relatively less urgent order (b).

2.4.2 Degree of dynamism

The degree of dynamism was defined by Lund et al. (1996):

$$dod := \frac{\text{Number of dynamic requests}}{\text{Total number of requests}} \tag{2.9}$$

It is the proportion of dynamic requests with respect to the total requests (including dynamic and advance requests). The definition ignores information related to dynamism defined in Section 2.4.1. For instance, a scenario where all events are announced in a relatively short burst has the same *dod* as a scenario where events are announced more evenly throughout the day (see Figure 2.5). This means that the applicability of *dod* is limited and not suitable for purely



(a) burst (b) continuously changing

Figure 2.5: Two scenarios both with 10 advance events and 10 dynamic events. Both scenarios have a proportion of dynamic requests of 50% (eq. 2.9) but the dynamism of the two scenarios is remarkably different.

dynamic scenarios (i.e. scenarios without advance events). Since this measure does not measure dynamism as we conceive it, we propose to rename it to the *proportion of dynamic requests*.

Larsen et al. (2002) recognized the limitations of eq. 2.9 and designed the *effective degree of dynamism* in an attempt to measure dynamism more accurately:

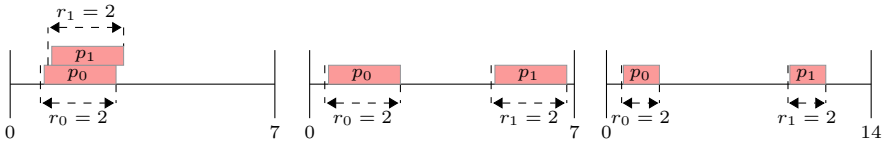
$$edod := \frac{\sum_{i=1}^{n_{imm}} \left(\frac{a_i}{\mathcal{T}}\right)}{n_{tot}} \tag{2.10}$$

Where n_{imm} is the number of dynamic requests and n_{tot} is the total number of events. They also proposed a similar measure that takes time windows into account:

$$edod_{tw} := \frac{1}{n_{tot}} \sum_{i=1}^{n_{tot}} \left(1 - \frac{r_i}{\mathcal{T}}\right) \tag{2.11}$$

Since the problem under investigation includes time windows, the analysis focuses on $edod_{tw}$ (eq. 2.11), without loss of generality.

Figure 2.6 shows three scenarios with their respective value for the $edod_{tw}$ measure. Figure 2.6(a) and Figure 2.6(b) show two similar scenarios that differ



(a) $edod_{tw} = \frac{5}{7}$ (b) $edod_{tw} = \frac{5}{7}$ (c) $edod_{tw} = \frac{6}{7}$

Figure 2.6: Three different scenarios with two transportation requests. Only the pickup time window is shown. (a) and (b) have $\mathcal{T} = 7$ and (c) has $\mathcal{T} = 14$.

in the arrival times of the transportation requests. They have the same value for $edod_{tw}$, even though their arrival times are quite different. It differs from the definition of dynamism introduced in the present chapter (Section 2.4.1), where the scenario depicted in Figure 2.6(b) has a higher degree of dynamism than that of Figure 2.6(a).

The arrival times in Figure 2.6(b) have been multiplied by 2 in order to obtain Figure 2.6(c). The reaction times are the same. According to our definition of dynamism these two scenarios should have the same level of dynamism, but according to $edod_{tw}$ they are different. This difference is problematic because it means that $edod_{tw}$ is dependent on the length of the scenario $\mathcal{T}$, hence scenarios of different length can not be compared using the $edod_{tw}$ measure.

2.4.3 Dynamism measure

We define the list of interarrival times Δ as follows:

$$\Delta := \{\delta_0, \delta_1, \ldots, \delta_{|\mathcal{E}|-2}\} = \{a_j - a_i | j = i + 1 \wedge \forall a_i, a_j \in \mathcal{E}\} \tag{2.12}$$

$$|\Delta| := |\mathcal{E}| - 1 \tag{2.13}$$

Based on the visualization of a scenario with 100% dynamism in Figure 2.3(a) we can define a *perfect interarrival time* that is required for 100% dynamism as follows:

$$\theta := \text{perfect interarrival time} = \frac{\mathcal{T}}{|\mathcal{E}|} \tag{2.14}$$

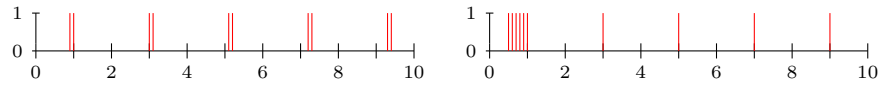
The perfect interarrival time enables computing the deviation of an interarrival time relative to the 100% case:

$$\sigma_i := \begin{cases} \theta - \delta_i & \text{if } i = 0 \text{ and } \delta_i < \theta \\ \theta - \delta_i + \dfrac{\theta - \delta_i}{\theta} \times \sigma_{i-1} & \text{if } i > 0 \text{ and } \delta_i < \theta \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

Consequently, the deviation of an entire scenario is defined as:

$$\sum_{i=0}^{|\Delta|} \sigma_i \tag{2.16}$$

Since bursts are defined as interarrival times that are smaller than θ, this definition focuses on interarrival times that are smaller than θ. In case $\delta_i < \theta$, a recursive penalty, expressed by the term $\frac{\theta - \delta_i}{\theta} \times \sigma_{i-1}$, is applied. This penalty proportionately adds the deviation of the previous interarrival time. In short, the penalty term is used to recognize bursts, to measure their size, and to take their contribution into account. The motivation for this recursive penalty can best be explained using an example. Consider the scenario shown in Figure 2.7. Following the description of dynamism in Section 2.4.1, the scenario



(a) Five small bursts (b) One big burst

Figure 2.7: 10 events, 5 x interarrival time of .1, 4 x interarrival time of 2.

in Figure 2.7(a) is more dynamic than the one in Figure 2.7(b). When examining the interarrival times of both scenarios, it shows that $\theta = 1$ and both have five interarrival times of .1 and four interarrival times of 2:

$$\Delta^a = \{.1, 2, .1, 2, .1, 2, .1, 2, .1\}$$

$$\Delta^b = \{.1, .1, .1, .1, .1, 2, 2, 2, 2\}$$

Computing only the deviations from the perfect interarrival time is not enough to distinguish between these scenarios. Therefore the recursive penalty used in eq. 2.15 distinguishes between these two scenarios by taking into account the deviation of the preceding interarrival time. In this example (Figure 2.7) these

deviations become:

$$\sigma^a = \{.9, 0, .9, 0, .9, 0, .9, 0, .9\}$$

$$\sigma^b = \{.9, 1.71, 2.439, 3.0942, 3.68478, 0, 0, 0, 0\}$$

The deviation of the event series from the 100% case has to be normalized with respect to the theoretical maximum deviation for a scenario (i.e. the 0% case). We compute the maximum as follows:

$$\sum_{i=0}^{|\Delta|} \bar{\sigma}_i \tag{2.17}$$

where

$$\bar{\sigma}_i := \theta + \begin{cases} \dfrac{\theta - \delta_i}{\theta} \times \sigma_{i-1} & \text{if } i > 0 \text{ and } \delta_i < \theta \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

In eq. 2.18, the perfect interarrival time is multiplied by the number of interarrival times and the recursive penalty is also added in order to dynamically increase the maximum. Adding the recursive penalty to the maximum prevents $\bar{\sigma}$ from becoming greater than σ.

Combining eq. 2.15 and eq. 2.17 the definition of dynamism becomes:

$$\textit{dynamism} := 1 - \frac{\text{deviation}}{\text{max deviation}} = 1 - \frac{\sum_{i=0}^{|\Delta|} \sigma_i}{\sum_{i=0}^{|\Delta|} \bar{\sigma}_i} \tag{2.19}$$

Continuing with the scenarios in Figure 2.7, dynamism is 50% for scenario A and 27.6% for scenario B (see Section 2.7 for the complete calculation).

2.4.4 Urgency measure

Based on Figure 2.4 we can define a measure for urgency of a single order event:

$$\textit{urgency}\,(e_i) := p_i^R - a_i = r_i \tag{2.20}$$

Urgency is the reaction time expressed in time units. In order to get an indication of the urgency of an entire scenario, one can compute the mean and standard deviation of urgency.

Note that this definition is similar to Larsen et al.'s *effective degree of dynamism* (eq. 2.11) but differs in a key aspect. The urgency value is not normalized to the length of the scenario. We believe that the length of a scenario and urgency should be independent and should therefore not be coupled in the definition of urgency.

2.5 Evaluation

We evaluate the dynamism and urgency measures by investigating their impact to route quality generated by scheduling algorithms. Route quality is defined as the inverse of the cost of a route, where route cost is computed using the objective function defined in eq. 2.4. Three hypotheses are investigated:

- When increasing the dynamism of a scenario the average route quality of an algorithm decreases
- When increasing the urgency of a scenario the average route quality of an algorithm decreases
- When increasing both dynamism and urgency the average route quality of an algorithm decreases

2.5.1 Dataset generator

To evaluate the influence of dynamism and urgency on strategies developed for dynamic PDPTW, it is imperative to be able to create scenarios with any level of urgency and dynamism. Furthermore, these scenarios have to be as similar as possible, except for their possibly different urgency and dynamism, while still being the result of a stochastic process. Therefore, a dataset generator has been constructed and used to generate a dataset with 11 levels of dynamism (0 to 100% with steps of 10%) and 10 levels of urgency (0 to 45 minutes with steps of 5 minutes). This results in 110 different scenario settings. We produced 20 different instances for each setting resulting in 2200 scenarios.

Controlling dynamism of time series

A homogeneous Poisson process is a common model for the arrival of stochastic events, e.g. phone calls (Willkomm et al., 2009) and requests of an individual document on a web server (Arlitt & Williamson, 1997). The dynamism of time

series obtained using a Poisson process was first investigated. Interestingly, the results show that time series generated using a homogeneous Poisson process are in a range of 45% to 60% dynamism. Other methods for generating time series with dynamism lower than 45% and dynamism higher than 60% have been investigated. A time period $\mathcal{T} = 12$ and number of events $|\mathcal{E}| = 360$ for generating time series was used. The homogeneous Poisson process has a constant intensity function, which is defined as:

$$\lambda(t) = \frac{|\mathcal{E}|}{\mathcal{T}} = 30 \tag{2.21}$$

A non-homogeneous Poisson process has a variable intensity function. It is based on a sine wave to control the dynamism of a scenario. By varying the parameters of the sine wave, the properties of event bursts can be controlled:

$$\lambda(t) = a \cdot \sin(t \cdot f \cdot 2\pi - \pi \cdot p) + h \tag{2.22}$$

$$a = \qquad \text{amplitude} \tag{2.23}$$

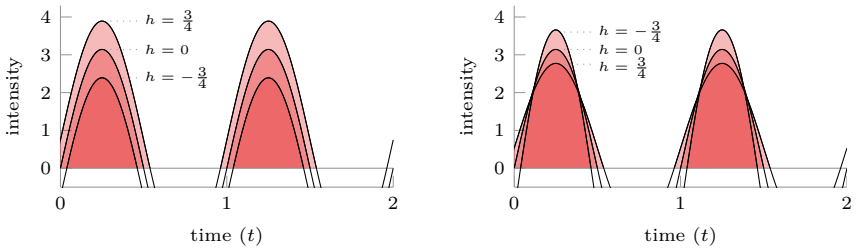
$$f = \qquad \text{frequency} \tag{2.24}$$

$$p \sim \mathcal{U}(0, 1) \qquad \text{phase shift} \tag{2.25}$$

$$h \sim \mathcal{U}(-.99, 1.5) \qquad \text{height} \tag{2.26}$$

In order to keep the total number of events constant with different levels of dynamism, the amplitude and height parameters are rescaled such that the total area under the intensity function equals $|\mathcal{E}|$. After rescaling, the resulting events following a non-homogeneous Poisson process are generated using the thinning method (Lewis & Shedler, 1979). Figure 2.8 visualizes the effect of the height parameter on the event intensity and therefore on the dynamism of a scenario.

Since the non-homogeneous Poisson process only generates scenarios with a dynamism lower than or equal to scenarios generated with the homogeneous Poisson process, a different method had to be used for generating more dynamic scenarios. A method that generates higher levels of dynamism is attained by drawing interarrival times from a normal distribution. We used the truncated normal distribution $\mathcal{N}\left(\frac{\mathcal{T}}{|\mathcal{E}|}, 0.04\right)$ with a lower bound of 0 and a standard deviation of 0.04 was found experimentally to yield the best results. If a value x was drawn such that $x < 0$, a new number was drawn from the distribution. Truncating a normal distribution actually shifts the mean, hence the mean was rescaled to make sure the effective mean was equal to $\frac{\mathcal{T}}{|\mathcal{E}|}$.



(a) Before rescaling

(b) After rescaling

Figure 2.8: Visualization of intensity function $\lambda(t) = \pi \cdot \sin(t \cdot 2\pi) + h$ with $h = \frac{3}{4}$, 0 and $-\frac{3}{4}$. The area above $y = 0$ is highlighted to indicate the event intensity. In (a) the function is shown before any scaling, in (b) the function is scaled such that the area above $y = 0$ equals 1 for all three functions. Generally, lower values for the h parameters result in lower dynamism since this creates more intense bursts in shorter periods (resulting in higher peaks in (b)).

The fourth method for generating interarrival times is a uniform distribution with mean $\frac{\mathcal{T}}{|\mathcal{E}|}$ and a maximum deviation σ. The σ value is (for each scenario again) drawn from the truncated normal distribution $\mathcal{N}(1, 1)$ with bounds $[0, .25]$. If a value σ is obtained from the distribution such that $\sigma > .25$ or $\sigma < 0$, a new value is drawn. The mean is not scaled, and therefore the effective mean of σ is higher than 1.

An experiment was conducted where each previously described method was used to generate 1000 samples (time series). For each sample the dynamism was computed using eq. 2.19. We repeated this experiment until we found the parameters that produce scenarios in the entire range of 0% to 100% dynamism. Figure 2.9 shows the final results of this experiment as a frequency diagram. Based on these experimental results we conclude that these four methods can

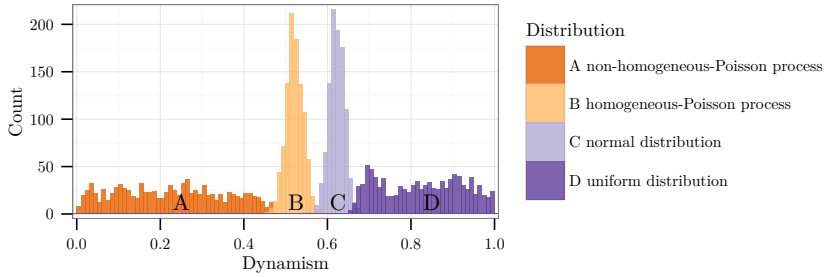


Figure 2.9: Frequency diagram comparing four methods for generating time series. Each method was used to produce 1000 time series, for each time series the dynamism was measured using eq. 2.19.

be used to generate scenarios with dynamism ranging from 0 to 100%. Table 2.1 shows the time series generators and the levels of dynamism they generate. The probability that one of the time series generated has exactly the desired

Dynamism values	**Time series generator**
0, 5, 10, 15, 20, 25, 30, 35, 40, 45	non-homogeneous Poisson process
50, 55	homogeneous Poisson process
60, 65	Normal distribution
70, 75, 80, 85, 90, 95, 100	Uniform distribution

Table 2.1: Overview of dynamism values used in the dataset and the corresponding time series generator.

dynamism value is very small. A radius of 1% has therefore been defined around each dynamism value. For this dataset, we consider a scenario with dynamism d such that $n - 1 < d < n + 1$ to have dynamism n, where n is one of the dynamism values listed in Table 2.1.

Generating comparable scenarios with different dynamism and urgency levels

The aim of the dataset is to have a set of scenarios where all settings are the same except for the dynamism and urgency levels. Also, we strive to minimize any interactions between variables, e.g. dynamism should not correlate with time window intervals. This ensures that any effect measured is solely the result of the difference in dynamism and or urgency.

The dataset generators are stochastic functions. Although all parameters are set as to make desired results as likely as possible, undesirable scenarios can not be completely avoided. Therefore, we employ a filter that only accepts scenarios corresponding with the following requirements:

- All scenarios must have exactly $|\mathcal{E}|$ events, (the time series generators all produce time series which have on *average* the correct number of events, but not always).
- Scenarios must have a dynamism that fits in one of the dynamism bins from Table 2.1.

We further define the concept of *office hours* as the period $[0, \mathcal{O})$ in which new orders are accepted. To ensure feasibility of individual orders we need to take

into account the travel time, service time durations and urgency:

$$\mathcal{O} = \mathcal{T} - pst_{max} - dst_{max} - \begin{cases} 2 \cdot tt_{max} & \text{if } u < \frac{1}{2} \cdot tt_{max} \\ 1\frac{1}{2} \cdot tt_{max} - u & \text{otherwise} \end{cases} \tag{2.27}$$

Here, pst_{max} and dst_{max} are the maximum pickup and delivery service times respectively, tt_{max} is the maximum travel time between a pickup and delivery location and u is urgency.

The pickup and delivery time windows have to be randomly chosen while respecting the constraints set out by the urgency level and the announce time. The p_i^R is defined as the sum of a_i and u, hence it follows that p_i^L needs to be between a_i and the sum of a_i and u:

$$p_i^L = \begin{cases} \sim \mathcal{U}\big(a_i, p_i^R - 10\big) & \text{if } u > 10 \\ a_i & \text{otherwise} \end{cases} \tag{2.28}$$

Here, 10 is the minimum pickup time window length unless urgency is less than 10, in that case the urgency level is the pickup time window length. The upper bound of d_i^R can be defined as:

$$ubd_i^R = \mathcal{T} - tt(dloc_i, depot_{loc}) - dst_i \tag{2.29}$$

This translates as the latest possible time to start the delivery operation such that the delivery time window constraints are met and the vehicle can still be back at the depot on time. The lower bound of d_i^L was already defined in eq. 2.8:

$$lbd_i^L = p_i^L + pst_i + tt_i \tag{2.30}$$

We define a minimum delivery time window length of 10, which results in an upper bound of d_i^L:

$$ubd_i^L = ubd_i^R - 10 \tag{2.31}$$

Based on these bounds we draw the opening of the delivery time window from the following uniform distribution:

$$d_i^L \sim \mathcal{U}\big(lbd_i^L, \max\big(lbd_i^L, ubd_i^L\big)\big) \tag{2.32}$$

To find d_i^R we need to redefine the lower bound (from eq. 2.7) by using the actual value of d_i^L:

$$lbd_i^R = \min\big(\max\big(p_i^R + pst_i + tt_i, d_i^L + 10\big), ubd_i^R\big) \tag{2.33}$$

Finally, the closing of the delivery time window is defined as:

$$d_i^R \sim U\big(lbd_i^R, ubd_i^R\big) \tag{2.34}$$

All locations in a scenario are points on the Euclidean plane. It has a size of 10 by 10 kilometer with a depot at the center of this square. At the start of the scenario all 10 vehicles are at the depot. The vehicles have a constant travel speed of 50 km/h. All pickup and delivery locations are drawn from a two dimensional uniform distribution $\mathcal{U}_2(0, 10)$.

For simulating a scenario we use the discrete time simulator RinSim (van Lon & Holvoet, 2012) version 3.0.0 (van Lon, 2014b). The time unit is set to milliseconds and the tick size to 1000 ms. The pickup and delivery service times pst_i and dst_i are set to 5 minutes. For reproducibility, all code and data are published on an accompanying web page (van Lon, 2016a).

2.5.2 Heuristic algorithms used to solve dynamic PDPTW

The cheapest insertion heuristic (Algorithm 1) and the 2-opt optimization procedure (Algorithm 2) were used in the experiments. Since the 2-opt procedure requires a complete schedule as input, it uses the cheapest insertion heuristic for inserting new orders to yield a complete schedule. These two heuristics have been used in earlier work for vehicle routing problems (Coslovich et al., 2006; Psaraftis, 1983; Solomon, 1987) and the heuristics are general enough not to have a bias towards scenarios with specific levels of dynamism or urgency. Each

```
Input: ⟨T, E, V⟩;                                          /* A scenario as input */
Data: S;                                                   /* the current schedule or ∅ */
S_best = ∅
foreach e ∈ E, e ∉ S do
    /* generate all PDP insertion points in the current schedule:                    */
    insertions = generate_insertion_points(S)
    for i ∈ insertions do
        /* construct a new schedule by inserting e at insertion i                    */
        S_new = construct(S,e,i)
        if cost(S_new) < cost(S_best) then
            S_best = S_new
        end
    end
end
```

Algorithm 1: Cheapest insertion heuristic, source code available in (van Lon, 2014a).

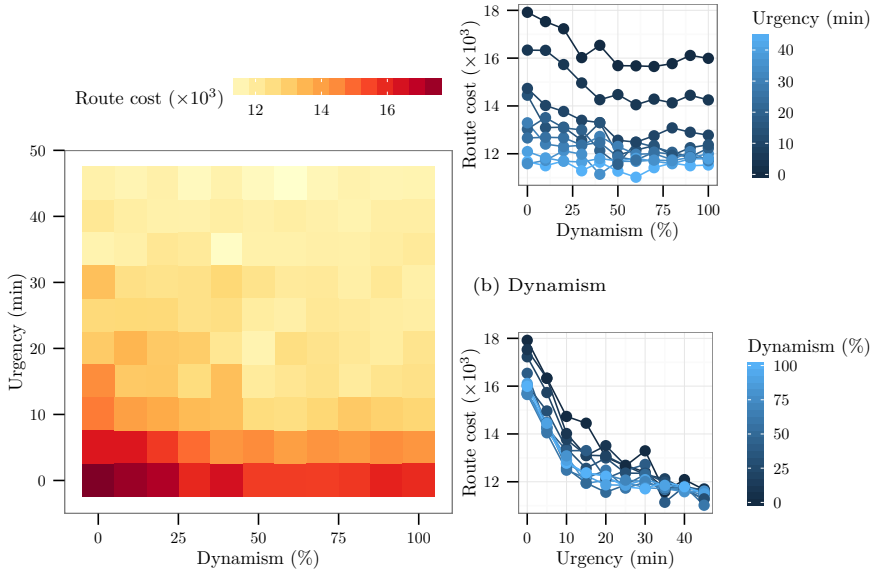
time a new order is announced, the algorithms are executed to produce a new schedule for the fleet of vehicles. It is assumed that execution of the algorithm is instantaneous with respect to the dynamics of the simulations.

```
Input: S
S_best = S
swaps = generate_swaps(S)
 foreach e ∈ swaps do
    S_new = swap(S,e)
    if cost(S_new) < cost(S_best) then
       S_best = S_new
    end
end
/* If a better schedule has been found, we start another iteration                    */
if S_best ≠ S then
   2-opt(S_best)
end
```

Algorithm 2: 2-opt procedure, source code available in (van Lon, 2014a).

2.5.3 Results and analysis

20 scenarios have been generated per level of dynamism and urgency. For each scenario both algorithms were used for controlling the fleet of vehicles, Figures 2.10 and 2.11 show the experimental results.



(a) Heatmap (b) Dynamism (c) Urgency

Figure 2.10: Experimental results using cheapest insertion heuristic.

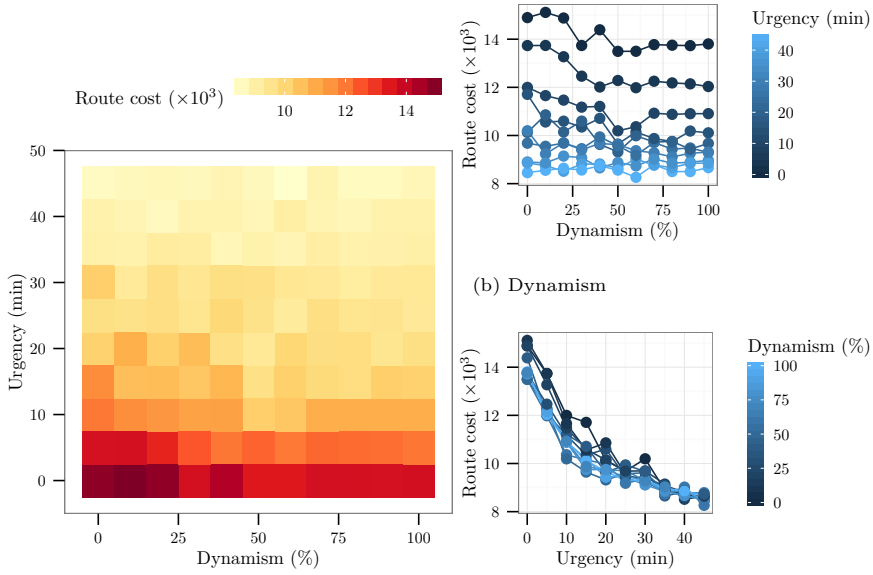


Figure 2.11: Experimental results using cheapest insertion heuristic with 2-opt.

The first hypothesis: increasing dynamism decreases average route quality, is not supported by the results (Figures 2.10 and 2.11). In fact, for very urgent scenarios, the route costs decreased when dynamism was increased (Figures 2.10(b) and 2.11(b)). We believe that the relatively small effect of dynamism on route costs can be explained by the assumed instant response of the algorithms. This means that the algorithms never need to or can be interrupted during calculations and will always give their best possible answer. However, it is expected (Kilby et al., 1998) that when requests arrive at a constant rate, an algorithm has relatively little computation time due to restarts triggered by new requests. Therefore, an interesting direction for future work is to investigate advanced algorithms that require more computation time and can also be interrupted.

Our second hypothesis is that more urgent scenarios (lower urgency values) result in increased route costs. This prediction is strongly supported, under all levels of dynamism considered. The data further suggests that when scenarios become less urgent, route costs decreases diminish. This result is expected. When new requests need to be handled urgently by a fleet of vehicles still busy handling previous requests, it is natural that some delays are introduced.

Further, when requests are urgent there is less room for optimizing the length of routes, leading to longer processing times, which eventually results in higher total routing costs.

The third hypothesis states that increasing both dynamism and urgency leads to a lower average route quality. The computational results do not support this hypothesis, as can be seen by the lack of a peak in the bottom right of Figures 2.10(a) and 2.11(a). Rather surprisingly, an interaction effect can be observed in the lower left corner of the same image. This effect is significantly highly non-linear according to a multiple regression polynomial model. Very urgent scenarios with low dynamism seem to be the hardest scenarios for the algorithms to solve. An explanation for this area of most difficult scenarios is possibly that these scenarios incorporate large bursts of very urgent orders. These bursts may be as big or bigger than the fleet of available vehicles, which therefore quickly run out of time to meet the time windows of requests.

Model selection based on the Akaike Information Criterion revealed that the route cost was best predicted by a multiple regression model in which dynamism was included as a quadratic polynomial, urgency as a cubic polynomial, plus the interaction between both polynomials (overall model fit for Algorithm 1: adjusted $R^2 = 0.609$, $p < 2 \cdot 10^{-16}$, $AIC = 37591$, for Algorithm 2: $R^2 = 0.7322$, $p < 2 \cdot 10^{-16}$, $AIC = 36833$, significance of dynamism, urgency and their interaction effect in Algorithm 1 and 2 was always $p < 10^{-16}$). Furthermore, calculated eta squared values show that urgency had a much larger effect than dynamism or the interaction between both factors (for Algorithm 1: eta squared is 0.54, 0.04 and 0.02, for Algorithm 2: eta squared is 0.70, 0.04 and 0.01).

Based on these results it can be concluded that dynamism and urgency are two different concepts, each affecting the problem in a different way. This justifies the presented theory that dynamism and urgency should be separated, as opposed to the measure by Larsen (2000), in two different measures.

2.6 Conclusion

The present chapter argues that urgency and dynamism are conceptually different and we propose separate measures for both concepts. In support of this conceptual separation, the experimental results show that the degree of dynamism and urgency have a different influence on the solution quality in dynamic logistic problems. Interestingly, the degree of dynamism is negatively correlated with operating costs while more urgent scenarios are correlated with significantly higher operating costs.

The negative correlation between degree of dynamism and operating cost is possibly explained by the algorithms' assumptions. It can be expected that with a real time setup, where an algorithm can be interrupted during computation, a positive correlation may exist. Furthermore, the correlations between urgency and operating cost are expected to be general in kind, since urgent requests constrain the fleet of vehicles, regardless of the algorithm that is being used.

During the realization of this article the authors published a new benchmark dataset for dynamic PDPTW with different levels of dynamism, urgency and scale (Chapter 3), where scale is a combination of number of vehicles, number of requests and area size. The dataset enables systematic comparison of the performance of a broad range of algorithms under varying conditions. Similar to Máhr et al. (2010), a comparison of centralized and decentralized approaches applied to this dataset should shed more light on the strengths and weaknesses of both approaches. Such a comparison should clarify whether problems with different levels of dynamism, urgency and scale can better be addressed with either of the two approaches.

Acknowledgements

This research was funded by the Research Fund KU Leuven, the Fund for Scientific Research (FWO) - Flanders, the ESF "H2SWARM" program, and the KULeuven projects IDO-BioCo3 and Excellence Center Financing PF/2010/007. A.E. Turgut acknowledges grant TUBITAK-2219.

2.7 Appendix: Dynamism calculation example

Two examples of computing dynamism using eq. 2.19.

$$\mathcal{E}^a = \{.9, 1, 3, 3.1, 5.1, 5.2, 7.2, 7.3, 9.3, 9.4\}$$

$$\mathcal{T}^a = 10$$

$$\theta^a = 1$$

$$\Delta^a = \{.1, 2, .1, 2, .1, 2, .1, 2, .1\}$$

$$\sigma^a = \{.9, 0, .9, 0, .9, 0, .9, 0, .9\}$$

$$\bar{\sigma}^a = \{1, 1, 1, 1, 1, 1, 1, 1, 1\}$$

$$\text{dynamism}^a = 1 - \frac{4.5}{9} = 0.5$$

$$\mathcal{E}^b = \{.5, .6, .7, .8, .9, 1, 3, 5, 7, 9\}$$

$$\mathcal{T}^b = 10$$

$$\theta^b = 1$$

$$\Delta^b = \{.1, .1, .1, .1, .1, 2, 2, 2, 2\}$$

$$\sigma^b = \{.9, 1.71, 2.439, 3.0942, 3.68478, 0, 0, 0, 0\}$$

$$\bar{\sigma}^b = \{1, 1.81, 2.539, 3.1951, 3.78478, 1, 1, 1, 1\}$$

$$\text{dynamism}^b = 1 - \frac{11.82798}{16.32888} = 0.2756$$

Chapter 3

Towards systematic evaluation of multi-agent systems in large scale and dynamic logistics

Using the definitions of dynamism and urgency from previous chapter and the definition of scale that we propose in this chapter, we construct a dataset generator. With this dataset generator we construct a dataset with varying levels of dynamism, urgency, and scale. This chapter contains the paper:

> van Lon, R. R. S. & Holvoet, T. (2015). Towards systematic evaluation of multi-agent systems in large scale and dynamic logistics. In Q. Chen, P. Torroni, S. Villata, J. Hsu, & A. Omicini (Eds.), *PRIMA 2015: Principles and Practice of Multi-Agent Systems: 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings* (pp. 248–264). Cham: Springer International Publishing. doi:10.1007/978-3-319-25524-8_16

The content presented in this chapter is the result of discussions of both authors. Rinde R.S. van Lon programmed the software and wrote the paper, Tom Holvoet provided feedback on the writing.

Abstract

A common hypothesis in multi-agent systems (MAS) literature is that decentralized MAS are better at coping with dynamic and large scale problems compared to centralized algorithms. Existing work investigates this hypothesis in a limited way, often with no support for further evaluation, slowing down the advance of more general conclusions. Investigating this hypothesis more systematically is time consuming as it requires four main components: 1) formal metrics for the variables of interest, 2) a problem instance generator using these metrics, 3) (de)centralized algorithms and 4) a simulation platform that facilitates the execution of these algorithms. Present chapter describes the construction of an instance generator based on previously established formal metrics and simulation platform with support for (de)centralized algorithms. Using our instance generator, a benchmark logistics dataset with varying levels of dynamism and scale is created and we demonstrate how it can be used for systematically evaluating MAS and centralized algorithms in our simulator. This benchmark dataset is essential for enabling the adoption of a more thorough and systematic evaluation methodology, allowing increased insight in the strengths and weaknesses of both the MAS paradigm and operational research methods.

3.1 Introduction

In PDPs a fleet of vehicles is tasked with transporting customers or goods from origin to destination (Parragh et al., 2008; Savelsbergh & Sol, 1995). In dynamic PDPs the orders describing the vehicles' tasks arrive during the operating hours (Berbeglia et al., 2010), necessitating online assignment of vehicles to orders. The dynamic nature and potential large scale of this problem makes exact algorithms often infeasible.

Decentralized MAS's are often presented as a good alternative to centralized algorithms (Fischer et al., 1995; Glaschenko et al., 2009; Weyns et al., 2006), MAS's are especially promising for large scale and dynamic problems due to their ability to make quick local decisions. Previous work has shown that MAS's can sometimes outperform centralized algorithms in specific cases (Fischer et al., 1995; Máhr et al., 2008, 2010; Mes et al., 2007). However, to the best of our knowledge there has never been a systematic effort to compare centralized algorithms to decentralized MAS's with varying levels of dynamism and scale.

Although the previously mentioned papers each do a thorough evaluation of a MAS applied to a logistics problem, it is often hard to do further comparisons using these papers because of the lack of available problem data, source code or

both. It has been argued before that this is a problem in science in general (Ince et al., 2012), and in multi-agent systems literature in particular (van Lon & Holvoet, 2013).

In this chapter we introduce a dataset generator and a benchmark dataset of the dynamic PDPTW with support for varying three variables. The degree of dynamism and urgency of a dynamic PDPTW are two variables that were introduced before (Chapter 2). The proposed dataset contains an additional variable, scale, that we define in the context of PDPTW as a multiplier applied to the number of vehicles and orders in a problem. Using this dataset it will be possible to systematically investigate the following hypotheses in the context of PDPTW:

- Multi-agent systems perform better when compared to centralized algorithms on very dynamic problem instances
- Multi-agent systems perform better when compared to centralized algorithms on more urgent problem instances
- Multi-agent systems perform better when compared to centralized algorithms on large scale problem instances

Investigating these hypotheses should lead to insight in the performance of both decentralized MAS's and centralized algorithms for PDPTWs. These insights can then be used to make more informed decisions when designing a system that needs to cope with dynamic, urgent and large scale problems. Additionally, the dataset generator, the benchmark dataset instance and the simulator (van Lon & Holvoet, 2012) that we use are open sourced. This improves the reproducibility of this chapter while presenting an opportunity for other researchers to investigate the above hypotheses using their own algorithms.

This chapter is organized as follows. First, the relevant literature is discussed (Section 3.2) and we define dynamic PDPTWs including the measures for dynamism, urgency and scale and the measure for algorithm performance (Section 3.3). This is followed by a description of the dataset generator and dataset benchmark instance (Section 3.4). It is demonstrated how the hypotheses of dynamism, urgency and scale can be investigated using the proposed benchmark instance (Section 3.5), leading to the conclusion that the benchmark dataset facilitates a systematic and long term research effort into these hypotheses (Section 3.6).

3.2 Related work

Several literature surveys discuss the dynamic VRP and its special case, dynamic PDPTW (Berbeglia et al., 2010; Gendreau & Potvin, 1998; Pillac et al., 2013; Psaraftis, 1995). The dynamic PDPTW is often treated as a stochastic problem where some a priori information is known about the orders. This section only discusses papers that do not use a priori information but view the problem from a completely dynamic perspective.

3.2.1 Centralized algorithms

Madsen et al. (1995) developed an insertion heuristic for the dynamic DARP with time windows for moving elderly and disabled people. Potvin et al. (1993) presented a learning system based on linear programming that can learn an optimal policy taking into account decisions of an expert in past scenarios. Mitrović-Minić et al. (2004) presented an approach based on two time horizons: a short time horizon aimed at achieving the short-term goal of minimization of distance traveled, and a longer time horizon aimed at achieving the long-term goal of facilitating the insertion of future requests. Gendreau et al. (2006) introduced a dynamic version of tabu search with a neighboring structure based on ejection chains. When new requests arrive, the algorithm reacts by insertion and ejection moves and with local search.

3.2.2 Multi-agent systems

An alternative approach to the dynamic PDPTW is using a decentralized MAS instead of a centralized planner. Fischer et al. (1995) used a MAS with the extended contract net protocol for cooperative transportation scheduling and they showed that its performance was comparable to existing operational research (OR) techniques. Mes et al. (2007) compared traditional heuristics with a distributed MAS that uses a Vickrey auction to bid for new pickup and delivery requests when they appear, showing that the MAS approach performs often better than traditional heuristics. In subsequent work Mes et al. (2010) further improved the performance of the MAS by introducing a look-ahead mechanism in which bidding uses value functions to estimate the expected future revenue of inserting a new order in an agent plan. Máhr et al. (2010) thoroughly evaluated a MAS with auctions and a mixed-integer program on real world data of a PDPTW. Their results show that both approaches have comparable performance. Glaschenko et al. (2009) discussed the deployment of

a MAS for a taxi company in London, adopting the MAS led to an increase of taxi fleet utilization by 5 - 7 %.

3.3 Dynamic pickup-and-delivery problems

We base our definition of dynamic PDPs on Chapter 2 which is an adaptation of the definition of (Gendreau et al., 2006). In PDPs there is a fleet of vehicles responsible for the pickup-and-delivery of items. The dynamic PDP is an online problem, the customer transportation requests are revealed over time during the fleet's operating hours. It is further assumed that the fleet of vehicles has no prior knowledge about the total number of requests nor about their locations or time windows.

3.3.1 Formal definition

For describing the dynamic PDP we adopt the formal definition of Chapter 2. A *scenario*, which describes the unfolding of a dynamic PDP, is defined as a tuple:

$$\langle \mathcal{T}, \mathcal{E}, \mathcal{V} \rangle := \text{scenario},$$

where

$$\begin{aligned} [0, \mathcal{T}) &:= \text{time frame of the scenario}, & \mathcal{T} > 0 \\ \mathcal{E} &:= \text{list of events}, & |\mathcal{E}| \geq 2 \\ \mathcal{V} &:= \text{set of vehicles}, & |\mathcal{V}| \geq 1 \end{aligned}$$

$[0, \mathcal{T})$ is the period in which the fleet of vehicles $\mathcal{V}$ has to handle all customer requests. The events represent customer transportation requests. Since we consider the purely dynamic PDPTW, all events are revealed between time 0

and time $\mathcal{T}$. Each event $e_i \in \mathcal{E}$ is defined by the following variables:

$$a_i := \text{announce time}$$

$$p_i := [p_i^L, p_i^R) = \text{pickup time window, } p_i^L < p_i^R$$

$$d_i := [d_i^L, d_i^R) = \text{delivery time window, } d_i^L < d_i^R$$

$$pst_i := \text{pickup service time span}$$

$$dst_i := \text{delivery service time span}$$

$$ploc_i := \text{pickup location}$$

$$dloc_i := \text{delivery location}$$

$$tt_i := \text{travel time from pickup location to delivery location}$$

Reaction time is defined as:

$$r_i := p_i^R - a_i = \text{reaction time} \tag{3.1}$$

The time window related variables of a transportation request are visualized in Figure 3.1.

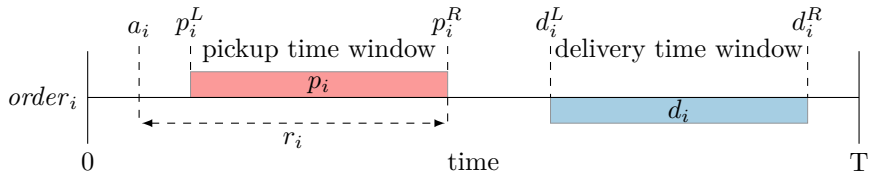


Figure 3.1: Visualization of the time related variables of a single order event $e_i \in \mathcal{E}$.

Furthermore we assume that:

- vehicles start at a depot and have to return after all orders are handled;
- the fleet of vehicles $\mathcal{V}$ is homogeneous;
- the cargo capacity of vehicles is infinite (e.g. courier service);
- the vehicle is either stationary or driving at a constant speed;
- vehicle diversion is allowed, this means that a vehicle is allowed to divert from its destination at any time;

- vehicle fuel is infinite and driver fatigue is not an issue;
- the scenario is completed when all pickup and deliveries have been made and all vehicles have returned to the depot; and,
- each location can be reached from any other location.

Vehicle schedules are subject to both hard and soft constraints. The opening of time windows is a hard constraint, hence vehicles need to adhere to these:

$$sp_i \geq p_i^L \tag{3.2}$$

$$sd_i \geq d_i^L \tag{3.3}$$

Here, sp_i is the start of the pickup operation of order event e_i by a vehicle; similarly, sd_i is the start of the delivery operation of order event e_i by a vehicle. The time window closing (p_i^R and d_i^R) is a soft constraint incorporated into the objective function, it is defined similarly to (Gendreau et al., 2006) and needs to be minimized:

$$min := \sum_{j \in \mathcal{V}} \left(vtt_j + td\left\{bd_j, \mathcal{T}\right\}\right) + \sum_{i \in \mathcal{E}} \left(td\left\{sp_i, p_i^R\right\} + td\left\{sd_i, d_i^R\right\}\right) \tag{3.4}$$

where

$$td\left\{\alpha, \beta\right\} := max\left\{0, \alpha - \beta\right\} \;=\; \text{tardiness} \tag{3.5}$$

Here, vtt_j is the total travel time of vehicle v_j; bd_j is the time at which vehicle v_j is back at the depot. In summary, the objective function computes the total vehicle travel time, the tardiness of vehicles returning to the depot and the total pickup and delivery tardiness.

We further impose the following hard constraints on the construction of scenarios to ensure consistency and feasibility of individual orders:

$$r_i \geq 0 \tag{3.6}$$

$$d_i^R \geq p_i^R + pst_i + tt_i \tag{3.7}$$

$$d_i^L \geq p_i^L + pst_i + tt_i \tag{3.8}$$

These constraints are visualized in Figure 3.2. The reaction time constraint (eq. 3.6) ensures that an order is always announced before its due date. The time window constraints (eq. 3.7 and eq. 3.8) ensure that pickup and delivery time windows are compatible with each other. Hence, a pickup operation started at any time within p_i guarantees feasibility of a delivery within d_i given that a vehicle is available and respecting vehicle capacity, service time and travel time constraints.

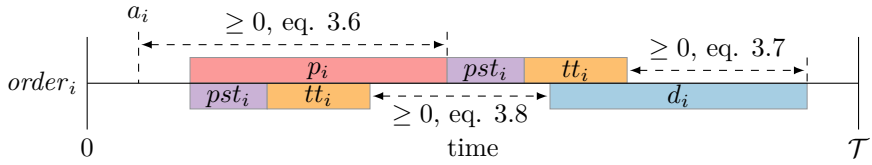


Figure 3.2: Visualization of the time window constraints of an order event $e_i \in \mathcal{E}$.

3.3.2 Dynamism

In this section we describe the measure for the degree of dynamism first defined in Chapter 2. Informally, a scenario that changes continuously is said to be dynamic while a scenario that changes occasionally is said to be less dynamic. In the context of PDPTWs a change is an event that introduces additional information to the problem, such as the events in $\mathcal{E}$. More formally, the degree of dynamism, or the continuity of change, is defined as:

$$dynamism := 1 - \frac{\sum_{i=0}^{|\Delta|} \sigma_i}{\sum_{i=0}^{|\Delta|} \bar{\sigma}_i} \tag{3.9}$$

where

$$\Delta := \{\delta_0, \delta_1, \ldots, \delta_{|\mathcal{E}|-2}\} = \{a_j - a_i | j = i+1 \wedge \forall a_i, a_j \in \mathcal{E}\} \tag{3.10}$$

$$\theta := \text{perfect interarrival time} = \frac{\mathcal{T}}{|\mathcal{E}|} \tag{3.11}$$

$$\sigma_i := \begin{cases} \theta - \delta_i & \text{if } i = 0 \text{ and } \delta_i < \theta \\ \theta - \delta_i + \dfrac{\theta - \delta_i}{\theta} \times \sigma_{i-1} & \text{if } i > 0 \text{ and } \delta_i < \theta \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

$$\bar{\sigma}_i := \theta + \begin{cases} \dfrac{\theta - \delta_i}{\theta} \times \sigma_{i-1} & \text{if } i > 0 \text{ and } \delta_i < \theta \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

This measure can compute the degree of dynamism of any scenario.

3.3.3 Urgency

In Chapter 2 urgency is defined as the maximum reaction time available to the fleet of vehicles in order to respond to an incoming order. Or more formally:

$$urgency\,(e_i) := p_i^R - a_i = r_i \tag{3.14}$$

To obtain the urgency of an entire scenario the mean and standard deviation of the urgency of all orders can be computed.

3.3.4 Scale

Assigning a scale level to a PDP instance allows to conduct a scalability experiment to investigate the existence of a correlation between the scale of a PDP and the computation time and solution quality of an algorithm.

In the context of computer systems scaling up is defined as maintaining a fixed execution time per task while scaling the workload up in proportion to the number of processors applied to it (Gunther, 2006). Analogously, scaling in the context of PDPs can be defined as maintaining a fixed computation time per order while scaling the workload (number of orders) up in proportion to the number of vehicles in the fleet.

However, there are three factors that limit the usefulness of this definition. First, it is known that PDPTWs are NP-hard (Savelsbergh & Sol, 1995), therefore an exact algorithm for a PDPTW requires time that is superpolynomial in the input size. Therefore, maintaining a fixed computation time per order when using an exact algorithm is infeasible. When using an anytime algorithm (an algorithm that can be stopped at any moment during its execution to return a valid solution) such as a heuristic, maintaining a fixed computation time per order is trivial, but will likely have an influence on the solution quality.

Second, the previously mentioned notion of urgency influences the amount of available computation time. Within an order's urgency period three activities need to be performed, first a vehicle needs to be selected, then the selected vehicle needs to drive towards the pickup location and it needs to perform the actual pickup operation. The longer the computation of the vehicle selection takes, the less time remains for the driving and picking up.

Third, depending on the degree of dynamism there may be many orders with a small interarrival time. Each order that arrives while a computation takes place forces a premature halt and subsequent restart of the algorithm. Therefore, maintaining a fixed computation time per order is nonsensical for PDPTWs.

For these reasons, we define scaling in PDPTWs as *maintaining a fixed objective value per order while scaling the number of orders up in proportion to the number of vehicles in the fleet*. Using this definition, scaling up a scenario $\langle \mathcal{T}, \mathcal{E}, \mathcal{V} \rangle$ with a factor α will create a new scenario $\langle \mathcal{T}, \mathcal{E}', \mathcal{V}' \rangle$ where $|\mathcal{V}'| = |\mathcal{V}| \cdot \alpha$ and $|\mathcal{E}'| = |\mathcal{E}| \cdot \alpha$. To compute the objective value per order, the global objective value needs to be divided by the number of orders.

3.4 Dataset

This section describes the construction of the scenario generator that creates scenarios with specific levels of dynamism, urgency and scale. Using the scenario generator a benchmark dataset is constructed.

3.4.1 Scenario generator

To create a scenario generator capable of generating scenarios with specific levels of scale, dynamism, and urgency we adapted the generator developed in Chapter 2.

Controlling dynamism of time series

Based on Chapter 2 we assigned a time series generator method to a specific range of dynamism levels such that the entire range $[0, 1]$ is covered (Table 3.1).

Table 3.1: Overview of dynamism ranges and the corresponding time series generator used for generating scenarios in that range.

Dynamism range	**Time series generator**
$[0, .475)$	non-homogeneous Poisson process
$[.475, .575)$	homogeneous Poisson process
$[.575, .675)$	Normal distribution
$[.675, 1]$	Uniform distribution

The non-homogeneous Poisson process that is used for $[0, .475)$ has an intensity function based on a sine wave with the following parameters:

$$\lambda(t) = a \cdot \sin(t \cdot f \cdot 2\pi - \pi \cdot p) + h \tag{3.15}$$

$$a = 1 \qquad \text{amplitude} \tag{3.16}$$

$$f = 1 \qquad \text{frequency} \tag{3.17}$$

$$p \sim \mathcal{U}(0, 1) \qquad \text{phase shift} \tag{3.18}$$

$$h \sim \mathcal{U}(-.99, 1.5) \qquad \text{height} \tag{3.19}$$

In order to keep the total number of events constant with different levels of dynamism, the amplitude and height parameters are rescaled such that the total area under the intensity function equals $|\mathcal{E}|$.

For the $[.475, .575)$ range we used the homogeneous Poisson process, with the (constant) intensity function defined as:

$$\lambda(t) = \frac{|\mathcal{E}|}{\mathcal{T}} = 30 \tag{3.20}$$

The normal distribution for the $[.575, .676)$ range is the truncated normal distribution $\mathcal{N}\left(\frac{\mathcal{T}}{|\mathcal{E}|}, 0.04\right)$ with a lower bound of 0 and a standard deviation of 0.04. If a value x was drawn such that $x < 0$, a new number was drawn from the distribution. Truncating a normal distribution actually shifts the mean, hence the mean was rescaled to make sure the effective mean was equal to $\frac{\mathcal{T}}{|\mathcal{E}|}$.

In the $[.675, 1]$ range a uniform distribution with mean $\frac{\mathcal{T}}{|\mathcal{E}|}$ and a maximum deviation from the mean, σ, is used. The σ value is (for each scenario again) drawn from the truncated normal distribution $\mathcal{N}(1, 1)$ with bounds $[0, 15]$. If a value σ is obtained from the distribution such that $\sigma > 15$ or $\sigma < 0$ a new value is drawn. Since the mean is not scaled, the effective mean of σ is higher than 1.

Generating comparable scenarios with different dynamism, urgency and scale levels

The generator should be able to generate a set of scenarios where all settings are the same except for dynamism, urgency and scale levels. Also, any interactions between variables should be minimized, e.g. dynamism should not correlate with time window intervals. This ensures that any effect measured is solely caused by the difference in dynamism, urgency and or scale.

Because the dataset generator is stochastic, the number of events $|\mathcal{E}|$ and the degree of dynamism of a scenario can not be directly controlled. To construct a consistent dataset, scenarios that do not have exactly $|\mathcal{E}|$ events are rejected. For each desired dynamism level a bin with an acceptable deviation is defined, only generated scenarios with a dynamism value that lies within a bin are accepted.

We further define the concept of *office hours* as the period $[0, \mathcal{O})$ in which new orders are accepted. To ensure feasibility of individual orders we need to take into account the travel time, service time durations and urgency:

$$\mathcal{O} = \mathcal{T} - pst_{max} - dst_{max} - \begin{cases} 2 \cdot tt_{max} & \text{if } u < \frac{1}{2} \cdot tt_{max} \\ 1\frac{1}{2} \cdot tt_{max} - u & \text{otherwise} \end{cases} \tag{3.21}$$

Here, pst_{max} and dst_{max} are the maximum pickup and delivery service times respectively, tt_{max} is the maximum travel time between a pickup and delivery location, and u is urgency.

The pickup and delivery time windows have to be randomly chosen while respecting the constraints as set out by the urgency level and the announce time. The p_i^R is defined as the sum of a_i and u, hence it follows that p_i^L needs to be between a_i and the sum of a_i and u:

$$p_i^L = \begin{cases} \sim \mathcal{U}\big(a_i, p_i^R - 10\big) & \text{if } u > 10 \\ a_i & \text{otherwise} \end{cases} \tag{3.22}$$

Here, 10 is the minimum pickup time window length unless urgency is less than 10, in that case the urgency level equals the pickup time window length. The upper bound of d_i^R can be defined as:

$$ubd_i^R = \mathcal{T} - tt(dloc_i, depot_{loc}) - dst_i \tag{3.23}$$

This translates as the latest possible time to start the delivery operation such that the delivery time window constraints are met and the vehicle can still be back at the depot on time. The lower bound of d_i^L was already defined in eq. 3.8:

$$lbd_i^L = p_i^L + pst_i + tt_i \tag{3.24}$$

We define a minimum delivery time window length of 10, which then results in an upper bound of d_i^L:

$$ubd_i^L = ubd_i^R - 10 \tag{3.25}$$

Based on these bounds we draw the opening of the delivery time window from the following uniform distribution:

$$d_i^L \sim \mathcal{U}\big(lbd_i^L, \max\big(lbd_i^L, ubd_i^L\big)\big) \tag{3.26}$$

To find d_i^R we need to redefine the lower bound (from eq. 3.7) by using the actual value of d_i^L:

$$lbd_i^R = \min\big(\max\big(p_i^R + pst_i + tt_i, d_i^L + 10\big), ubd_i^R\big) \tag{3.27}$$

Finally, the closing of the delivery time window is defined as:

$$d_i^R \sim \mathcal{U}\big(lbd_i^R, ubd_i^R\big) \tag{3.28}$$

For the pickup and delivery service times we choose $pst_i = dst_i = 5$ minutes.

All locations in a scenario are points on the Euclidean plane. It has a size of 10 by 10 kilometer with a depot at the center of this square. Vehicles start at the depot and have a constant travel speed of 50 km/h. All pickup and delivery locations are drawn from a two dimensional uniform distribution $\mathcal{U}_2(0, 10)$.

3.4.2 Benchmark dataset

The benchmark dataset that we created for this chapter has three levels for each of the dimensions of interest resulting in a total of $3 \cdot 3 \cdot 3 = 27$ scenario categories. The dimensions of interest are dynamism, urgency and scale, the used values are listed in Table 3.2a, the other parameters are listed in Table 3.2b. Since the generation of the order arrival times is a stochastic process the exact

Table 3.2: Overview of the parameters used to generate the benchmark dataset.

(a) Dimensions

Dimension	**Values**		
Dynamism	.2	.5	.8
Urgency	5	20	35
Scale	1	5	10

(b) Settings

Parameter	**Value**
$\mathcal{T}$	8 hours
$\lvert\mathcal{E}\rvert$	scale $\cdot$ 240
$\lvert\mathcal{V}\rvert$	scale $\cdot$ 10

degree of dynamism can not be controlled. Therefore, we define a dynamism bin using a radius of 1% around each dynamism value. For this dataset, we consider a scenario with dynamism d where $b - .01 < d < b + .01$ to have dynamism b, where b is one of the dynamism bins listed in Table 3.2a.

For each scenario category 50 instances are generated, resulting in a total of $50 \cdot 27 = 1350$ scenarios. Each scenario is written to a separate file with the following name format: `dynamism-urgency-scale-id.scen`, for example `0.20-5-1.00-0.scen` depicts a scenario with 20% dynamism, an urgency level of 5 minutes, a scale of 1 and id 0. This format allows easy selection of a subset of the dataset. The scenario file contains the entire scenario in JavaScript Object Notation (JSON). Time in a scenario is expressed in milliseconds, distance in

kilometer and speed in kilometer per hour. A scenario is considered to be finished when all vehicles are back at the depot and the current time is $\geq \mathcal{T}$.

The open source discrete time simulator RinSim (van Lon & Holvoet, 2012) version 4.0.0 (van Lon, 2015e) has native support for the scenario format. With RinSim it is easy to run the scenario with centralized algorithms and multi-agent systems, allowing researchers to only have to focus on their algorithms. For reproducibility, the code of the dataset generator is released (van Lon, 2015c) as well as the dataset scenarios (van Lon, 2015b) and all other code and results (van Lon, 2015a).

3.5 Demonstration

As a demonstration a centralized algorithm is compared with a decentralized multi-agent system on 10 instances of each category in the benchmark dataset, resulting in a total of 270 experiments per approach. For reproducibility, the code and results of this experiment are published on an accompanying web page (van Lon, 2015a)

3.5.1 Heuristics

Just as in Chapter 2 two well known heuristics are used, the cheapest insertion heuristic (Algorithm 3) and the 2-opt optimization procedure (Algorithm 4). Since the 2-opt procedure requires a complete schedule as input, it uses the cheapest insertion heuristic to construct a complete schedule first. Both these algorithms have been used in earlier work for vehicle routing problems (Coslovich et al., 2006; Solomon, 1987).

```
Input: ⟨𝒯, ℰ, 𝒱⟩;                                          /* A scenario as input */
Data: 𝒮;                                                  /* the current schedule or ∅ */
𝒮_best = ∅
foreach e ∈ ℰ, e ∉ 𝒮 do
    /* generate all PDP insertion points in the current schedule:          */
    insertions = generate_insertion_points(𝒮)
    for i ∈ insertions do
        /* construct a new schedule by inserting e at insertion i          */
        𝒮_new = construct(𝒮,e,i)
        if cost(𝒮_new) < cost(𝒮_best) then
            𝒮_best = 𝒮_new
        end
    end
end
```

Algorithm 3: Cheapest insertion heuristic, source code available in (van Lon, 2015d).

```
Input: 𝒮
𝒮_best = 𝒮
swaps = generate_swaps(𝒮)
 foreach e ∈ swaps do
 |   𝒮_new = swap(𝒮,e)
 |   if cost(𝒮_new) < cost(𝒮_best) then
 |   |   𝒮_best = 𝒮_new
 |   end
end
/* If a better schedule has been found, we start another iteration                    */
if 𝒮_best ≠ 𝒮 then
 |   2-opt(𝒮_best)
end
```

Algorithm 4: 2-opt procedure, source code available in (van Lon, 2015d).

3.5.2 Centralized algorithm

Each time a new order is announced the cheapest insertion heuristic is executed to produce a new schedule for the fleet of vehicles. It is assumed that execution of the algorithm is instantaneous with respect to the dynamics of the simulations.

3.5.3 Contract net protocol multi-agent system

The multi-agent system implementation is based on the contract net protocol (CNP) as described by Fischer et al. (1995). For each incoming order an auction is organized, when the auction is finished the order will be assigned to exactly one vehicle. All vehicles always bid on each order, the bid contains an estimate of the additional cost that including the new order in the vehicles assignment would incur. This estimate is computed using the cheapest insertion heuristic as described in Algorithm 3. The vehicle with the lowest bid will win the auction and receive the order. Each vehicle computes a route to visit all its pickup and delivery sites using the 2-opt procedure described in Algorithm 4.

3.5.4 Results and analysis

The results[1] of the experiments are plotted along the dynamism, urgency and scale dimension in Figures 3.3, 3.4 and 3.5 respectively. Although all results indicate that the MAS performs better than the centralized algorithm, the current experiment is too limited to verify the hypotheses posed in this chapter. Instead, we discuss the behavior of both algorithms with respect to the dimensions of interest.

[1] In (van Lon, 2015a) the raw results are published.

Dynamism

Figure 3.3 shows that the level of dynamism has very little influence on the performance of both the MAS and the centralized algorithm. This lack of effect is very consistent among all urgency and scale settings.

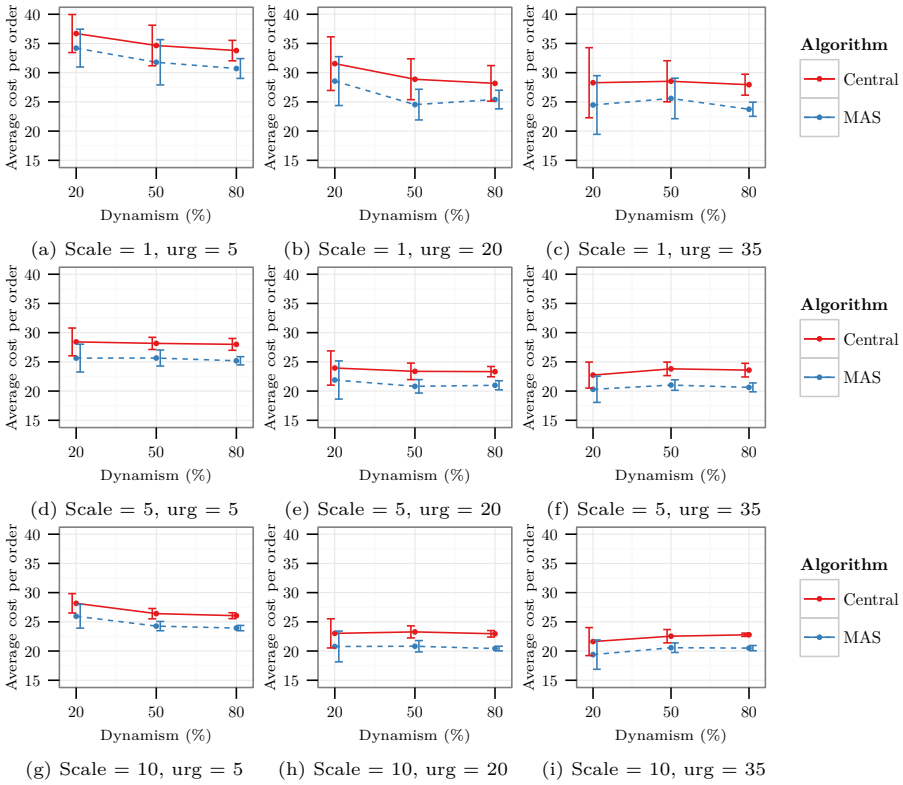


Figure 3.3: Comparison with mean relative cost versus dynamism for all levels of scale and urgency. The error bars indicate one standard deviation around the mean.

Urgency

In Figure 3.4 a clear trend can be observed for both algorithms, the less urgent orders are, the lower the average cost per order is. This effect can be explained by the fact that when orders are less urgent, vehicles have more time to handle other nearby orders first while still respecting the time windows.

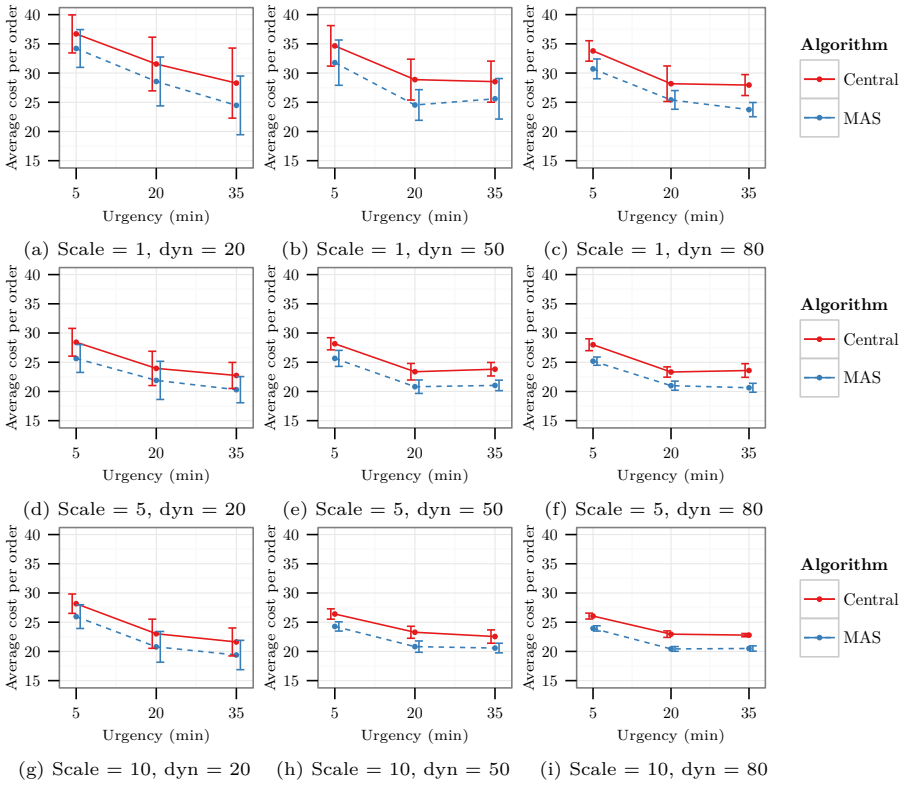


Figure 3.4: Comparison with mean relative cost versus urgency for all levels of scale and dynamism. The error bars indicate one standard deviation around the mean.

Scale

Contrary to what one would expect, Figure 3.5 shows that the larger scale the problem is the lower the average cost of an order. This surprising result can be explained by the fact that computation time is ignored in our current setup, this means that the algorithms have enough time to deal with greater complexity of larger scale problems. The lower average cost per order can be explained by the fact that with more vehicles the average distance of a new order to the closest vehicle is smaller, resulting in reduced average travel times and tardiness.

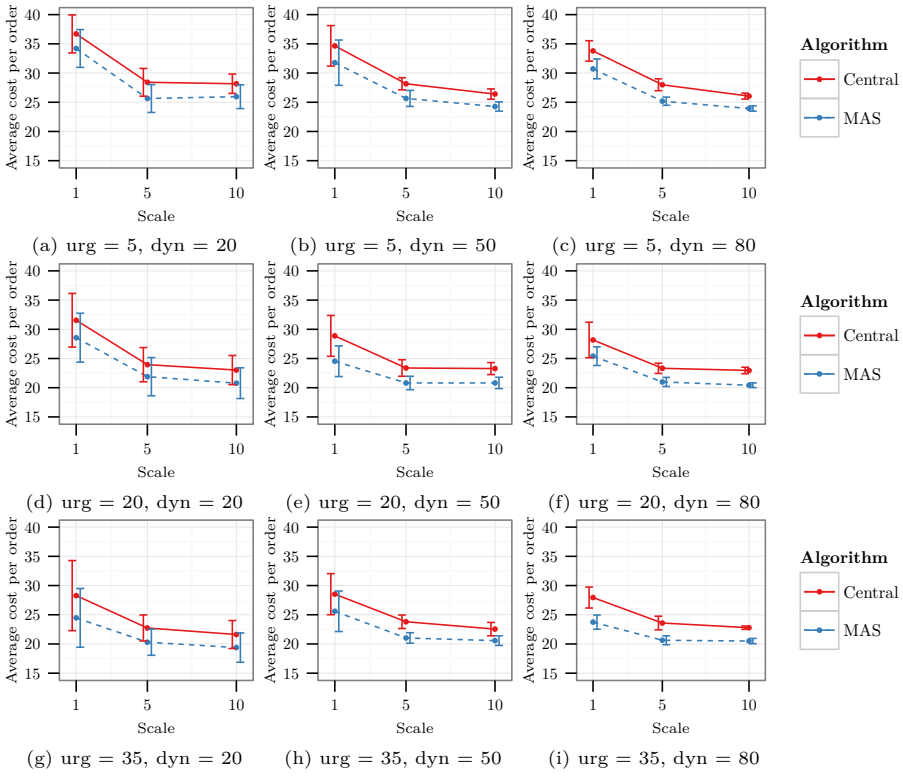


Figure 3.5: Comparison with mean relative cost versus scale for all levels of urgency and dynamism. The error bars indicate one standard deviation around the mean.

3.6 Conclusion

In this chapter we present an open source dataset generator and benchmark dataset instance of dynamic PDPTW with support for varying levels of dynamism, urgency and scale. We demonstrate how to use the benchmark instance to compare a decentralized MAS with a centralized algorithm. Although both algorithms are too basic to generalize upon the results, this demonstration can form a baseline to which future work can compare to. Using the work presented in this chapter, other researchers in the MAS and OR domains are empowered to conduct thorough and systematic evaluations of their work. In our next paper we plan to reap the benefits of this work by extending the comparison demonstration with a state of the art centralized algorithm and an advanced MAS.

Acknowledgements

This research is partially funded by the Research Fund KU Leuven.

Chapter 4

When do agents outperform centralized algorithms? A systematic empirical evaluation in logistics

Using the formal definitions and dataset presented in previous chapters, we systematically evaluate multi-agent systems and centralized algorithms. Additionally, we present a real-time logistics simulator, RinSim, that is used for the evaluation. This chapter contains the paper:

> van Lon, R. R. S. & Holvoet, T. (2017). When do agents outperform centralized algorithms? A systematic empirical evaluation in logistics. *Autonomous Agents and Multi-Agent Systems.* Under review. The first submitted version is published as a technical report (van Lon & Holvoet, 2016)

The content presented in this chapter is the result of discussions of both authors and also based on discussions held during the writing of the previous chapters. Rinde R.S. van Lon programmed the software and wrote the paper, Tom Holvoet provided feedback on the writing.

Abstract

Multi-agent systems (MAS) literature often assumes decentralized MAS to be especially suited for dynamic and large scale problems. In operational research, however, the prevailing paradigm is the use of centralized algorithms. Present chapter empirically evaluates whether a multi-agent system can outperform a centralized algorithm in dynamic and large scale logistics problems. This evaluation is novel in three aspects: 1) to ensure fairness both implementations are subject to the same constraints with respect to hardware resources and software limitations, 2) the implementations are systematically evaluated with varying problem properties, and 3) all code is open source, facilitating reproduction and extension of the experiments. Existing work lacks a systematic evaluation of centralized versus decentralized paradigms due to the absence of a real-time logistics simulator with support for both paradigms and a dataset of problem instances with varying properties. We extended an existing logistics simulator to be able to perform real-time experiments and we use a recent dataset of dynamic pickup-and-delivery problem with time windows instances with varying levels of dynamism, urgency, and scale. The OptaPlanner constraint satisfaction solver is used in a centralized way to compute a global schedule and used as part of a decentralized MAS based on the dynamic contract-net protocol (DynCNET) algorithm. The experiments show that the DynCNET MAS finds solutions with a relatively lower operating cost when a problem has all following three properties: medium to high dynamism, high urgency, and medium to large scale. In these circumstances, the centralized algorithm finds solutions with an average cost of 112.3% of the solutions found by the MAS. However, averaged over all scenario types, the average cost of centralized algorithms is 94.2%. The results indicate that the MAS performs best on very urgent problems that are medium to large scale.

4.1 Introduction

Multi-agent systems (Weiss, 1999; Wooldridge, 2002) is a broad research area involving autonomous software entities, called agents, that typically have a local view of the world. Areas include decentralized control systems, agent based simulation, game theory, trust & reputation, negotiation, etc. In the present chapter we use MAS's as a paradigm for designing decentrally controlled systems. MAS's have been applied in numerous industrial deployments as described by Pěchouček & Mařík (2008). One category of deployments involves OR and logistics. For example, Weyns et al. (2005) describe an application of

MAS technology for operating automated guided vehicles in a warehouse and Dorer & Calisti (2005) describe a MAS for dynamic transport optimization.

The focus in the present chapter is on dynamic logistics, more specifically, on dynamic PDPs (Berbeglia et al., 2010). Although literature reports on various studies on applying MAS's for dynamic PDP, no systematic evaluation has been conducted that allows to draw conclusions of benefits or limitations of MAS approaches compared to centralized approaches. The aim of the present chapter is to systematically evaluate both approaches with varying levels of dynamism, urgency, and scale.

4.1.1 Multi-agent systems related work

Fischer et al. (1995) were one of the first to compare a decentrally controlled MAS with centralized OR heuristics in logistics. In their paper, the authors use a natural mapping of agents to the problem domain, a truck agent is responsible for a single vehicle and a shipping company agent is responsible for handing out new tasks. These agents participate in a dynamic version of CNET first introduced by Smith (1980). Fischer et al. (1995) report that the centralized and the decentralized approaches have similar performance but the decentralized approach performs relatively better when the tasks are more urgent. The authors speculate that this might be a general property of contract-net-like algorithms, but they recognize that this speculation must be confirmed by more empirical experiments, such as the one presented in this chapter.

In a similar spirit, Mes et al. (2007) evaluated an agent-based scheduling approach and look-ahead heuristics for a real-time transportation problem on an underground transport network. In their study, the authors varied several problem properties such as time between orders (related to degree of dynamism), time window length (related to urgency) and the number of nodes in the network (related to scale). The look-ahead heuristics that they used are LocalControl and SerialScheduling. Unfortunately Mes et al. do not specify the exact definitions of the heuristics, hindering the reproducibility of their work. The experimental results show that the agent-based approach always outperforms the look-ahead heuristics. These results are very interesting, especially when considering that MAS's are not used as often in logistics compared to centralized algorithms.

In 2008, Máhr et al. (2008) did a similar comparison but used a mixed integer program (MIP) instead of simple heuristics. The authors used an auction based coordination mechanism similar to CNET. Their results show that the MAS based approach and the MIP based approach perform comparable in dynamic problem instances. However, compared to the present chapter, the problem size used by Máhr et al. is relatively small. The dataset of Chapter 3 that we

use contains instances that are 2 to 18 times larger. Interestingly, Máhr et al. suggest, for future work, to do a similar experiment but on differing problem sizes, which is, among other things, what we do in the present chapter.

In a subsequent work from 2010, Máhr et al. (2010) focused on two types of uncertainty in their problem definition: service time uncertainty and job arrival uncertainty. The results obtained by the authors were mixed. With high service time uncertainty the agent based approach performs better, while in the case of extreme service time combined with job arrival uncertainty the centralized optimization approach outperforms the agent-based approach. However, in the setup by Máhr et al. the urgency of the tasks is variable, it is unclear how this variation influences the result. Therefore, their experiment provides limited insight in the influence of specific problem properties on the effectiveness of centralized and decentralized approaches. This is contrary to the experiments described in the present chapter where we systematically investigate the different problem properties explicitly.

The works described above have several shortcomings that hinder the advancement of the fields of MAS's and OR. Firstly, there is no common platform on which centralized and decentralized algorithms can be tested on logistics problems in real-time with a fair allocation of hardware resources. Such a simulation platform would facilitate evaluations of algorithms from both the MAS's and OR domains, allowing researchers to focus on the improvement of the algorithms while also learning their relative strengths and weaknesses. Secondly, the previously mentioned work did not publish the datasets, algorithms, and supporting code that was used to conduct experiments. It has been argued before by Ince et al. (2012) and van Lon & Holvoet (2013) that this is a problem that needs to be addressed as it would aid reproducibility and extensibility of existing research. Ideally, the opening of source code, data, and related tools should be the default state of practice as this increases the accountability and thus the value of this field of scientific research. Thirdly, to be able to investigate the circumstances for which specific algorithms perform better than another, it is paramount to be able to independently vary specific problem properties. Therefore, exact definitions of the problem properties are required, allowing precise measurements of the properties. These measures can then be used to meticulously create problem instances that vary only in the selected problem property. Unfortunately, the previously cited works did not isolate the relevant properties (for example urgency in (Máhr et al., 2010)), this limits the usefulness of the experimental results with respect to properties in the problem.

4.1.2 Operational research related work

Most of the papers discussed above target a variant of the dynamic PDP. Berbeglia et al. (2010) gave an overview of variants of dynamic PDPs. In this chapter we target the dynamic PDPTW which is a special case of the dynamic VRP. In these problems, dynamism is often caused by the arrival of new tasks (Pillac et al., 2013). At the beginning of a work day, typically only a proportion of tasks is known. In the present chapter we consider a *purely* dynamic PDP, no information about tasks is known beforehand. Therefore it is not possible to plan ahead, all computations have to be done online. In general, there are three different centralized approaches to the PDP: exact methods, heuristics, and stochastic modeling or sampling. Exact methods are known to be less scalable than non-exact methods (Pillac et al., 2013). And, because of the NP-hard nature of PDP, exact methods quickly become infeasible to use. Stochastic modeling or sampling assumes that some a priori information about the future is known, in the present chapter we do not assume to have such information. Therefore we focus our description of centralized approaches on heuristics. Heuristics are capable of quickly finding (sub-optimal) solutions. Gendreau et al. (2006) developed a dynamic version of tabu search with a neighboring structure based on ejection chains. The algorithm runs in between dynamic changes of the problem and when a vehicle has finished a pickup or delivery. Madsen et al. (1995) created an insertion heuristic for a dynamic DARP. Several rolling horizon heuristics were investigated by Yang et al. (2004), with a rolling horizon, only tasks in the near future, within the time horizon, are considered.

4.1.3 Objectives

The goal of the current chapter is to systematically evaluate the performance of a centralized and a decentralized algorithm in a real-time logistics problem. The algorithms guide a cooperative fleet of vehicles to service dynamically appearing customers while minimizing customer waiting times and vehicle travel times. The aim is not to find the best conceivable algorithm but to get insight into the strengths and weaknesses of equivalent centralized and decentralized algorithms under varying circumstances while constrained by the same amount of computational power. We consider a centralized algorithm equivalent to its decentralized counterpart if they use the same underlying solver of problem instances. The method of control, i.e. centralized or decentralized, determines how the solver is used which is the distinguishing difference between the algorithms. Since the two algorithms are constrained by the same amount of computational power, any performance difference measured between the

algorithms can be attributed to their method of control. There are several hypotheses related to the domain of logistics that are of interest for the current chapter:

1. A CNET based MAS finds solutions with a lower operating cost compared to a centralized algorithm on more dynamic problem instances
2. A CNET based MAS finds solutions with a lower operating cost compared to a centralized algorithm on more urgent problem instances
3. A CNET based MAS finds solutions with a lower operating cost compared to centralized algorithms on larger scale problem instances

Operating cost is defined as a combination of customer waiting times and vehicle travel times. To investigate these hypotheses systematically, it is imperative to formally define the concepts of dynamism, urgency, and scale. Dynamism and urgency have recently been defined in the context of dynamic logistics (Chapter 2). In short, dynamism is defined as the continuity of change and urgency is defined as the amount of time that is available to respond to an incoming request. These properties are, together with scale, used in Chapter 3 to define a dataset with varying levels of dynamism, urgency, and scale. The open source logistics simulator RinSim (van Lon & Holvoet, 2012) has support for this dataset, allowing easy comparison of centralized and decentralized algorithms. Present chapter describes how we use this dataset and simulator to investigate the aforementioned hypotheses.

4.1.4 Contributions and overview

The formal problem definition and definitions of dynamism, urgency, and scale as defined by the dataset (Chapter 3) are presented in Section 4.2. The present chapter contributes the following:

- the RinSim simulator is extended such that centralized and decentralized approaches can be compared in a fair manner, each approach receives the same amount of processing power and is subject to the same real-time constraints (Section 4.3);
- an online centralized optimization algorithm and a decentralized dynamic contract-net protocol (DynCNET) that uses the same problem solver, based on the well known OptaPlanner library, are implemented (Section 4.4);

- the centralized and decentralized algorithms are systematically evaluated on differing levels of dynamism, urgency, and scale (Section 4.5); and,
- the code of the simulator, algorithms, and experiments as well as the datasets and results are made available online to allow complete reproducibility and future extension of the present work.

The chapter is concluded in Section 4.6.

4.2 Dynamic pickup-and-delivery problems

We adopt the definition of dynamic PDPs from the dataset described in Chapter 3. In PDPs there is a fleet of vehicles responsible for the pickup-and-delivery of items. Dynamic PDP is an online problem. Customer transportation requests are revealed over time, during the fleet's operating hours. It is further assumed that the fleet of vehicles has no prior knowledge about the total number of requests nor about their locations or time windows. In this section, we provide an overview of the existing work about dynamic PDP and the dataset as it serves as a foundation of the evaluation in present chapter.

4.2.1 Formal definition

In Chapter 3 a *scenario*, which describes the unfolding of a dynamic PDP, is defined as a tuple:

$$\langle \mathcal{T}, \mathcal{E}, \mathcal{V} \rangle := \text{scenario},$$

where

$$\begin{aligned} [0, \mathcal{T}) &:= \text{time frame of the scenario}, & \mathcal{T} > 0 \\ \mathcal{E} &:= \text{list of events}, & |\mathcal{E}| \geq 2 \\ \mathcal{V} &:= \text{set of vehicles}, & |\mathcal{V}| \geq 1 \end{aligned}$$

$[0, \mathcal{T})$ is the period in which the fleet of vehicles $\mathcal{V}$ has to respond to customer requests. The events, $\mathcal{E}$, represent customer transportation requests. Since we consider the purely dynamic PDPTW, all events are revealed between time 0

and time $\mathcal{T}$. Each event $e_i \in \mathcal{E}$ is defined by the following variables:

$$a_i := \text{announce time}$$

$$p_i := [p_i^L, p_i^R) = \text{pickup time window, } p_i^L < p_i^R$$

$$d_i := [d_i^L, d_i^R) = \text{delivery time window, } d_i^L < d_i^R$$

$$pst_i := \text{pickup service time span}$$

$$dst_i := \text{delivery service time span}$$

$$ploc_i := \text{pickup location}$$

$$dloc_i := \text{delivery location}$$

$$tt_i := \text{travel time from pickup location to delivery location}$$

Reaction time is defined as:

$$r_i := p_i^R - a_i = \text{reaction time} \tag{4.1}$$

The time window related variables of a transportation request are visualized in Figure 4.1.

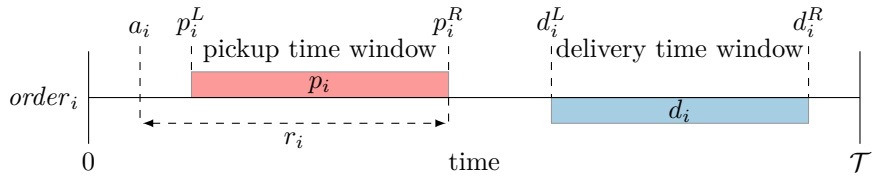


Figure 4.1: Visualization of the time related variables of a single order event $e_i \in \mathcal{E}$.

Furthermore it is assumed that:

- vehicles start at a depot and have to return after all orders are handled;
- the fleet of vehicles $\mathcal{V}$ is homogeneous;
- the cargo capacity of vehicles is infinite (e.g. courier service);
- the vehicle is either stationary or driving at a constant speed;
- vehicle diversion is allowed, this means that a vehicle is allowed to divert from its destination at any time;

- vehicle fuel is infinite and driver fatigue is not an issue;
- the scenario is completed when all pickup and deliveries have been made and all vehicles have returned to the depot; and,
- each location can be reached from any other location.

Vehicle schedules are subject to both hard and soft constraints. The opening of time windows is a hard constraint, hence vehicles need to adhere to these:

$$sp_i \geq p_i^L \tag{4.2}$$

$$sd_i \geq d_i^L \tag{4.3}$$

sp_i is the start of the pickup operation of order event e_i by a vehicle; similarly, sd_i is the start of the delivery operation of order event e_i by a vehicle. The time window closing (p_i^R and d_i^R) is a soft constraint incorporated into the objective function, it computes the operating cost and needs to be minimized:

$$min := \sum_{j \in \mathcal{V}} \left(vtt_j + td\left\{bd_j, \mathcal{T}\right\}\right) + \sum_{i \in \mathcal{E}} \left(td\left\{sp_i, p_i^R\right\} + td\left\{sd_i, d_i^R\right\}\right) \tag{4.4}$$

where

$$td\left\{\alpha, \beta\right\} := max\left\{0, \alpha - \beta\right\} \;=\; \text{tardiness} \tag{4.5}$$

vtt_j is the total travel time of vehicle v_j; bd_j is the time at which vehicle v_j is back at the depot. In summary, the objective function computes the total vehicle travel time, the tardiness of vehicles returning to the depot and the total pickup and delivery tardiness.

4.2.2 Dataset

Earlier work has argued for, and presented, a dataset characterized by three different properties of dynamic PDPs: dynamism, urgency, and scale (Chapter 3).

Dynamism

Dynamism is defined in Chapter 2. Informally, a scenario that changes continuously is said to be dynamic while a scenario that changes occasionally is said to be less dynamic. In the context of PDPTWs a change is an event

that introduces additional information to the problem, such as the events in $\mathcal{E}$. Formally, the degree of dynamism, or the continuity of change, is defined as:

$$dynamism := 1 - \frac{\sum_{i=0}^{|\Delta|} \sigma_i}{\sum_{i=0}^{|\Delta|} \bar{\sigma}_i} \tag{4.6}$$

Δ is the list of event interarrival times:

$$\Delta := \{\delta_0, \delta_1, \ldots, \delta_{|\mathcal{E}|-2}\} = \{a_j - a_i | j = i + 1 \wedge \forall a_i, a_j \in \mathcal{E}\} \tag{4.7}$$

The interarrival time for a scenario with 100% dynamism is called the perfect interarrival time:

$$\theta := \text{perfect interarrival time} = \frac{\mathcal{T}}{|\mathcal{E}|} \tag{4.8}$$

Based on this definition, the deviation and maximum possible deviation to the perfect interarrival time can be computed:

$$\sigma_i := \begin{cases} \theta - \delta_i & \text{if } i = 0 \text{ and } \delta_i < \theta \\ \theta - \delta_i + \dfrac{\theta - \delta_i}{\theta} \times \sigma_{i-1} & \text{if } i > 0 \text{ and } \delta_i < \theta \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

$$\bar{\sigma}_i := \theta + \begin{cases} \dfrac{\theta - \delta_i}{\theta} \times \sigma_{i-1} & \text{if } i > 0 \text{ and } \delta_i < \theta \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

eq. 4.6 uses the proportion of the actual deviation and the maximum possible deviation. Using this definition the degree of dynamism of any scenario can be computed.

Urgency

In Chapter 2 urgency is defined as the maximum reaction time available to the fleet of vehicles in order to respond to an incoming order. Or more formally:

$$urgency\,(e_i) := p_i^R - a_i = r_i \tag{4.11}$$

To obtain the urgency of an entire scenario the mean and standard deviation of the urgency of all orders can be computed.

Scale

Scale is defined in Chapter 3 as maintaining a fixed objective value per order while scaling the number of orders up in proportion to the number of vehicles in the fleet. Scaling up a scenario $\langle \mathcal{T}, \mathcal{E}, \mathcal{V} \rangle$ with a factor α will create a new scenario $\langle \mathcal{T}, \mathcal{E}', \mathcal{V}' \rangle$ where $|\mathcal{V}'| = |\mathcal{V}| \cdot \alpha$ and $|\mathcal{E}'| = |\mathcal{E}| \cdot \alpha$.

4.3 A realistic experimentation platform

The dataset presented in Chapter 3 uses the RinSim logistics simulator (van Lon & Holvoet, 2012). In the present chapter we intend to quantify the performance of algorithms on scenarios with different properties. Dynamism and urgency are both time related properties, that, in the real world, have a direct impact on the amount of available computation time before an action is required. Scale, on the other hand, is a property that impacts the solution space. Since the dynamic PDP is NP-hard, the problem scale has a significant impact on the amount of time needed for computations. In order to evaluate the impact of these properties on the performance of the algorithms in dynamic PDPs, it is imperative to execute the algorithms in real-time. In a logistics scenario, this means that while vehicles are driving or performing operations the algorithms can compute in parallel. To support a realistic evaluation of the algorithms on the dataset, the RinSim simulator is extended with real-time support. This section first presents an overview of the RinSim architecture followed by the design and evaluation of the real-time extension.

RinSim is a discrete-time logistics simulator that supports running both centralized algorithms and decentralized multi-agent systems. RinSim is written in Java and has a modular design (Figure 4.2), a `Model` encapsulates a part of a problem domain or algorithm. The simulator can be customized by selecting the models that are used, this allows simulating a wide variety of logistics problems while maximally reusing existing code. RinSim has a number of built-in models.

`TimeModel` is one of the core models in the simulator, it is responsible for simulating the advancing of time. RinSim is an activity-based simulator, a special case of an event-based simulator (Law, 2007). Event-based simulation models a system as it evolves over time, the system state changes at discrete points in time. Events indicate system state changes at specific points in time. Typically, system state changes trigger the scheduling of new events in the future. An advantage of event-based simulation is that time advances are irregular, allowing the simulator to jump over periods with no events. In activity-based simulation, the simulator clock is advanced using a fixed time increment. Any

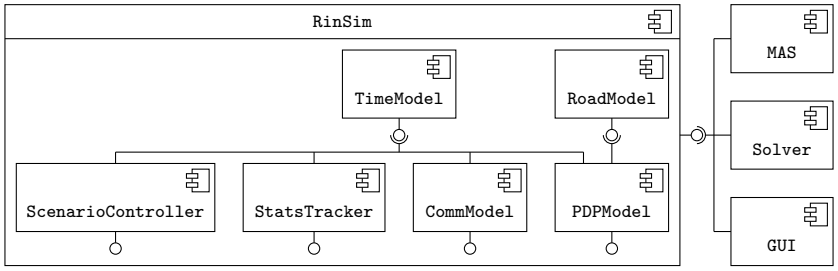


Figure 4.2: UML component diagram of RinSim. The simulator subsystem can be configured with a variety of models that all provide some interface. MASs, solvers, and the graphical user interface use these interfaces to interact with RinSim.

events that are scheduled to occur during this interval are considered to occur at the end of the interval. Therefore, events in activity-based simulations may deviate from their real time. However, the time increment can be chosen to be small enough for these deviations to have no significant effect on the simulations. RinSim is designed with a variety of use cases in mind, one use case is where agents are free to roam around and make ad hoc decisions. When there are many of these agents, the total number of events that need to be scheduled is overwhelming. For that reason, RinSim uses a fixed time increment called a 'tick'. The simulator is initialized with a fixed tick length, for example a tick length of 250 milliseconds. Each tick, RinSim advances the clock and notifies all objects in the simulator that implement the `TickListener` interface (Figure 4.3). The order in which the `TimeModel` notifies the `TickListener`s is

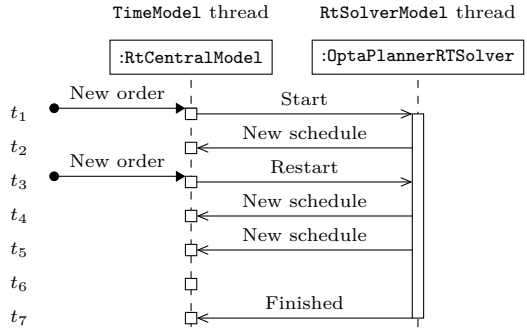


Figure 4.3: Execution order of `TickListener`s in the `TimeModel`.

fixed, this ensures that the simulation is deterministic allowing reproducibility of experiments. Each tick, the `TimeModel` hands out a `TimeLapse` instance that indicates the current time and duration of the tick. Each `TickListener` can

choose to *consume* the `TimeLapse` and spend it on an action. Once a `TimeLapse` is completely consumed, it can not be used again during that same tick. Using this mechanism RinSim ensures time consistency throughout a simulation.

A `RoadModel` provides an interface for traveling on a road structure. RinSim provides several `RoadModel` implementations, there are graph based implementations that allow objects to traverse a graph. Additionally, there are graph based `RoadModel`s that allow dynamic changes to the graph or that have support for collisions, which allows to simulate a warehouse environment for autonomous guided vehicles. Alternatively, there is a `RoadModel` based on a plane (such as used in Chapter 3) which allows vehicles to travel in a straight line. All `RoadModel`s provide location consistency and ensure that maximum speeds are adhered to.

There are several other major components in RinSim. `PDPModel` is a model that provides simulation of pickup-and-delivery of goods by a vehicle. It ensures capacity and location constraints such that a pickup or delivery operation can only be performed when a vehicle is at the correct location. `CommModel` is a model that provides message based communication to agents, `StatsTracker` records statistics of a simulation run and `ScenarioController` allows the simulation of a predefined scenario (such as the scenarios from Chapter 3).

Figure 4.2 also shows several external components. The `MAS` component shows how an agent implementation would interact with the simulator. The `Solver` component has a similar interaction with RinSim but both the `MAS` and `Solver` components provide default implementations to aid in the development of the respective algorithms. The `GUI` component provides the RinSim graphical user interface (Figure 4.4). The RinSim GUI provides several customizable visualizations for different aspects of a simulation.

Besides enforcing consistency inside the simulator models, the code of RinSim itself is meticulously checked by a large number of unit and integration tests (over 1550 tests at the time of writing) to ensure code quality. There are a number of papers reporting on applications of RinSim for scientific experiments. RinSim has been used for simulating bike sharing by Preisler et al. (2015, 2016), and in our research group for experiments with dynamism and urgency (Chapter 2), for experiments on the dataset (Chapter 3), for evolving multi-agent systems for PDPTW (Merlevede et al., 2014; van Lon et al., 2012), and for simulating autonomous guided vehicles in a warehouse (Dinh et al., 2016). Additionally, RinSim is used at KU Leuven, Belgium, as an educational platform for students in the context of a course on MAS.

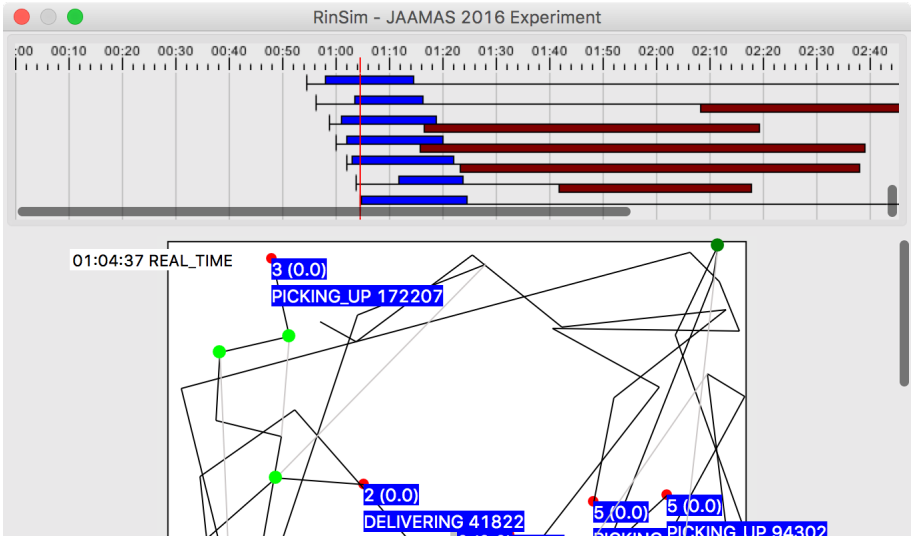


Figure 4.4: Screen shot of RinSim. The top part of the screen shows a time window visualization with pickup time windows above the line in blue and delivery time windows below the line in red. The bottom part of the screen shows a two dimensional geographical view of the simulation world. It shows vehicles (red dots), pickup locations (green dots), and the routes vehicles are following (black lines).

4.3.1 Real-time extension

The standard Java virtual machine (JVM) has no built-in support for real-time execution. However, with a careful software design, the standard JVM can be used to obtain soft real-time behavior. Soft real-time, as opposed to hard real-time, allows occasional deviations from the desired execution timing. In order to obtain acceptable soft real-time behavior, we applied two strategies, first, we minimize the possible deviations from the desired execution timing, and second, we measure and report the actual deviations that occur.

When simulating without real-time constraints, the `TimeModel` will compute all ticks as fast as possible. In a real-time simulator the interval between the *start* of two ticks should be the tick length (e.g. 250 ms). Since the JVM doesn't allow precise control over the timings of threads it is generally impossible to guarantee hard real-time constraints. In real-time mode, RinSim uses a dedicated thread for executing the ticks. If computations need to be done that are expected to last longer than a tick, they must be done in a different thread. RinSim provides a separate model for running solvers in a separate thread called `RtSolverModel`. This minimizes interference of `RtSolverModel`

computations with the advancing of time in the simulated world as executed by the `TimeModel`. Additionally, the processor affinity of the threads are set at the operating system level. Setting the processor affinity to a Java thread instructs the operating system to use one processor exclusively for executing that thread. In practice, the actual scheduling of threads on processors depends on the number of available processors and the operating system. Informal tests on a multi core processor running Linux have shown that different threads are indeed run on different processor cores, exactly as specified. By setting the processor affinity of the `TimeModel` thread, deviations from the desired execution timing are minimized.

Nevertheless, time deviations can and do happen because the behavior of the standard JVM can not be controlled completely. In order to be able to measure the possible deviations, RinSim keeps a real-time tick log (Figure 4.5). In this

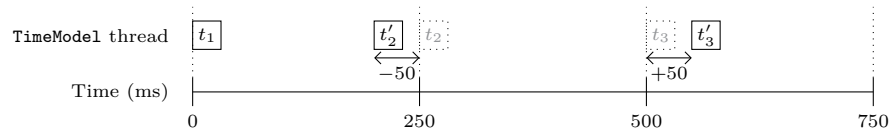


Figure 4.5: Illustration of the execution of real-time ticks. In this example tick t'_2 is executed 50 ms earlier than the perfect timing as indicated by tick t_2, tick t'_3 is executed 50 ms later than the perfect timing as indicated by tick t_3.

log the exact timings (in nanoseconds) of all real-time ticks are kept. With this log, different runs of the simulator can be compared and possible influence on the results can be investigated.

Running a complete logistics simulation in real-time is time consuming, as it will simulate every tick synchronized with real time. However, depending on the specific simulation that is being run, there may be long intervals where no computations are being done other than that of the simulator advancing time in the simulated world. For this reason, RinSim employs a mechanism to dynamically switch between real-time and simulated time (Figure 4.6). When the simulator is in simulated time, ticks will be executed as fast as possible speeding up the simulation significantly. As soon as a computation needs to be done, the simulator must first switch back to real-time mode before this computation can be started.

When a solver starts computing, it receives a snapshot of the current state of the world and starts optimizing the current schedule using that snapshot. Now, the longer a solver is computing, it becomes increasingly likely that the information with which it started computing is outdated. To avoid keeping outdated information for too long, RinSim provides the facility to keep the solver updated in real-time (Figure 4.7). However, each time the solver thread needs

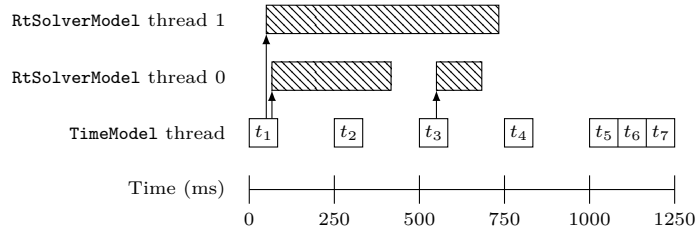


Figure 4.6: An example of RinSim with a `RtSolverModel` with two threads. In tick t_1, two solver computation tasks are dispatched in their own threads. In tick t_4 it is detected that all computations have finished, therefore the simulator switches to simulated time in the next tick. Tick t_5, t_6, and t_7 are executed consecutively in simulated time.

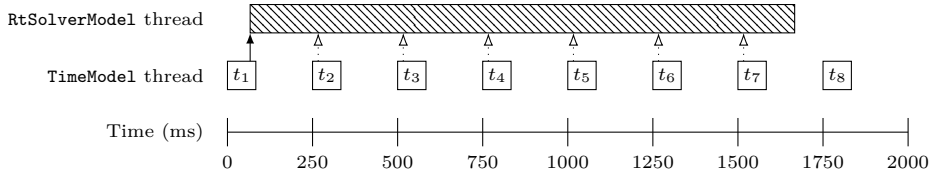


Figure 4.7: Graphical depiction of the `TimeModel` updating the solver on every tick. In tick t_1 the solver is started, subsequent ticks can optionally stop, update, and start the solver.

to be updated the solver has to pause for a short period of time, therefore the number of updates should be limited. To balance between the cost of computing based on outdated information and the cost of interrupting the solver thread, the solver is updated only when the problem has changed in a way that changes the search space significantly. The first event which is considered significant is when a new order arrives. A new order must eventually be assigned, so it is important to take this into account as soon as possible. The second significant event is when a vehicle has committed to perform a specific servicing operation. This is important as it fixes a part of the search space, the order that is being serviced can no longer be moved to another vehicle.

4.3.2 Real-time reliability

Using the log of interarrival times that RinSim keeps, the effect of deviations on the results can be examined. Therefore we did an experiment using three different solvers on the same scenario. We performed 10 repetitions for each algorithm using the same random seed and same scenario. This setup allows to investigate the influence of any deviations of tick interarrival times on the measured scenario cost (using the objective function from Section 4.2.1). The solvers that were used are a cheapest insertion (CI) heuristic, a first fit decreasing

heuristic (FFD), and FFD combined with tabu search (FFD.TABU). Figure 4.8 shows that FFD.TABU outperforms the simpler heuristics but it introduces some variation in the cost. The tick interarrival time logs (Table 4.1) show

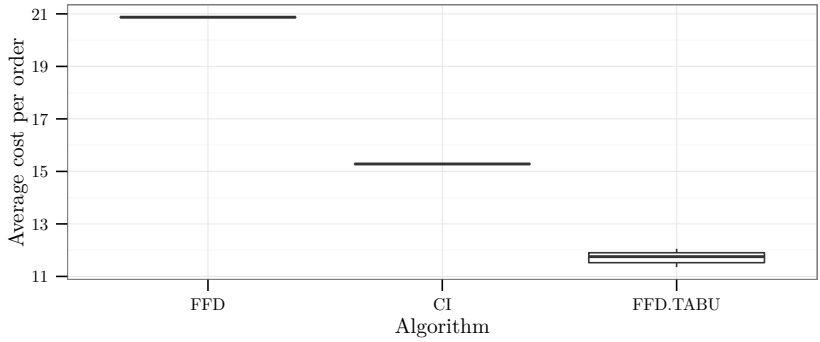


Figure 4.8: Boxplots of three OptaPlanner algorithms on a scenario with 10 repetitions with the same random seed, dynamism of 50%, urgency of 20 minutes, and scale 10. FFD.TABU performs best but the cost values are more stochastic.

that FFD.TABU uses about 10 times more real-time ticks compared to the CI and FFD algorithms. The difference in real-time ticks is expected because

Table 4.1: Accumulated tick interarrival time statistics for the 10 experiments that were conducted for each algorithm. Count indicates the number of real-time ticks, the columns to the right indicate the number of ticks that have a deviation of -10, -1, +1 or +10 milliseconds, respectively, to the expected interarrival time of 250 milliseconds.

Algorithm	μ	σ	Count	-10 ms	-1 ms	$+1$ ms	$+10$ ms
CI	250.0493 ms	2.85 ms	34950	23	168	231	36
FFD	250.0428 ms	3.44 ms	34950	26	193	278	40
FFD.TABU	250.0004 ms	1.73 ms	373055	198	3076	3086	198

FFD.TABU is more complex and therefore requires more computation time. The variation in cost of FFD.TABU is caused by the small variations of tick length that cause small differences in reached solution quality at a specific time in the simulation. For example, when comparing two simulation runs, a deviation of a single tick may already have an impact on the final result. Consider the situation where due to the deviation of a tick of a few milliseconds a new solution is found by the algorithm one tick later (or earlier), this causes the vehicles to receive the new solution one tick (250 ms) later (or earlier). These small differences may have relatively large effects because the costs that are incurred accumulate over time. Therefore, to minimize the effect of this

real-time related stochasticity, a scientific experiment should never rely on a single repetition of a simulation. For that reason, we repeat each simulation three times and we use ten scenarios with the same properties in our experiment setup (Section 4.5.1).

4.3.3 Computational fairness

When comparing centralized algorithms with decentralized MAS's in a real-time setting, it is important that assignment of computational resources is balanced. For example, both approaches must have the same number of available processor cores, but not necessarily the same number of threads. For this reason, the `RtSolverModel` has a thread grouping option, this binds all solver threads to the same processor core (using processor affinity). When more than one thread is bound to the same processor core, the execution of the threads are interleaved, giving a similar percentage of computation time to each thread. Even though a MAS is typically deployed in a distributed fashion and has therefore access to many processor cores, in the experiments described in this chapter the hardware constraints are balanced because the goal is to evaluate centralized and decentralized software paradigms, and not their deployments. In a real-world deployment the hardware constraints of centralized and decentralized approaches most certainly differ from our simulation setup, however, there will invariably be *some* hardware constraints. Therefore, we compare the centralized and decentralized software paradigms irrespective of deployment related hardware constraints.

4.3.4 Experiments

RinSim has several features to ease running large scale experiments with a lot of individual real-time simulations. RinSim can run multiple simulations in parallel, by giving each simulation its own dedicated set of processor cores, simulations do not affect each other's computational resources. However, this also puts an upper bound to the number of simulations that can be run in parallel. For example, when twelve cores are available and each simulation requires two cores, the maximum number of simulations that can be run in parallel is five. These five simulations will use ten cores, so that leaves two cores for the operating system to perform background tasks. When an experiment contains more simulations than can be run in parallel, the remainder will be queued by RinSim.

The standard JVM performs just-in-time compilation and adaptive optimization of often used code. These JVM activities can influence the real-time experiments.

Therefore, RinSim provides a warm up period that runs several simulations for a predefined time to warm up the JVM. This warm up period reduces the influence on the simulations because the JVM will already be optimized to the code that is going to be run. When running real-time experiments it is recommended to always use a warm up period, as we do (Section 4.5.1). RinSim also has an option to change the ordering in which individual simulations are run. For example, when two different configurations are tested, it is better to alternate between the configurations instead of first running all simulations with one configuration and then all simulations with another configuration. Alternating the configurations ensures that the individual configurations are subject to similar fluctuations in computation speed and memory availability that are beyond the control of the JVM.

4.4 Algorithms

To evaluate centralized and decentralized algorithms it is important that the quality of the algorithms is comparable. Comparing a strong centralized algorithm with a weak MAS will not yield meaningful results. For that reason, we use the same solver algorithms framework in both the centralized as well as the decentralized setting. The evaluation focuses on how the algorithms are used, not on the specific algorithms that are used. For the centralized algorithm we have chosen the well known OptaPlanner framework created by De Smet et al. (2016). For the decentralized algorithm we have chosen a multi-agent system with DynCNET because it is a proven technology that has been applied numerous times for task allocation optimization in the context of manufacturing (Shen et al., 2006). DynCNET uses the same solver from the OptaPlanner framework, but in a decentralized fashion.

4.4.1 Centralized algorithm

OptaPlanner (De Smet et al., 2016) is an open source Java constraint satisfaction engine that optimizes planning problems. The project is developed by De Smet et al. (2016) and sponsored by RedHat. OptaPlanner provides a wide range of optimization algorithms such as construction heuristics and metaheuristics. It has support for many problem domains such as scheduling and vehicle routing.

OptaPlanner allows customization of the problem definition to that of the PDPTW (Section 4.2) as is used in the dataset. OptaPlanner is incorporated into RinSim using the `RtCentralModel` and `OptaPlannerRtSolver` classes, the model controls all vehicles centrally using the schedule computed by the

solver (Figure 4.9). The `OptaPlannerRtSolver` continuously updates the `RtCentralModel` of its progress and the model restarts the solver when the problem definition changes (i.e. when a new order is announced). The continuous

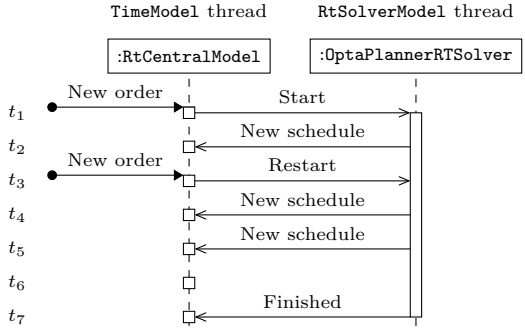


Figure 4.9: UML interaction diagram of communication between `RtCentralModel` and `OptaPlannerRtSolver`. In this example the solver is interrupted at t_3, the problem definition is updated with the new order after which the solver continues its search. In t_4 and t_5 it has found a new improving schedule. In t_7 the solver is done computing as it couldn't find an improving schedule anymore.

updates of the solver use the RinSim mechanism as described in Section 4.3. This setup allows real-time control of the fleet of vehicles and avoids unnecessary long computations based on outdated information.

OptaPlanner requires a stop condition to halt its search process. We are using the *unimproved time spent* stop condition, that halts the search when the best score has not improved for an amount of time.

4.4.2 Decentralized multi-agent system

The multi-agent system that is investigated is an implementation of the DynCNET presented by Weyns et al. (2007). DynCNET is a dynamic extension of the CNET first proposed by Smith (1980). Inspired by how companies use subcontracting to collaboratively solve problems, CNET uses contracting to approach the task assignment problem. In CNET, the agent that tenders a task is called the *manager* and it sends a task announcement to potential *contractors*. Each potential contractor can either ignore the announcement or send a *bid* to the manager. The manager then selects its best bid and *awards* the task to the contractor. Figure 4.10 shows the UML interaction diagram for the CNET auction process. Although an auction can be, and usually is, used in a competitive setting, we use auctions in a purely cooperative setting. We assume that both the contractors and the manager are working for the same company.

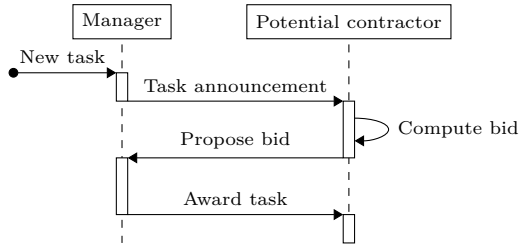


Figure 4.10: UML interaction diagram of a CNET auction.

The dynamic extension of CNET provides flexibility to the assignment until a contractor has to commit to the execution of the task. The same task can be announced several times before its execution, its assignment changing after every announcement.

In our MAS implementation for the dynamic PDPTW, both the vehicle as well as the transportation requests are modeled as agents. In the remainder of this text we will call the agent controlling a vehicle a `VehicleAgent` and the agent responsible for a transportation request an `OrderAgent`. `OrderAgent`s are playing the role of the manager in DynCNET, `VehicleAgent`s are the potential contractors. Figure 4.11 shows an interaction diagram of an auction using our DynCNET implementation. At the end of an auction, each `VehicleAgent` is either awarded the order or notified of the end of the auction. At this moment the `VehicleAgent`s have the possibility of starting a new auction by offering one of their previously awarded orders. The `VehicleAgent` will inform the `OrderAgent` responsible for the order that is to be offered to start a new auction, the `OrderAgent` will then perform a new auction process similar to Figure 4.11. A possible outcome of this auction is that the order is not awarded to another vehicle but stays assigned to the original vehicle. Allowing the vehicles to start a new auction process enables the dynamic (re)allocation of orders and makes our CNET implementation dynamic.

Order agent

The `OrderAgent` (the manager in CNET terminology) is responsible for the auction process. It announces the start of the auction to all vehicles and waits until it receives enough bids to make a decision. The stop criterion for the bidding process is:

$$|bids| \geq 2 \wedge (|bids| = |vehicles| \vee auction\_duration \geq max\_auction\_duration) \tag{4.12}$$

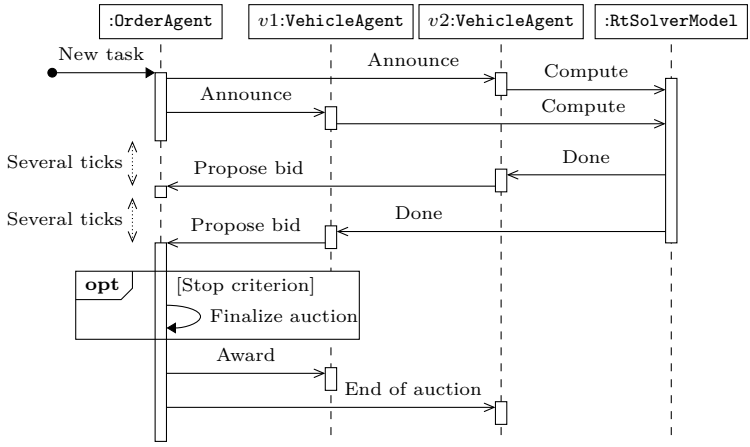


Figure 4.11: UML interaction diagram of an auction of an order with two vehicles. Upon receiving the auction announcement, both `VehicleAgent`s start computing a bid. The computations take several ticks. As soon as the `OrderAgent` has met the stop criterion, in this case receiving two bids is enough, the auction is finalized and the order is awarded to $v1$. Vehicle $v2$ is notified of the end of the auction. The `RtSolverModel` lifeline is a simplified view of the implementation, the actual computations can be performed in multiple threads (as discussed in Section 4.3). Note that the filled arrows indicate synchronous calls and the stick arrows indicate asynchronous calls.

where, $|bids|$ is the number of received bids, $|vehicles|$ is the total number of vehicles which equals the potential maximum number of bids and $max\_auction\_duration$ is a duration limit that is a parameter of the MAS. An alternative stop criterion would be to wait until all vehicles have submitted their bid, but we use a maximum auction duration because it is more robust in case of (partial) communication loss, vehicle failures, and other unforeseen events.

When the stop criterion evaluates to *true*, the `OrderAgent` finalizes the auction by selecting the best bid as the winner. The best bid is defined as the bid with the lowest price (cost). The order is assigned to the winner, the winner must therefore service that order, unless it decides to auction it and somebody else wins that auction at a later time. All `VehicleAgent`s are informed of the end of the auction. This allows agents that are still computing their bids for this auction to cancel their computations. Bids that are received after the finalization of the auction are ignored.

Vehicle agent

A `VehicleAgent` needs to compute a bid value in order to propose a bid. In our implementation the bids are computed using a solver (the bid solver) managed by the `RtSolverModel`. The cost of an order is defined as the additional cost that including that order incurs to a vehicle's current schedule:

$$cost(order) = cost(new\_schedule) - cost(current\_schedule) \tag{4.13}$$

where, *current_schedule* is the schedule of the vehicle including all previous order assignments, and *new_schedule* is the current schedule of the vehicle including the proposed order. The task of the solver is finding the best *new_schedule* in a relatively short amount of time to get a reliable estimate of the cost of the auctioned order. The time for computing the new schedule is limited because the auction process has a limited duration, the bid needs to be proposed before the end of this duration in order to ensure that the `OrderAgent` will take the bid into account.

As soon as the assignment of orders to a vehicle has changed, the `VehicleAgent` needs to update its schedule. The vehicle's schedule is optimized by a solver (the schedule solver), although it is imperative to generate a *complete* schedule quickly (e.g. to respond to urgent requests), the solver can compute for a longer time as the solver can continuously notify the `VehicleAgent` of improved schedules. This allows the optimization process to continue for an extended period.

The `VehicleAgent` considers starting a new auction (a reauction) in the following two situations:

- when a vehicle hasn't won an auction for at least five minutes; or,
- when the vehicle's current schedule has changed.

When starting a new auction the vehicle has to decide which of its previously assigned orders it should auction. The order that when removed yields the greatest schedule cost reduction, for that vehicle, is selected. Computing the cost reduction of removing an order from the current route does not require an optimization step (the route is not optimized again) and can therefore be computed quickly for all orders assigned to a vehicle (similar to eq. 4.13). Orders for which the pickup operation is in process or is already done are not considered for auctioning as they can't be reassigned. If the order with the greatest cost reduction is the last received order, no auction is performed to avoid excessive auctioning. The `VehicleAgent` itself has to propose a bid to its own auction, only when another agent proposes a better bid will the order be reassigned.

4.4.3 Tuning

For tuning we have run the algorithms on the dataset used by Gendreau et al. (2006). This dataset was chosen because it is very similar to the dataset presented in Chapter 3 as the problem definition is nearly identical. Additionally, using the Gendreau et al. dataset allows comparison with their algorithm. The Gendreau et al. dataset consists of 15 scenarios divided in three different scenario classes (Table 4.2). It is worth noting that within a scenario class there

Table 4.2: Characteristics of the three scenario classes of the Gendreau et al. dataset.

Scenario class	Duration	Average request intensity	Fleet size
`4H-24`	4 hours	24 requests per hour	10 vehicles
`4H-33`	4 hours	33 requests per hour	10 vehicles
`7.5H-24`	7.5 hours	24 requests per hour	20 vehicles

is quite some variability of characteristics. For example in `4H-24` the number of orders ranges from 84 to 94, that is an intensity of 21 to 23.5 requests per hour. Similar variability exists in the other scenario classes and also for characteristics such as dynamism and urgency. We performed a benchmark experiment with 28 different OptaPlanner solver configurations. We tested two construction heuristics, first-fit decreasing (FFD) heuristic and cheapest insertion heuristic (CI) and combined each with 14 different local search algorithms. All available local search algorithms provided by OptaPlanner are used with the parameters as advised by the OptaPlanner manual (De Smet et al., 2016). We used a period of ten seconds for the unimproved time spent stop condition, this is a rather long period that we chose to give OptaPlanner enough time for searching. Since OptaPlanner can be interrupted when the problem changes, there is no downside to this long period. When OptaPlanner is interrupted, it remembers its current best solution, inserts the new problem information, and then continues the search. The algorithms were run three times per scenario, each time with a different random seed. This resulted in a total of $28 \cdot 15 \cdot 3 = 1260$ simulations. Table 4.3 displays the most relevant results, the complete overview of results can be found in (van Lon, 2017g). The best Gendreau et al. algorithms outperform the OptaPlanner algorithms for all scenario classes. This is expected because the Gendreau et al. algorithms are designed and optimized specifically for this problem while the OptaPlanner algorithms are generic local search heuristics. However, the results in Table 4.3a to 4.3c indicate that the relative difference between the Gendreau et al. algorithms and the best OptaPlanner algorithm is larger for the small scale scenario (26.4% for `4H-24`) and smaller for the larger scale scenarios (4.9% for `4H-33` and 9% for `7.5H-24`). Since the small scale scenarios in the dataset from Chapter 3 are already larger scale than the scenarios in the `4H-24` class, we expect that the performance difference on the

Table 4.3: Selection of results of the tuning experiment on the Gendreau dataset. Per scenario class, the average cost and rank of a selection of algorithms is shown.

Algorithm	Cost	#
Gendreau	485.6	1
FFD.SHTS	613.8	2
FFD.SA1	614.6	3
FFD.SA4	616.2	4
CI	1023.6	28
FFD	1420.6	29

(a) `4H-24`

Algorithm	Cost	#
Gendreau	3159.4	1
FFD.SHTS	3313.1	2
FFD.LAT	3336.5	3
FFD.SCHC	3337.5	4
CI	5051.7	28
FFD	5778.3	29

(b) `4H-33`

Algorithm	Cost	#
Gendreau	634.8	1
CI.TABU	691.9	2
CI.SA1	693.1	3
FFD.SHTS	702.9	13
CI	1074.4	28
FFD	2656.0	29

(c) `7.5H-24`

Algorithm	Description
Gendreau	The cost values reported are the average of the best algorithm per scenario. The best algorithms proposed by Gendreau et al. (2006) are an adaptive descent algorithm and a tabu search algorithm.
CI	Cheapest insertion construction heuristic.
FFD	First-fit decreasing construction heuristic.
FFD.SHTS	FFD with step counting hill climbing with tabu search and strategic oscillation.
FFD.SA1	FFD with simulated annealing with an accepted count limit of 1.
FFD.SA4	FFD with simulated annealing with an accepted count limit of 4.
FFD.LAT	FFD with late acceptance with tabu search.
FFD.SCHC	FFD with step counting hill climbing.
CI.TABU	CI with tabu search.

(d) Algorithm descriptions, details about the algorithms can be found in (De Smet et al., 2016).

Chapter 3 dataset between the Gendreau et al. algorithms and the OptaPlanner algorithms lies between 5 and 9%. This difference is acceptable for the purpose of the current study: analyzing the performance difference between centralized and decentralized usage of the same algorithm. Based on the results we conclude that FFD.SHTS is the best OptaPlanner algorithm to use in the experiments.

4.4.4 MAS tuning

The MAS has four main parameters that influence the experiment results the most:

- `Bid`, bid solver unimproved time spent.
- `Schedule`, schedule solver unimproved time spent.
- `MAD`, maximum auction duration (eq. 4.12).
- Reauctions, can be enabled, disabled, or enabled with a cooldown period. The cooldown period is defined as the time that a `VehicleAgent` is

not allowed to start a new reauction for an order that was previously reauctioned unsuccessfully.

Choosing the best values for these parameters is especially important because all computational tasks done by agents in the MAS have to be performed on a single core. This means that in large scale scenarios, 100 agents need to compute their bids on the same core. Using the dataset generator of Chapter 3, we created a dataset of large scale scenarios for tuning the MAS. The dataset consists of three levels of dynamism, three levels of urgency, and one level of scale. This gives nine scenario classes, with five scenarios per class, the tuning dataset contains $3 \cdot 3 \cdot 5 = 45$ scenarios. In the first MAS tuning experiment we varied `Bid`. Table 4.4 shows the MAS settings and the results. The cost

Table 4.4: First MAS tuning experiment settings and results. Cost is the cumulative cost of the average cost per scenario class, rank is the average rank of the MAS over the nine scenario classes.

`Bid` (ms)	`Schedule` (ms)	`MAD` (s)	Reauctions	Cost	Rank
1	100	5	Enabled	129.1	6.8
2	100	5	Enabled	113.7	5.4
5	100	5	Enabled	105.0	3.9
8	100	5	Enabled	102.2	2.9
10	100	5	Enabled	101.0	2.3
15	100	5	Enabled	102.9	2.2
25	100	5	Enabled	110.8	4.4

and rank values indicate that the `Bid` values 8, 10, and 15 perform best. When looking at the results for these best settings we found that around 66-93% of the auctions are concluded after receiving bids from all agents. Because we expect that receiving less than all bids is not beneficial, we decided to increase the auction duration to ten seconds.

We designed a second experiment that uses the best `Bid` values from last experiment, adds a `Bid` value of 20, tests a higher `Schedule` value, and uses a longer `MAD`. Table 4.5 shows the settings and the results of the second MAS tuning experiment. The results show that a `Schedule` of 100 milliseconds performs better than a `Schedule` of 250 milliseconds. For `Bid` 8, 10, and 15, with `Schedule` 100, the percentage of auctions that are concluded after receiving bids from all agents has increased to 77-97%. It further appears that increasing `MAD` improves performance of `Bid` 15 while performance of `Bid` 8 and 10 is almost unaffected. We decided to keep the `MAD` at 10, and to select the algorithm with the lowest cost and rank, that is a `Bid` of 15 milliseconds and a `Schedule` of 100 milliseconds.

Table 4.5: Second MAS tuning experiment settings and results. Cost is the cumulative cost of the average cost per scenario class, rank is the average rank of the MAS over the nine scenario classes.

`Bid` (ms)	`Schedule` (ms)	`MAD` (s)	Reauctions	Cost	Rank
8	100	10	Enabled	102.0	5.4
10	100	10	Enabled	101.3	3.9
15	100	10	Enabled	101.3	3.0
20	100	10	Enabled	104.7	4.4
8	250	10	Enabled	102.5	5.6
10	250	10	Enabled	102.1	4.9
15	250	10	Enabled	103.0	3.7
20	250	10	Enabled	106.2	5.1

In the third MAS tuning experiment we investigated the effectiveness of reauctions. Table 4.6 shows the settings and the results of the experiment. As can be seen, the performance difference between enabling and disabling

Table 4.6: Third MAS tuning experiment settings and results. Cost is the cumulative cost of the average cost per scenario class, rank is the average rank of the MAS over the nine scenario classes.

`Bid` (ms)	`Schedule` (ms)	`MAD` (s)	Reauctions	Cost	Rank
15	100	10	Enabled	101.3	3.1
15	100	10	Disabled	103.2	4.1
15	100	10	Cooldown period 1 min.	101.2	2.7
15	100	10	Cooldown period 10 min.	100.8	2.8
15	100	10	Cooldown period 20 min.	100.5	2.3

reauctions are small. Nevertheless, enabling reauctions performs better compared to disabling them. Using a cooldown period seems to be beneficial, additionally, we found that a longer cooldown period results in a lower number of reauctions and a higher reauction success rate (Table 4.7). The slightly

Table 4.7: Third MAS tuning experiment reauction details. Total num. reauctions is the cumulative number of reauctions, success rate is the average reauction success rate per scenario.

Reauctions	Total num. reauctions	Success rate
Enabled	53214	27.2%
Disabled	0	
Cooldown period 1 min.	49533	27.6%
Cooldown period 10 min.	31827	31.8%
Cooldown period 20 min.	26522	32.8%

higher costs when a shorter cooldown period is used appears to be related with the high number of unsuccessful reauctions in this case. We expect that these unsuccessful reauctions and the computations that are involved delay computations for regular auctions, which explains the higher costs in this case. For the main experiments we decided to use a cooldown period of 20 minutes.

4.5 Evaluation

To evaluate the hypotheses about multi-agent systems and centralized algorithms, our implementations (Section 4.4) are run using real-time RinSim (Section 4.3) on the dataset of Chapter 3.

4.5.1 Experiment setup

The dataset of Chapter 3 has three dimensions: dynamism, urgency, and scale, with three values per dimension that results in a total of 27 different scenario classes. We use ten scenarios for each class and we perform three repetitions per scenario (with different random seeds), this results in a total of $27 \cdot 10 \cdot 3 = 810$ simulations per algorithm. For each dimension there is a hypothesis that addresses the performance of the algorithms on that dimension.

The dataset of Chapter 3 contains scenarios with a length of 8 hours. Since we need to do a large number of experiments in real-time we shortened the scenario length to 4 hours. We used the dataset generator to generate a new dataset and made it available online (van Lon, 2017g). The reduction of scenario length was done purely for computational reasons as running such a large number of simulations in real-time takes considerable time. Additionally, the tick size is set to 250 milliseconds and scenarios now require a real-time simulator by default.

Because performance in real-time simulations is hardware dependent, all real-time simulations are performed on the same computer. We used a dedicated Ubuntu machine (version 12.04.5 LTS) with 24 logical cores (two six core Intel Xeon 2.6GHz E5-2630 v2 processors with hyper threading). For the experiments the Java HotSpot 64-Bit Server VM (runtime version: 1.8.0_74-b02) was used. A single simulation requires two logical cores, one for the simulator and one for the solver computations. For the solver computations we used a thread pool of size three, meaning that any additional computations are queued until one of the three threads are available. At least one core needs to be available for the operating system so a maximum number of 11 simulations were run in parallel. Even though no other processes were started on the dedicated computer during

the course of the experiments, we opted to use the experiment ordering feature in RinSim (Section 4.3.4). The factorial setup order that is used is: repetition, scenario, algorithm. This means that the first few simulations are: $r_0 s_0 a_0$, $r_0 s_0 a_1$, $r_0 s_1 a_0$, $r_0 s_1 a_1$, etc. Here r_n stands for random seed (repetition) n, s_n stands for scenario n, and a_n stands for algorithm n. This setup ensures that the execution of the different algorithms is interleaved as much as possible. Additionally, a JVM warm up period of 30 seconds is used to let the JVM perform code optimizations before the actual experiment is started. We choose 30 seconds because this is the default warm up time in OptaPlanner (De Smet et al., 2016). RinSim nor the algorithms make use of any I/O operations during a simulation. The entire experiment is programmed using Java and is run entirely from memory. Before and after a simulation, RinSim does use I/O operations to read the scenario file and write the results to disk, but since these operations are not done during the simulation itself, this does not influence the results.

For performing the experiments described in this section we used the following software. As simulator, we used RinSim v4.3.0 (van Lon, 2016d), for the centralized and decentralized algorithms we used RinLog v3.2.2 (van Lon, 2017a), to generate the scenarios we used the PDPTW dataset generator v1.1.0 (van Lon, 2016b). The experiment code including the launch scripts can be found in a separate repository (van Lon, 2017f). The datasets that were used in the tuning experiments, the main dataset, as well as all results discussed in this chapter, can be found in (van Lon, 2017g). We have made sure to archive each of these artifacts, using a Digital Object Identifier (DOI), to ensure their long-term availability.

Algorithms

Based on the tuning experiment (Section 4.4.3) we selected the FFD.SHTS algorithm that performed best on average on all three scenario classes. Table 4.8 shows how and with what settings FFD.SHTS is used.

Table 4.8: OptaPlanner settings used in the centralized and decentralized configurations. The type refers to how the solver is used, the limit is the maximum unimproved time parameter of the OptaPlanner solver.

Short name	Name	Type	Limit
MAS	DynCNET	Bid	15 ms
		Schedule	100 ms
COP	Centralized OptaPlanner	Schedule	10000 ms

4.5.2 Results and analysis

Figures 4.12-4.17 display the results of the experiments. Each data point in the graphs is the average of 30 simulations, 10 scenarios from the same class, each repeated 3 times. For each two means obtained under the same settings we analyzed whether they are statistically different using Welch's paired t-test. In the following analysis we refer to this test by mentioning the p-values (when relevant) that were observed. The significance threshold was set at $p = .01$. The cumulative computation time of all 1620 simulations was 1794.4 hours ($\approx$ 74.8 days), since 11 experiments were conducted in parallel, the actual computation time was 165.1 hours ($\approx$ 6.9 days).

Dynamism

Recall the first hypothesis (Section 4.1): *A CNET based MAS finds solutions with a lower operating cost compared to a centralized algorithm on more dynamic problem instances.* More specifically, we can refine this hypothesis into several related sub-hypotheses. First, we expect that more dynamism correlates positively with the average costs for MAS. Figure 4.12 shows that the costs of MAS remain stable or decrease for every level of scale and urgency when dynamism increases, therefore this hypothesis is rejected. The cost for MAS always decreases significantly between 20% and 50%, between 50% and 80% the decrease is not significant except in Figure 4.12(i) ($p \approx 0.006$) and in Figures 4.12(e), 4.12(f), and 4.12(g), where the cost is actually increasing. Second, we expect that more dynamism correlates positively with the average costs for COP. This hypothesis is rejected because the costs of COP are significantly decreasing in Figure 4.12 between 20% and 50% dynamism (except in Figure 4.12(c)), between 50% and 80% the costs difference is never significantly different. These results are similar to the results based on simulated time as reported in Chapter 3, this comes as a surprise since we expected that simulating in real-time would make dealing with highly dynamic situations more challenging. However, it turns out that dealing with occasional but relatively big bursts of change is more demanding for the algorithms than more frequent smaller changes. Third, we expect that increasing dynamism increases the cost of MAS less than that it increases the cost of COP. Figure 4.13 shows that in many situations, MAS performs relatively better when dynamism increases. In Figures 4.13(c), 4.13(e), and 4.13(f), however, this is not the case. In very urgent and large scale scenarios, the hypothesis can be accepted because the performance of MAS decreases more relative to COP. This result is contrary to the experiments performed in Chapter 2 and Chapter 3 where dynamism had almost no influence. We attribute this difference in results to the fact that in the present chapter

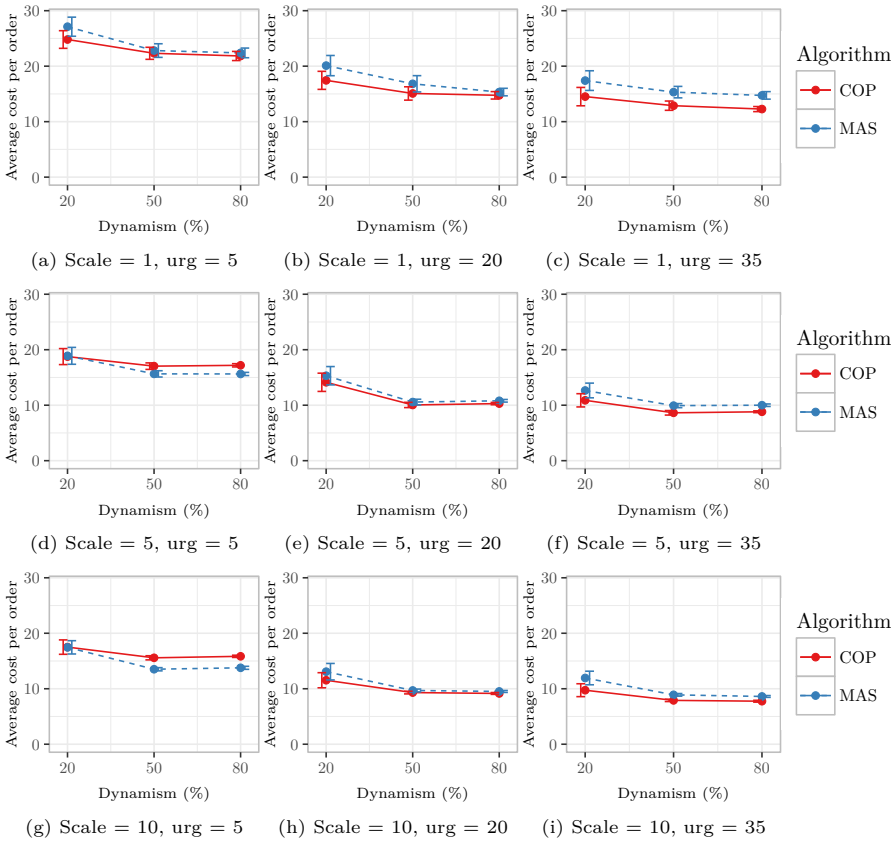


Figure 4.12: Comparison with mean relative cost versus dynamism for all levels of scale and urgency. The error bars indicate a confidence interval of 99%.

the experiments were conducted using real-time simulation, contrary to the simulations in Chapter 2 and Chapter 3. This is confirmed by the fact that this effect only occurs in very urgent and in large scale situations, as these are the conditions where real-time simulation has the biggest impact on the available computation time. In general, we can conclude, based on this experiment, that dynamism influences the *relative* performance of COP and MAS in very urgent and in large scale scenarios. In absolute terms, however, lower dynamism tends to induce higher average costs.

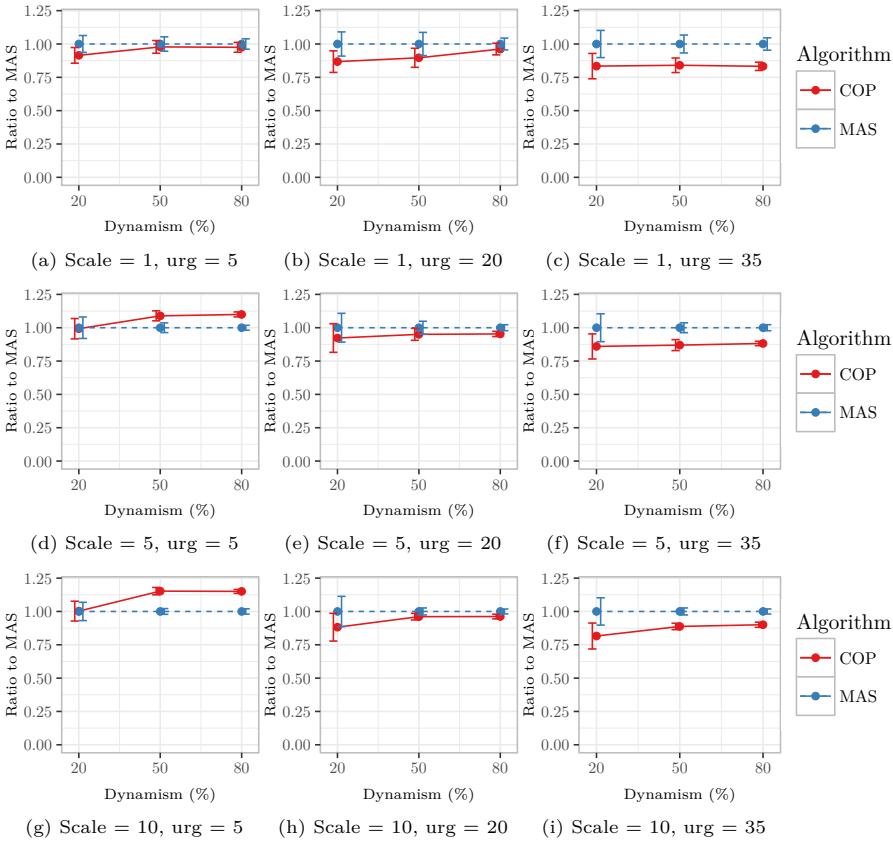


Figure 4.13: Comparison with competitive ratio to MAS versus dynamism for all levels of scale and urgency. The error bars indicate a confidence interval of 99%.

Urgency

The second hypothesis (Section 4.1) concerns urgency: *A CNET based MAS finds solutions with a lower operating cost compared to a centralized algorithm on more urgent problem instances.* Regarding urgency, we expect that more urgent problems (lower urgency values) are correlated with higher average costs per order for MAS and COP. The results (Figure 4.14) show that for both COP and MAS this is true. Between 5 and 20 minutes the difference in cost is always significant. Between 20 and 35 minutes, this is not the case for COP in Figures 4.14(a) and 4.14(g), and for MAS, the differences are not significant between 20 and 35 minutes in Figures 4.14(a), 4.14(b), 4.14(c), 4.14(d), and 4.14(g).

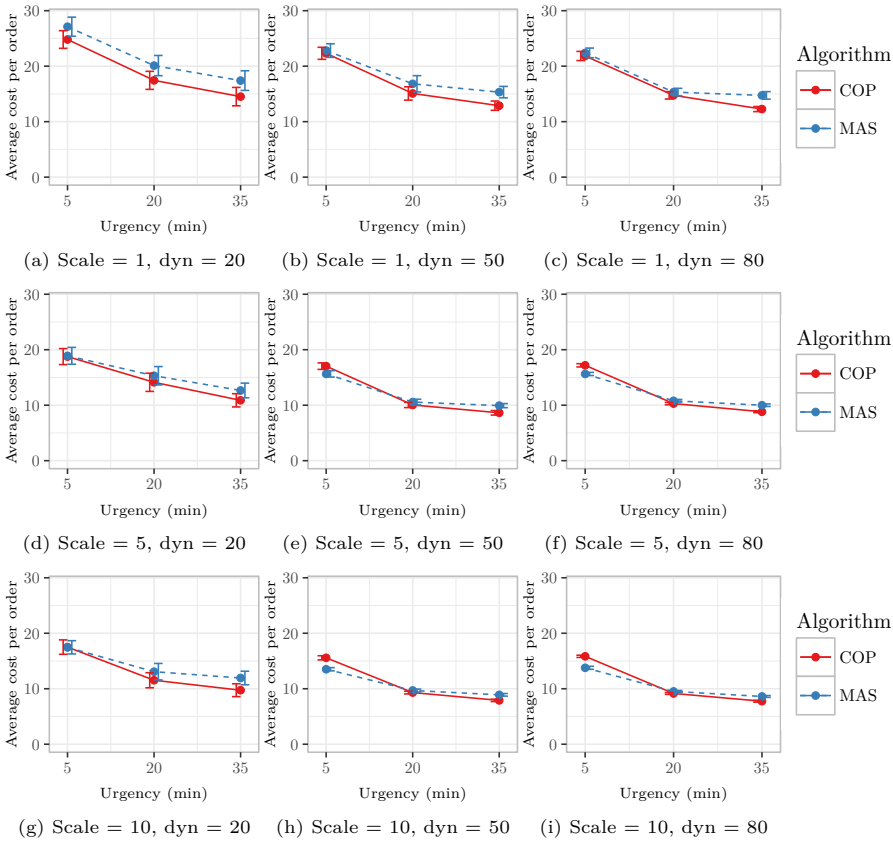
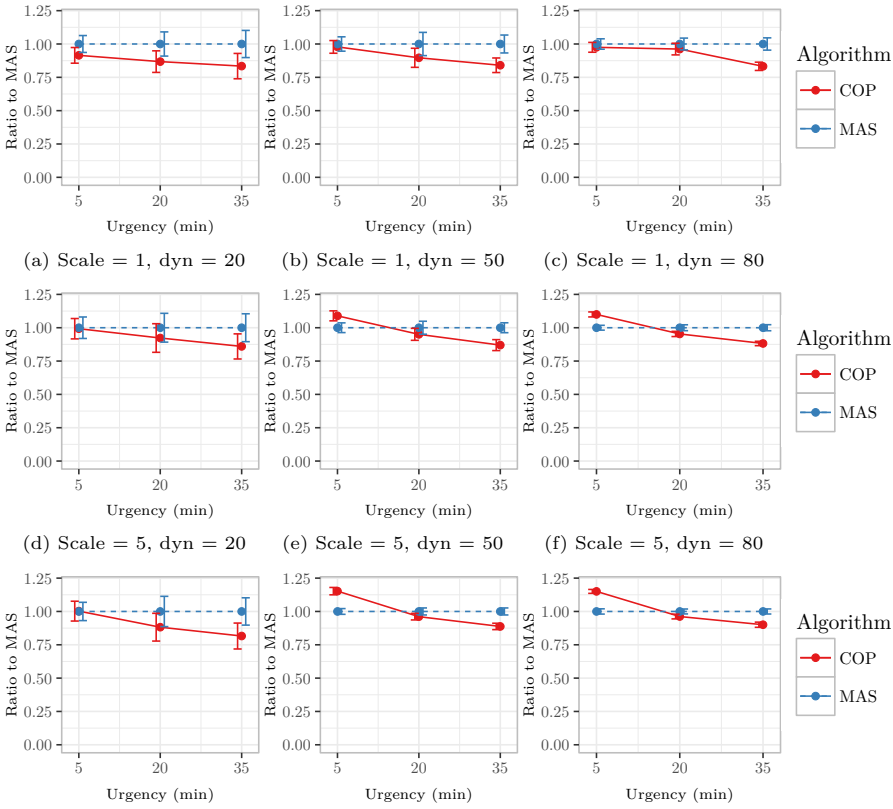


Figure 4.14: Comparison with mean relative cost versus urgency for all levels of scale and dynamism. The error bars indicate a confidence interval of 99%.

These results and the figures show a diminishing of the cost decreases the less urgent a problem gets. However, it appears that this effect is stronger for MAS than for COP. The relative cost of MAS versus COP (Figure 4.15) shows that less urgency (higher values) benefits COP more than it benefits MAS. Similarly, COP suffers to a greater extent from more urgency than MAS. We have to reject the hypothesis, however, because there are only four very urgent cases where MAS outperforms COP significantly (Figures 4.15(e), 4.15(f), 4.15(h), and 4.15(i)). In four other very urgent situations, the costs of MAS and COP are not significantly different (Figures 4.15(b), 4.15(c), 4.15(d), and 4.15(g)). Therefore, MAS is not always better at responding to the most urgent requests. More generally, it seems that MAS is better at coping with a simultaneous increase of urgency, dynamism, and scale compared to COP. This indicates that



(a) Scale = 1, dyn = 20 (b) Scale = 1, dyn = 50 (c) Scale = 1, dyn = 80

(d) Scale = 5, dyn = 20 (e) Scale = 5, dyn = 50 (f) Scale = 5, dyn = 80

(g) Scale = 10, dyn = 20 (h) Scale = 10, dyn = 50 (i) Scale = 10, dyn = 80

Figure 4.15: Comparison with competitive ratio to MAS versus urgency for all levels of scale and dynamism. The error bars indicate a confidence interval of 99%.

MAS is better at coping with a continuously changing and large scale problem that requires quick decisions.

Scale

The third hypothesis: *A CNET based MAS finds solutions with a lower operating cost compared to centralized algorithms on larger scale problem instances*, raises several related hypotheses. First, when scaling up a problem it is expected that, because of the larger solution space, algorithms have more difficulty of finding good solutions. Therefore, we expect that the average cost per order increases for larger scale problems. Figure 4.16 shows that this is not true for

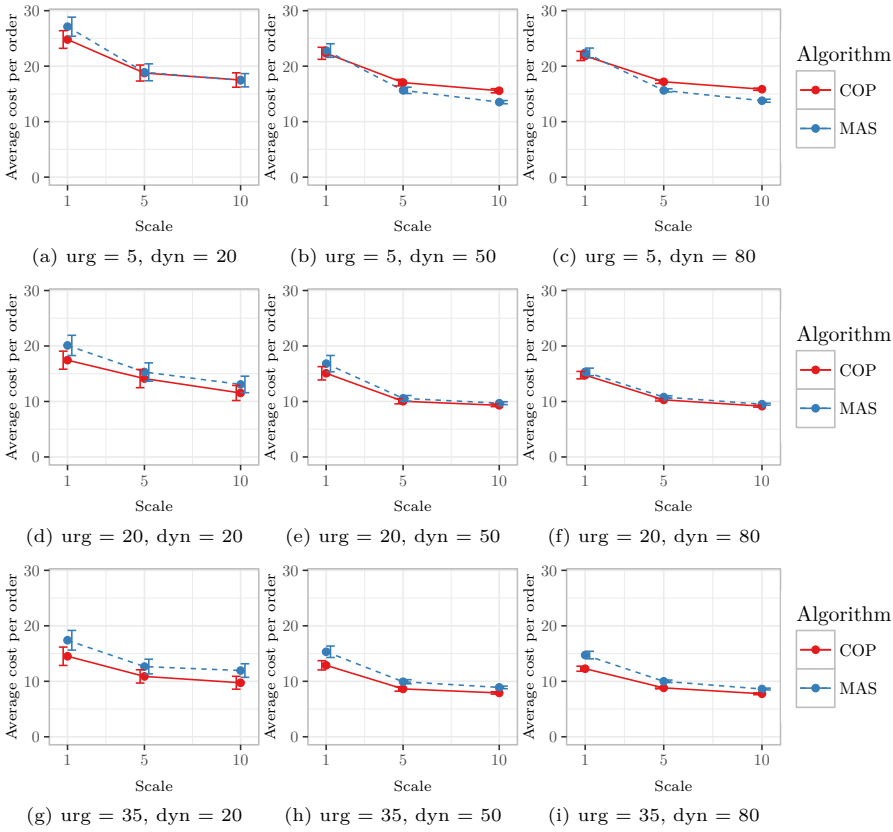
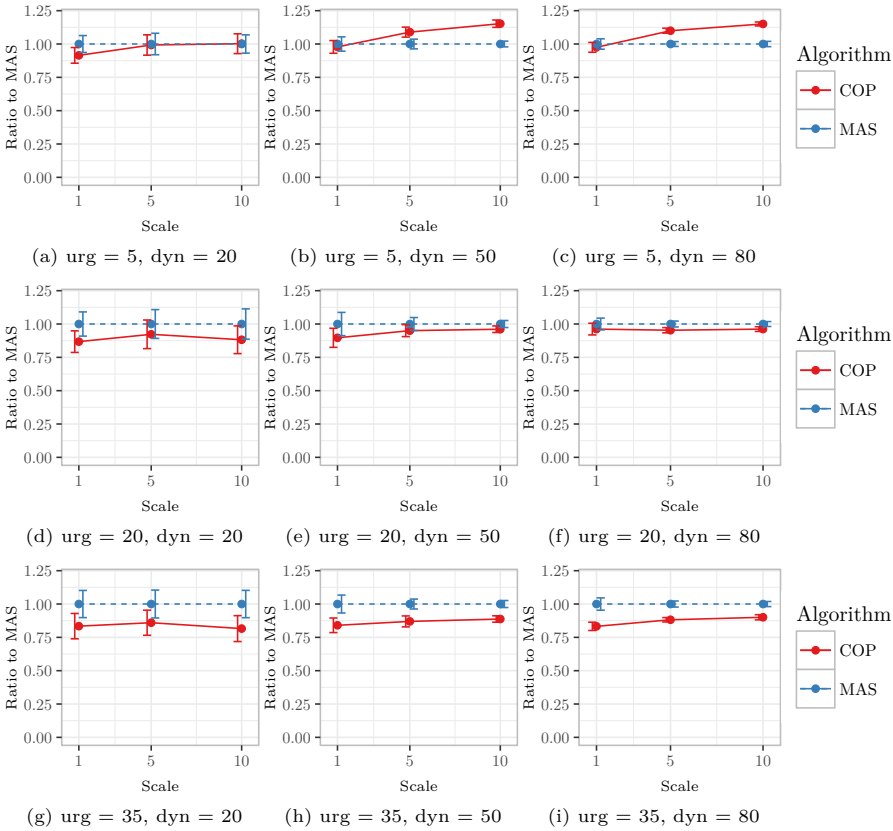


Figure 4.16: Comparison with mean relative cost versus scale for all levels of urgency and dynamism. The error bars indicate a confidence interval of 99%.

MAS. The average cost for scale 5 is always significantly lower than the average cost for scale 1, this is similarly the case between scale 5 and scale 10, except for Figures 4.16(a) and 4.16(g) where the difference is not significant ($p \approx .027$ and $p \approx .348$ respectively). For COP the situation is similar, there are only two cases where the cost is not decreasing significantly, between scale 5 and 10 in Figure 4.16(a) ($p \approx .050$) and in Figure 4.16(g) ($p \approx .102$). This leads us to reject the hypothesis for both COP and MAS. Although seemingly counter intuitive, these results are logical when considering the fact that with larger scale problems both the number of vehicles and the number of orders increase. Since there are more vehicles, the average distance of a new arriving order to the closest vehicle will be lower. This has a positive effect on the tardiness and distance traveled. The results indicate that both algorithms have enough time to explore the search space to exploit the larger number of available vehicles.

Figure 4.17 shows that the relative performance of COP and MAS depends on the level of dynamism and urgency. For example, in very urgent and medium to very dynamic situations (Figures 4.17(b) and 4.17(c)), MAS benefits more from an increasing scale compared to COP, while in not so dynamic situations, this trend seems to be reverse (Figures 4.17(d) and 4.17(g)). In Figure 4.17(f) the trend appears approximately parallel. Based on these differing trends we cannot accept the hypothesis that MAS is generally more scalable than COP.



(a) urg = 5, dyn = 20 (b) urg = 5, dyn = 50 (c) urg = 5, dyn = 80

(d) urg = 20, dyn = 20 (e) urg = 20, dyn = 50 (f) urg = 20, dyn = 80

(g) urg = 35, dyn = 20 (h) urg = 35, dyn = 50 (i) urg = 35, dyn = 80

Figure 4.17: Comparison with competitive ratio to MAS versus scale for all levels of urgency and dynamism. The error bars indicate a confidence interval of 99%.

4.5.3 Discussion

All three hypotheses, about MAS's being better to cope with increasing dynamism, urgency, and scale, compared to centralized algorithms have been rejected. However, the reverse hypotheses can neither be accepted. The results are more nuanced, the centralized algorithm is better in most situations but there exist specific problems that are very dynamic, very urgent, and large scale, for which MAS's are better. Table 4.9 shows that of the 27 different settings in the experiment, there are four settings where MAS significantly outperforms COP, 18 settings where COP significantly outperforms MAS, and five settings where the difference is not significant. The four settings where

Table 4.9: Average results of both COP and MAS for each setting. The 'Best' column indicates the best performing algorithm for that scenario class, a † indicates that the difference is not statistically significant ($p < 0.01$).

Dynamism	Urgency	Scale	COP	MAS	Ratio	p-value	Best
20	5	1	24.804	27.115	0.915	1.683e-07	COP
50	5	1	22.318	22.813	0.978	7.068e-02	COP†
80	5	1	21.836	22.392	0.975	7.209e-02	COP†
20	20	1	17.444	20.101	0.868	7.191e-11	COP
50	20	1	15.080	16.821	0.896	3.908e-07	COP
80	20	1	14.747	15.330	0.962	1.913e-02	COP†
20	35	1	14.514	17.399	0.834	9.080e-12	COP
50	35	1	12.878	15.322	0.840	1.390e-11	COP
80	35	1	12.267	14.733	0.833	2.128e-10	COP
20	5	5	18.758	18.896	0.993	1.227e-01	COP†
50	5	5	17.038	15.644	1.089	2.690e-14	MAS
80	5	5	17.188	15.629	1.100	2.293e-16	MAS
20	20	5	14.128	15.310	0.923	3.487e-13	COP
50	20	5	10.039	10.562	0.950	4.042e-07	COP
80	20	5	10.290	10.794	0.953	2.509e-07	COP
20	35	5	10.878	12.652	0.860	1.770e-15	COP
50	35	5	8.628	9.920	0.870	7.927e-16	COP
80	35	5	8.817	9.997	0.882	1.809e-12	COP
20	5	10	17.506	17.463	1.002	6.332e-01	MAS†
50	5	10	15.574	13.513	1.153	8.956e-25	MAS
80	5	10	15.843	13.771	1.150	1.886e-21	MAS
20	20	10	11.531	13.074	0.882	1.099e-14	COP
50	20	10	9.308	9.685	0.961	1.686e-06	COP
80	20	10	9.150	9.511	0.962	3.054e-06	COP
20	35	10	9.736	11.936	0.816	5.141e-20	COP
50	35	10	7.901	8.901	0.888	2.529e-18	COP
80	35	10	7.752	8.609	0.900	1.268e-14	COP

MAS outperforms COP all have an urgency of 5 minutes. This indicates that the advantage of COP of having a global view on the problem diminishes for very urgent problems. For very urgent problems, COP may not have enough time for searching the solution space, apparently, the implicit solution space partitioning of the CNET algorithm helps finding a good solution in a short amount of time. This is interesting because centralized algorithms can also benefit from this knowledge, for example, it would be interesting to experiment with a similar partitioning but in a centralized setting.

The average COP-to-MAS ratio is 0.942, meaning that when COP operates the fleet of vehicles it costs, on average, only 94.2% relative to MAS. This means that in general, COP is the preferred solution approach. However, if it is known beforehand that the problem is very urgent (ratio 1.039), very urgent and medium to large scale (ratio 1.081), or, medium to very dynamic, very urgent, and medium to large scale (ratio 1.123).

In practice, the deployment of the algorithms under investigation is relevant. A benefit of MAS's is their ability to be deployed decentrally as well as centrally, this allows MAS's to replace algorithms in an existing centralized deployment. Additionally, due to their decentralized nature, parallelizing MAS's is trivial, in fact the current implementation is already multi-threaded (although in the experiments limited to a single thread) but executed on a single core. Although out of scope of the current study, running the MAS's using multiple cores would theoretically decrease the average costs per order. Additionally, MAS's can be deployed in a distributed setting using smartphones to run the agents on. This allows a purely decentralized setup that is robust to hardware failure. If some hardware fails (e.g. a smartphone) it would only affect one vehicle and the orders that it has won in an auction. These effects could even be reduced by implementing a protocol between a vehicle agent and order agent that frequently checks whether it is still alive, similarly to pheromone evaporation in Delegate MAS (Holvoet et al., 2009).

4.6 Conclusion

A widely held belief in multi-agent systems literature is that MAS is advantageous in operational research problems that are very dynamic and/or large scale. However, such claims were never supported by evidence based on a systematic empirical study. Present chapter is the first to systematically investigate the influence of dynamism, urgency, and scale on the performance of both multi-agent systems and centralized algorithms. Based on the experimental results, we reject the hypotheses that the MAS has a lower average operating

cost compared to the centralized algorithm on more dynamic, more urgent, or larger scale problems. However, the reverse hypotheses can neither be accepted. The results are more nuanced, the solutions found by the centralized algorithm cost, on average, only 94.2% of the cost of the solutions found by the multi-agent system. This indicates that the centralized algorithm generally performs better compared to the multi-agent system. However, for scenarios that are medium to very dynamic, very urgent, and medium to large scale, the average relative cost of the centralized algorithm is 112.3%, indicating that under these circumstances, the multi-agent system performs better than the centralized algorithm. When assessing the performance of the algorithms individually per scenario property, there is not one algorithm that generally outperforms the other on that dimension.

Running an empirical study for comparing distinct algorithms is a tedious task. We have formally defined the pickup-and-delivery problem, including the scenario properties: dynamism, urgency, and scale. For the algorithms we used OptaPlanner, a well known satisfaction solver library. A tuning experiment was conducted to find the best performing OptaPlanner algorithm for this problem. The best algorithm was incorporated in an online centralized algorithm and a multi-agent system based on the dynamic contract-net protocol. In order to perform a fair empirical study it is imperative to use a real-time simulator that assigns the same processing power to the approaches. For this reason we have extended the RinSim logistics simulator and have demonstrated that fluctuations caused by the real-time nature of the simulator have a minimal impact on the end result.

To facilitate complete reproducibility, the simulator, datasets, algorithms, and results are available online. This allows some interesting directions for future work. For example, the algorithms could be evaluated on different problem instances generated with our dataset generator or on different problems in the field of logistics. Additionally, there are many other MAS coordination protocols such as Delegate MAS or Gradient Field, that can be evaluated and compared to the algorithms used in the present chapter. Similarly, there are many more centralized algorithms and libraries that implement them. Present chapter provides a benchmark, an ideal starting point for further research into more advanced algorithms.

During the realization of this chapter the authors published an investigation into optimizing multi-agent systems with genetic programming (Chapter 5) and evaluated it using the benchmark presented in this chapter.

Acknowledgments

This research is partially funded by the Research Fund KU Leuven.

Chapter 5

Optimizing agents with genetic programming - An evaluation of hyper-heuristics in dynamic real-time logistics

Building upon the results of the evaluation described in the previous chapter, we explore hyper-heuristics as a technique for optimizing agents in dynamic logistics. This chapter contains the paper:

> van Lon, R. R. S., Branke, J., & Holvoet, T. (2017). Optimizing agents with genetic programming: an evaluation of hyper-heuristics in dynamic real-time logistics. *Genetic Programming and Evolvable Machines*, (pp. 1–28). doi:10.1007/s10710-017-9300-5

The content presented in this chapter is the result of discussions led by Rinde R.S. van Lon with the other two authors. Rinde R.S. van Lon programmed the software and wrote the paper, Juergen Branke and Tom Holvoet provided feedback on the writing.

Abstract

Dynamic pickup and delivery problems (PDPs) require online algorithms for managing a fleet of vehicles. Generally, vehicles can be managed either centrally or decentrally. A common way to coordinate agents decentrally is to use the contract-net protocol (CNET) that uses auctions to allocate tasks among agents. To participate in an auction, agents require a method that estimates the value of a task. Typically, this method involves an optimization algorithm, e.g. to calculate the cost to insert a customer. Recently, hyper-heuristics have been proposed for automated design of heuristics. Two properties of automatically designed heuristics are particularly promising: 1) a generated heuristic computes quickly, it is expected therefore that hyper-heuristics perform especially well for urgent problems, and 2) by using simulation-based evaluation, hyper-heuristics can create a 'rule of thumb' that anticipates situations in the future. In the present chapter we empirically evaluate whether hyper-heuristics, more specifically genetic programming (GP), can be used to improve agents decentrally coordinated via CNET. We compare several GP settings and compare the resulting heuristic with existing centralized and decentralized algorithms based on the OptaPlanner optimization library. The tests are conducted in real-time on a dynamic PDP dataset with varying levels of dynamism, urgency, and scale. The results indicate that the evolved heuristic always outperforms the optimization algorithm in the decentralized multi-agent system (MAS) and often outperforms the centralized optimization algorithm. This chapter demonstrates that designing MASs using genetic programming is an effective way to obtain competitive performance compared to traditional operational research approaches. These results strengthen the relevance of decentralized agent based approaches in dynamic logistics.

5.1 Introduction

The PDP is a logistics problem where a fleet of vehicles transports customers or goods from origin to destination (Parragh et al., 2008). The dynamic PDPTW is an online variant where some or all customers' orders arrive during the operating hours and where customers impose time window constraints on pickups and deliveries (Berbeglia et al., 2010). Typically, the objective in PDPTW is to serve all customers while minimizing fuel costs and time window violations. In a purely dynamic PDPTW, no order is known before the operating hours. When a new order is announced, the available computation time for an algorithm is limited by the order's urgency, the amount of available time until the order needs to be serviced (Chapter 2). Together, the dynamism, urgency, and scale

of a problem, directly affect the amount of computations that need to be done as well as how much time is available for performing them (Chapter 3).

Decentralized MAS's are commonly considered to be a good fit for large scale and dynamic problems because of their ability to make quick local decisions (Fischer et al., 1995; Glaschenko et al., 2009; Weyns et al., 2006). Together, the local decisions made by all agents aim to solve the global optimization problem. There are two different approaches for making these decisions: 1) explicitly searching through the space of possible schedules using an (exact or heuristic) optimization procedure, or, 2) using a heuristic, a rule of thumb, that guides the agent by assigning priorities to actions without explicitly searching the space of schedules. The aim of the present chapter is to compare the performance of these two different approaches. For the first approach we use a tabu search algorithm from the OptaPlanner (De Smet et al., 2016) optimization library. For the second approach we use genetic programming to automatically design an agent-based heuristic.

5.1.1 Related work

A recent empirical study (Chapter 4) employs a MAS with an auction based CNET. The agents place bids to the customer indicating the estimated additional cost to perform the transportation task. Each agent computes this bid value by running an optimization procedure for a limited time. The experiments indicate that the MAS only outperforms a reference centralized algorithm in case the problem is medium to very dynamic, very urgent, and medium to large scale. A problem instance with these properties is changing continuously (medium to very dynamic), vehicles have a short amount of time to respond to incoming requests (very urgent), and there are relatively many vehicles and orders (medium to large scale). In this situation the computational demands are very high, limiting the viability of searching the solution space centrally. The CNET approach, however, uses implicit partitioning of the search space, apparently this helps in these circumstances to find a good solution in a short period. Since Chapter 4 considers purely dynamic PDPTWs, we know that the problem is likely to change soon after a bid value is computed. A reasonable assumption is therefore that a good bid value should incorporate expected future events that affect the transportation cost of an order. However, in the setup of Chapter 4, the optimization algorithm, OptaPlanner (De Smet et al., 2016), only considers all information that is known up to the moment of computation. An alternative for the optimization procedure is a heuristic that may include estimates of future events. Designing such a heuristic is, however, a difficult task. A local decision made by an agent can have far reaching global consequences. That is because a

collection of agents acting according to decentralized local rules constitutes a complex system with emergent and difficult to predict behavior.

Research on dynamic optimization problems, such as dynamic PDPTWs, is concerned with optimization in an environment that changes over time (Cruz et al., 2011). Dynamic optimization problems are often approached using metaheuristics (Nguyen et al., 2012; Yang et al., 2012). Metaheuristics, such as swarm intelligence and evolutionary computation, are a good fit for these dynamic problems because they are inspired on natural processes, which themselves are subject to a continuously changing environment. In present chapter, instead of using evolutionary algorithms to solve our problem directly, we use genetic programming to generate a heuristic that solves our problem.

Hyper-heuristics is a branch of optimization literature concerned with the automatic design of heuristics (Burke et al., 2013). Burke et al. (2010) distinguishes two different categories of hyper-heuristics, heuristic selection and heuristic generation. Heuristic selection comprises methodologies for choosing or selecting existing heuristics while heuristic generation is concerned with generating new heuristics from components of existing heuristics. GP is a subfield of evolutionary computing (Eiben & Smith, 2007), that works with variable size LISP-tree representations and thus is able to evolve functions of arbitrary complexity, making it particularly suitable for the design of heuristics. Hyper-heuristics and GP in particular, have been applied in a wide range of contexts, including production scheduling (Branke et al., 2016), traveling salesman problems (Keller & Poli, 2008), bin packing (Burke et al., 2006), etc.

The combination of hyper-heuristics and MAS for dynamic PDPTW has been explored before. To the best of our knowledge, Beham et al. (2009) were the first to apply hyper-heuristics to an agent-based algorithm for the PDPTW. In their MAS, vehicle agents are governed by two separate heuristics, one heuristic determines its next location to travel to and another heuristic determines the order(s) to pick up at a pickup site. Both heuristics are weighted sums of hand-crafted heuristics, the weights are set by an evolution strategy (ES). Determining the quality of the heuristics during evolution is done with a simulation-based fitness function. Beham et al. (2009) did not compare their approach with alternative algorithms.

Similarly, van Lon et al. (2012) used GP to evolve the guiding heuristic for a MAS in a dynamic PDPTW context. Vehicles have a capacity of one order, implying that a vehicle must immediately go to an order's destination after pick up. The evolved heuristic assigns priorities to all available orders. Each vehicle that is not currently carrying an order executes its heuristic frequently, and travels to the order with the highest priority. The agents do not communicate amongst each other, leading to inefficiencies in case several vehicles have the

same priority. Because the problem is dynamic, priorities of vehicles change, causing vehicles to divert from their route. In their paper, van Lon et al. show that their MAS approach with an evolved heuristic outperforms a centralized meta-heuristic.

The work by Vonolfen et al. (2013) extends (van Lon et al., 2012). Instead of using just three terminals in GP as was done in (van Lon et al., 2012), Vonolfen et al. use 18 different terminals. This includes several terminals that incorporate information about other agents' distances and destinations. The authors compare their approach with two algorithms, a (centralized) tabu search algorithm and the evolution strategy presented in (Beham et al., 2009). Vonolfen et al. report that the tabu search algorithm outperforms both the GP as well as the ES approach, while GP outperforms ES.

Continuing in this line of research, Merlevede et al. (2014) use neuroevolution of augmenting topologies (NEAT) to evolve a neural network as a priority heuristic. The authors use the same MAS approach as in (van Lon et al., 2012) but they evaluate their performance on an existing dynamic PDPTW benchmark. They are the first to report negative results, the reference centralized algorithm always outperforms the NEAT approach. These results are likely caused by the lack of a coordinating mechanism for their MAS.

5.1.2 Contributions and overview

The papers described above that apply hyper-heuristics to MAS for dynamic PDPTW have several drawbacks which we aim to overcome in present chapter. First, the discussed hyper-heuristics have not been evaluated in real-time. In a dynamic logistics problem, algorithm computation time directly affects the performance of the fleet of vehicles. Therefore, when comparing hyper-heuristics to traditional optimization algorithms in dynamic PDPTW, a real-time simulator is required. Second, for a fair comparison of two different algorithms, it is important that both algorithms are subject to exactly the same constraints. When comparing hyper-heuristics in a MAS setting, a fair comparison is to have a reference algorithm that is also used in a MAS setting. Unfortunately, none of the above described works evaluate their agent-based hyper-heuristic in this way. Third, to understand the exact circumstances in which one algorithm outperforms another, it is imperative to vary the problem properties on which they are evaluated. Fourth, to allow reproducibility and extensibility, the algorithms, datasets, and software that are used should be freely available.

The aim of present chapter is to determine whether using hyper-heuristics can improve the performance of an existing MAS for a real-time logistics problem. More specifically, we are investigating two hypotheses comparing

a hyper-heuristic setup with the centralized OptaPlanner algorithm and the decentralized MAS both from Chapter 4:

- GP designed heuristic in a MAS can outperform OptaPlanner in a MAS.
- GP designed heuristic in a MAS can outperform centralized OptaPlanner.

If these hypotheses are true, it would demonstrate the relevance of decentralized MAS's in dynamic logistics and constitute an important first step towards their automatic design. Since a heuristic typically requires only a fraction of the computation time that a solver requires, we also investigate the following hypothesis:

- GP designed heuristic works especially well for more urgent problems because of its minimal computational cost.

Using the dataset and dataset generator from Chapter 3 we can train and test the heuristics on instances with different values of dynamism, urgency, and scale. We define a specialized heuristic as a heuristic that is trained on one specific scenario setting with specific properties, as opposed to a generalized heuristic that is trained on a wide range of scenario settings. We expect that:

- Specialized heuristics outperform general heuristics on scenarios for which they are specialized.
- Generalized heuristics outperform specialized heuristics on scenarios for which they are not specialized.

The chapter is organized as follows. A formal problem definition, including dynamism, urgency, and scale, and the real-time simulation platform are presented (Section 5.2). The MAS that we start from is presented in Section 5.3. The chapter presents the following contributions:

- a new application of hyper-heuristics to decentralized MAS using GP is presented (Section 5.4);
- the performance of GP and the resulting heuristics are thoroughly evaluated using real-time simulation and compared to existing results obtained by a centralized and a decentralized OptaPlanner algorithm under varying circumstances (Section 5.5);
- following Chapter 4, all code, data, and results needed to reproduce this work are made available online.

Finally, we summarize the chapter and discuss directions for future research (Section 5.6).

5.2 Dynamic pickup-and-delivery problems

This section is adapted from Chapter 3 and Chapter 4. In PDPs there is a fleet of vehicles responsible for the pickup-and-delivery of items. Dynamic PDP is an online problem. Customer transportation requests are revealed over time, during the fleet's operating hours. It is further assumed that the fleet of vehicles has no prior knowledge about the total number of requests nor about their locations or time windows. In this section, we provide an overview of the work about dynamic PDP from Chapter 3 and Chapter 4 as it serves as a foundation of the evaluation in present chapter.

5.2.1 Formal definition

In Chapter 3 a *scenario*, which describes the unfolding of a dynamic PDP, is defined as a tuple:

$$\langle \mathcal{T}, \mathcal{E}, \mathcal{V} \rangle := \text{scenario},$$

where

$$\begin{aligned} [0, \mathcal{T}) &:= \text{time frame of the scenario}, & \mathcal{T} > 0 \\ \mathcal{E} &:= \text{list of events}, & |\mathcal{E}| \geq 2 \\ \mathcal{V} &:= \text{set of vehicles}, & |\mathcal{V}| \geq 1 \end{aligned}$$

$[0, \mathcal{T})$ is the period in which the fleet of vehicles $\mathcal{V}$ has to respond to customer requests. The events, $\mathcal{E}$, represent customer transportation requests. Since we consider the purely dynamic PDPTW, all events are revealed between time 0

and time $\mathcal{T}$. Each event $e_i \in \mathcal{E}$ is defined by the following variables:

$$a_i := \text{announce time}$$

$$p_i := [p_i^L, p_i^R) = \text{pickup time window, } p_i^L < p_i^R$$

$$d_i := [d_i^L, d_i^R) = \text{delivery time window, } d_i^L < d_i^R$$

$$pst_i := \text{pickup service time span}$$

$$dst_i := \text{delivery service time span}$$

$$ploc_i := \text{pickup location}$$

$$dloc_i := \text{delivery location}$$

Reaction time is defined as:

$$r_i := p_i^R - a_i = \text{reaction time} \tag{5.1}$$

The time window related variables of a transportation request are visualized in Figure 5.1.

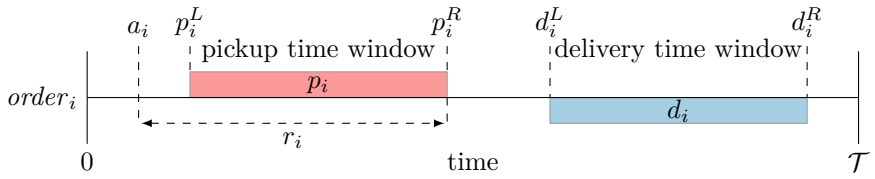


Figure 5.1: Visualization of the time related variables of a single order event $e_i \in \mathcal{E}$.

Furthermore it is assumed that:

- vehicles start at a depot and have to return after all orders are handled;
- the fleet of vehicles $\mathcal{V}$ is homogeneous;
- the cargo capacity of vehicles is infinite (e.g. courier service);
- the vehicle is either stationary or driving at a constant speed;
- vehicle diversion is allowed, this means that a vehicle is allowed to divert from its destination at any time;

- vehicle fuel is infinite and driver fatigue is not an issue;
- the scenario is completed when all pickup and deliveries have been made and all vehicles have returned to the depot; and,
- each location can be reached from any other location.

Vehicle schedules are subject to both hard and soft constraints. The opening of time windows is a hard constraint, hence vehicles need to adhere to these:

$$sp_i \geq p_i^L \tag{5.2}$$

$$sd_i \geq d_i^L \tag{5.3}$$

sp_i is the start of the pickup operation of order event e_i by a vehicle; similarly, sd_i is the start of the delivery operation of order event e_i by a vehicle. The time window closing (p_i^R and d_i^R) is a soft constraint incorporated into the objective function, it needs to be minimized:

$$min := \sum_{j \in \mathcal{V}} \left(vtt_j + td\left\{bd_j, \mathcal{T}\right\}\right) + \sum_{i \in \mathcal{E}} \left(td\left\{sp_i, p_i^R\right\} + td\left\{sd_i, d_i^R\right\}\right) \tag{5.4}$$

where

$$td\left\{\alpha, \beta\right\} := max\left\{0, \alpha - \beta\right\} \ = \ \text{tardiness} \tag{5.5}$$

vtt_j is the total travel time of vehicle v_j; bd_j is the time at which vehicle v_j is back at the depot. In summary, the objective function computes the total vehicle travel time, the tardiness of vehicles returning to the depot and the total pickup and delivery tardiness.

5.2.2 Dataset

Earlier work has argued for, and presented, a dataset characterized by three different properties of dynamic PDPs: dynamism, urgency, and scale (Chapter 3).

Dynamism

Dynamism is defined in Chapter 2. Informally, a scenario that changes continuously is said to be dynamic while a scenario that changes occasionally is said to be less dynamic. In the context of PDPTWs a change is an event

that introduces additional information to the problem, such as the events in $\mathcal{E}$. Formally, the degree of dynamism, or the continuity of change, is defined as:

$$dynamism := 1 - \frac{\sum_{i=0}^{|\Delta|} \sigma_i}{\sum_{i=0}^{|\Delta|} \bar{\sigma}_i} \tag{5.6}$$

Δ is the list of event interarrival times:

$$\Delta := \{\delta_0, \delta_1, \ldots, \delta_{|\mathcal{E}|-2}\} = \{a_j - a_i | j = i + 1 \wedge \forall a_i, a_j \in \mathcal{E}\} \tag{5.7}$$

For a scenario with 100% dynamism, the perfect interarrival time is defined as:

$$\theta := \text{perfect interarrival time} = \frac{\mathcal{T}}{|\mathcal{E}|} \tag{5.8}$$

Based on this definition, the deviation and maximum possible deviation to the perfect interarrival time can be computed:

$$\sigma_i := \begin{cases} \theta - \delta_i & \text{if } i = 0 \text{ and } \delta_i < \theta \\ \theta - \delta_i + \dfrac{\theta - \delta_i}{\theta} \times \sigma_{i-1} & \text{if } i > 0 \text{ and } \delta_i < \theta \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

$$\bar{\sigma}_i := \theta + \begin{cases} \dfrac{\theta - \delta_i}{\theta} \times \sigma_{i-1} & \text{if } i > 0 \text{ and } \delta_i < \theta \\ 0 & \text{otherwise} \end{cases} \tag{5.10}$$

Eq. 5.6 uses the proportion of the actual deviation and the maximum possible deviation. Using this definition the degree of dynamism of any scenario can be computed.

Urgency

In Chapter 2 urgency is defined as the maximum reaction time available to the fleet of vehicles in order to respond to an incoming order. Or more formally:

$$urgency\,(e_i) := p_i^R - a_i = r_i \tag{5.11}$$

To obtain the urgency of an entire scenario the mean and standard deviation of the urgency of all orders can be computed.

Scale

Scale is defined in Chapter 3 as maintaining a fixed objective value per order while scaling the number of orders up in proportion to the number of vehicles in the fleet. Scaling up a scenario $\langle \mathcal{T}, \mathcal{E}, \mathcal{V} \rangle$ with a factor α will create a new scenario $\langle \mathcal{T}, \mathcal{E}', \mathcal{V}' \rangle$ where $|\mathcal{V}'| = |\mathcal{V}| \cdot \alpha$ and $|\mathcal{E}'| = |\mathcal{E}| \cdot \alpha$.

5.2.3 Realistic simulation platform

The experiments performed in Chapter 4 use the RinSim real-time logistics simulator (van Lon & Holvoet, 2012). For fair comparison we use the same simulator. RinSim is a discrete-time logistics simulator that supports running both centralized algorithms and decentralized multi-agent systems. RinSim is written in Java and has a modular design (Figure 5.2), a `Model` encapsulates a part of a problem domain or algorithm. The simulator can be customized

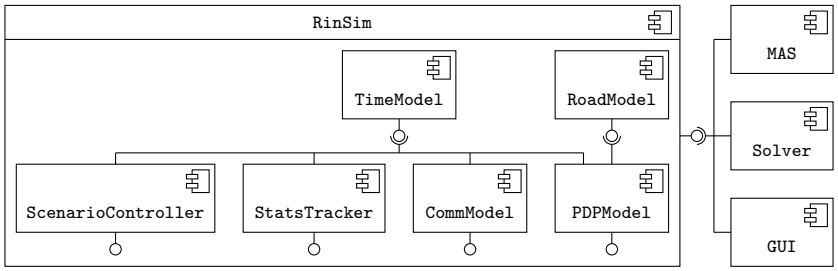


Figure 5.2: UML component diagram of RinSim. The simulator subsystem can be configured with a variety of models that all provide some interface. MASs, solvers, and the graphical user interface use these interfaces to interact with RinSim.

by selecting the models that are used, this allows simulating a wide variety of logistics problems while maximally reusing existing code.

RinSim supports simulations using simulated time as well as real-time. The standard JVM has no built-in support for real-time execution. However, RinSim is designed such that it provides soft real-time behavior using the standard JVM. Soft real-time, as opposed to hard real-time, allows occasional deviations from the desired execution timing.

RinSim discretizes time into intervals called 'ticks'. The simulator is initialized with a fixed tick length, for example a tick length of 250 milliseconds. When simulating without real-time constraints, the simulator computes all ticks as fast as possible. In a real-time simulator the interval between the *start* of two ticks should be the tick length (e.g. 250 ms). Since the JVM doesn't

allow precise control over the timings of threads it is generally impossible to guarantee hard real-time constraints. In real-time mode, RinSim uses a dedicated thread for executing the ticks. If computations need to be done that are expected to last longer than a tick, they must be done in a different thread. This minimizes interference of computations with the advancing of time in the simulated world. Additionally, the processor affinity of the threads are set at the operating system level. Setting the processor affinity to a Java thread instructs the operating system to use one processor exclusively for executing that thread. In practice, the actual scheduling of threads on processors depends on the number of available processors and the operating system.

Running a complete logistics simulation in real-time is time consuming, as it will simulate every tick synchronized with real time. However, depending on the specific simulation that is being run, there may be long intervals where no computations are being done other than that of the simulator advancing time in the simulated world. For this reason, RinSim employs a mechanism to dynamically switch between real-time and simulated time. When the simulator is in simulated time, ticks will be executed as fast as possible speeding up the simulation significantly. As soon as a computation needs to be done, the simulator must first switch back to real-time mode before this computation can be started.

5.3 Multi-agent systems for dynamic PDP

This section is adapted from Chapter 4. The multi-agent system that is extended is an implementation of the DynCNET presented by Weyns et al. (2007). DynCNET is a dynamic extension of the CNET first proposed by Smith (1980). Inspired by how companies use subcontracting to collaboratively solve problems, CNET uses contracting to approach the task assignment problem. In CNET, the agent that tenders a task is called the *manager* and sends a task announcement to potential *contractors*. Each potential contractor can either ignore the announcement or send a *bid* to the manager. The manager then selects the best bid and *awards* the task to the contractor. Figure 5.3 shows the UML interaction diagram for the CNET auction process. Although an auction can be, and usually is, used in a competitive setting, we use auctions in a purely cooperative setting. We assume that both the contractors and the manager are working for the same company. The dynamic extension of CNET provides flexibility to the assignment until a contractor has to commit to the execution of the task. The same task can be announced several times before its execution, its assignment changing after every announcement.

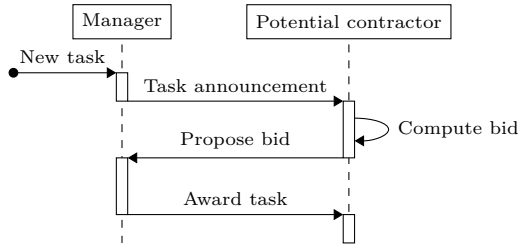


Figure 5.3: UML interaction diagram of a CNET auction.

In our MAS implementation for the dynamic PDPTW, both the vehicle as well as the transportation requests are modeled as agents. In the remainder of this text we will call the agent controlling a vehicle a `VehicleAgent` and the agent responsible for a transportation request an `OrderAgent`. `OrderAgent`s are playing the role of the manager in DynCNET, `VehicleAgent`s are the potential contractors. Figure 5.4 shows an interaction diagram of an auction using our DynCNET implementation. At the end of an auction, each `VehicleAgent`

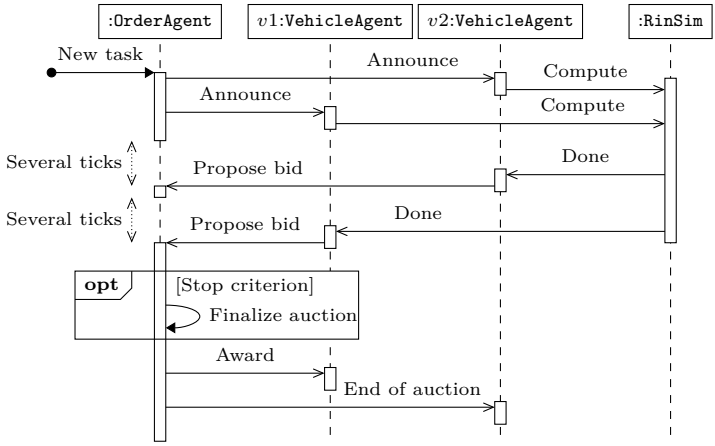


Figure 5.4: UML interaction diagram of an auction of an order with two vehicles. Upon receiving the auction announcement, both `VehicleAgent`s start computing a bid. The computations take several ticks. As soon as the `OrderAgent` has met the stop criterion, in this case receiving two bids is enough, the auction is finalized and the order is awarded to $v1$. Vehicle $v2$ is notified of the end of the auction. The `RinSim` lifeline is a simplified view of the multi-threaded computation facilities provided by RinSim. Note that the filled arrows indicate synchronous calls and the stick arrows indicate asynchronous calls.

is either awarded the order or notified of the end of the auction. At this moment the `VehicleAgent`s have the possibility of starting a new auction by offering one of their previously awarded orders. The `VehicleAgent` will inform

the `OrderAgent` responsible for the order that is to be offered to start a new auction, the `OrderAgent` will then perform a new auction process similar to Figure 5.4. A possible outcome of this auction is that the order is not awarded to another vehicle but stays assigned to the original vehicle. Allowing the vehicles to start a new auction process enables the dynamic (re)allocation of orders and makes the CNET implementation dynamic.

5.3.1 Order agent

The `OrderAgent` (the manager in CNET terminology) is responsible for the auction process. It announces the start of the auction to all vehicles and waits until it receives enough bids to make a decision. The stop criterion for the bidding process is:

$$|bids| \geq 2 \wedge (|bids| = |vehicles| \vee auction\_duration \geq 5000)$$

where, $|bids|$ is the number of received bids, $|vehicles|$ is the total number of vehicles which equals the potential maximum number of bids and $auction\_duration$ is the duration of the auction in milliseconds.

When the stop criterion evaluates to *true*, the `OrderAgent` finalizes the auction by selecting the best bid as the winner. The best bid is defined as the bid with the lowest price (cost). The order is assigned to the winner, the winner must therefore service that order, unless it decides to auction it and somebody else wins that auction at a later time. All `VehicleAgent`s are informed of the end of the auction. This allows agents that are still computing their bids for this auction to cancel their computations. Bids that are received after the finalization of the auction are ignored.

5.3.2 Vehicle agent

A `VehicleAgent` needs to compute a bid value in order to propose a bid. In Chapter 4 the bid value is computed using a solver. The cost of an order is defined as the additional cost that including that order incurs to a vehicle's current schedule:

$$cost(order) = cost(new\_schedule) - cost(current\_schedule) \tag{5.12}$$

where, $current\_schedule$ is the schedule of the vehicle including all previous order assignments, and $new\_schedule$ is the current schedule of the vehicle including the proposed order. The task of the solver is finding the best $new\_schedule$ in a relative short amount of time to get a reliable estimate of

the cost of the auctioned order. The time for computing the new schedule is limited because the auction process has a limited duration, the bid needs to be proposed before the end of this duration in order to ensure that the `OrderAgent` will take the bid into account.

As soon as the assignment of orders to a vehicle has changed, the `VehicleAgent` needs to update its schedule. The vehicle's schedule is optimized by a solver (the schedule solver), although it is imperative to generate a *complete* schedule quickly, the solver can compute for a longer time as the solver can continuously notify the `VehicleAgent` of improved schedules. This allows the optimization process to continue for an extended period.

The `VehicleAgent` considers starting a new auction (a reauction) in the following two situations:

- when a vehicle has not won an auction for at least five minutes; or,
- when the vehicle's current schedule has changed.

When starting a new auction the vehicle has to decide which of its previously assigned orders it should auction. The order that when removed yields the greatest schedule cost reduction, for that vehicle, is selected. Computing the cost reduction of removing an order from the current route does not require an optimization step (the route is not optimized again) and can therefore be computed quickly for all orders assigned to a vehicle (similar to eq. 5.12). Orders for which the pickup operation is in process or is already done are not considered for auctioning as they can't be reassigned. If the order with the greatest cost reduction is the last received order, no auction is performed to avoid excessive auctioning. The `VehicleAgent` itself has to propose a bid to its own auction, only when another agent proposes a better bid will the order be reassigned.

In Chapter 4 computations by the agents are done using an optimization algorithm from the OptaPlanner library (De Smet et al., 2016). OptaPlanner is an open source Java constraint satisfaction engine that optimizes planning problems. The project is developed by De Smet et al. and sponsored by RedHat. OptaPlanner provides a wide range of optimization algorithms such as construction heuristics and metaheuristics. It has support for various problem domains such as scheduling and vehicle routing. In the experiments described in this chapter we use version 6.4.0. In Chapter 4 it was established that a first-fit decreasing construction heuristic followed by step counting hill climbing with tabu search and strategic oscillation performs best on dynamic PDPTWs. Therefore we use the same algorithm in this chapter. In the remainder of this chapter, when we refer to OptaPlanner we refer to this specific algorithm unless mentioned otherwise.

5.4 Genetic programming for enhancing agents

To enhance the MAS discussed in Section 5.3 using GP we replaced OptaPlanner in the `VehicleAgent` with an evolved heuristic.

5.4.1 Heuristics in agents

As described in Section 5.3, the `VehicleAgent` has three different decisions to make:

1. Assigning a bid value to an auctioned parcel, currently being done using cheapest insertion cost with the insertion computed by OptaPlanner.
2. Deciding what parcel to reauction, currently taking the most expensive parcel.
3. Finding the cheapest route to all destinations, currently computed using OptaPlanner.

Assigning a bid value to a parcel (1) and deciding which parcel to reauction (2) can easily be done by a heuristic:

```
(vehicle,parcel) -> cost
```

The heuristic is executed by a vehicle, the output is an estimation of the cost of adding the specified parcel into the route of the vehicle and possibly further considerations.

5.4.2 Genetic programming setup

Since the quality of a heuristic cannot be deduced analytically, we are using simulation-based fitness evaluation. Since real-time simulation is very time consuming, we are using RinSim (Section 5.2.3) with simulated time during evolution. Additionally, to also save computation time, we use the cheapest insertion cost heuristic instead of OptaPlanner for computing the cheapest route to all destinations. To avoid spending too much time on simulating inferior individuals we use RinSim with a custom stop condition:

$$\text{stop}(t) := \begin{cases} \exists v_i \in \mathcal{V}\ \text{route\_length}(v_i) > \max\left(40, |\mathcal{E}_t| - |\mathcal{D}_t|\right) & \text{if } t \leq 8 \text{ hours} \\ \text{true} & \text{otherwise} \end{cases}$$

where t is the current time, $|\mathcal{E}_t|$ is the number of parcel announce events up to time t, and $|\mathcal{D}_t|$ is the number of delivered parcels up to time t. The stop condition is designed to stop the simulation if it takes too long to deliver all parcels or if there is a single vehicle that is hoarding parcels. Hoarding is defined as a vehicle that has more than about 50% of all possible visits in its route. The theoretical maximum number of visits is indicated by $2 \cdot |\mathcal{E}_t| - |\mathcal{D}_t|$. A vehicle route may contain each parcel at most twice (once for pickup, once for delivery), if the route length is larger than the number of undelivered parcels this means that about 50% of the parcels are in that route. The stop condition only applies when the total route length is more than 40. The stop condition halts simulations of bad quality individuals, saving computation time for individuals of higher quality.

The fitness function, that needs to be minimized, is:

$$\text{fitness} := \begin{cases} \text{fitness}^{\max} - t & \text{if simulation terminated early} \\ \text{cost (eq. 5.4)} & \text{otherwise} \end{cases}$$

The fitness of individuals that are stopped by the stop condition is the maximum fitness value subtracted with the time of the simulator at which it was stopped. This adds some differentiation to low quality individuals.

The GP settings that we use are listed in Table 5.1. The best number of

Table 5.1: Genetic programming settings.

Parameter	Value
Population size	500
Generations	100
Number of evaluations per individual	50
Num evals in last generation	250
Crossover proportion	90%
Mutation proportion	10%
Elitism	1
Selection method	Tournament selection (size 7)
Maximum tree depth	17

evaluations is highly problem specific (Branke et al., 2016). The choice of number of evaluations per individual needs to be high enough to avoid over specialization within a single generation while it needs to be low enough to keep the experiments computationally feasible. Preliminary experiments showed that 50 evaluations produces convergence graphs that are considerably smoother compared to lower number of evaluations, while still being computationally feasible. Similar to (van Lon et al., 2012), we choose a large number of evaluations in the last generation

since this is the most important generation as it chooses the champion heuristic. The maximum tree depth of 17 is also used by Koza (1994, p. 265) and is the default of ECJ, the evolution software framework (Luke et al., 2011) that we use.

Table 5.2 lists the functions, and Table 5.3 lists the terminals that are used.

Table 5.2: Functions used in GP.

Function name	Arity	Description
if4	4	if `arg0` < `arg1` then `arg2` else `arg3`
+, -, /, x	2	Mathematical operators
pow	2	$\texttt{arg0}^{\texttt{arg1}}$, raises `arg0` to the power of `arg1`
neg	1	Negates `arg0`
min, max	2	Takes the minimum or maximum, respectively, of the provided arguments.

Table 5.3: Terminals used in GP. The terminals have a context of a vehicle (the vehicle that executes the heuristic) and a parcel of interest.

Function name	Description
insertion cost insertion travel time insertion tardiness insertion over time insertion flexibility	Computes the difference between the current and a possible new tour of a vehicle, as computed by the cheapest insertion heuristic. Cost is the sum of travel time, tardiness, and over time (as in eq. 5.4). Flexibility is defined in eq. 5.13.
ado mido mado	Average, minimum, or maximum travel time, respectively, from the pickup and delivery location of the parcel of interest to all locations in the vehicle's route. These heuristics are inspired by the heuristics of the same name by Beham et al. (2009).
pickup urgency delivery urgency	The time left until the end of the pickup/delivery time window of the parcel of interest (in minutes).
time left	The time left in minutes until the end of the day.
slack	The amount of idle time, in minutes, that the current vehicle has.
route length	The current size of the vehicle's route.
0,1,2,10	Constants, to limit the search space we only use the four most relevant constants.

One of the terminals is based on the concept of flexibility in a route. Flexibility is the degree to which arrival times in a vehicle's route can be changed without

Listing 5.1: Simple heuristic example code.

```
(x (max (- (+ insertion overtime delivery urgency)
           insertion flexibility)
        (pow insertion tardiness 2.0))
   (pow 10.0 insertion cost))
```

introducing time window violations. This is calculated as follows:

$$\text{flexibility}(\text{route}) := \sum_{r_i \in \text{route}}^{|\text{route}|} \text{lpa}(r_i) - \text{epa}(r_i) \tag{5.13}$$

where, $\text{lpa}(r_i)$ is the last possible arrival time without time window violations and $\text{epa}(r_i)$ is the earliest possible arrival time without time window violations.

We use the standard tree-based representation of GP. A simple example of a heuristic composed of an arbitrary set of functions and terminals is shown as a Lisp expression (Listing 5.1) and as a tree (Figure 5.5).

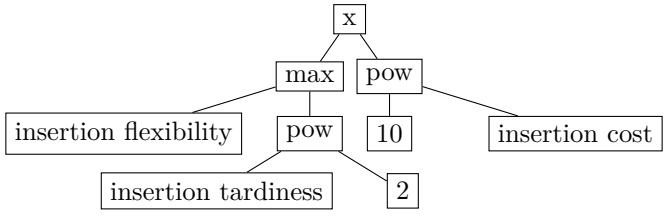


Figure 5.5: Simple heuristic example visualized as a tree.

We simulate each individual on 50 different scenarios. Each scenario describes a period of four hours in which 120 orders (in the small scale variant) are announced. Since a scenario is the product of a stochastic process, the difficulty of scenarios varies. Within a generation this is not a problem because fitness indicates an algorithm's quality on a set of scenarios. Consequently, when comparing two algorithms within a generation, the fitness values can be compared directly. However, a convergence graph that shows absolute values will show a lot of noise because the values between generations can not be compared directly. Therefore, we normalize the fitness values relative to the cost of the decentralized cheapest insertion cost heuristic.

5.4.3 Tuning

For investigating the performance of GP we ran some experiments with a smaller number of generations. Figure 5.6 shows a breakdown of the convergence graph of three such runs. The figure shows that most of the improvement during

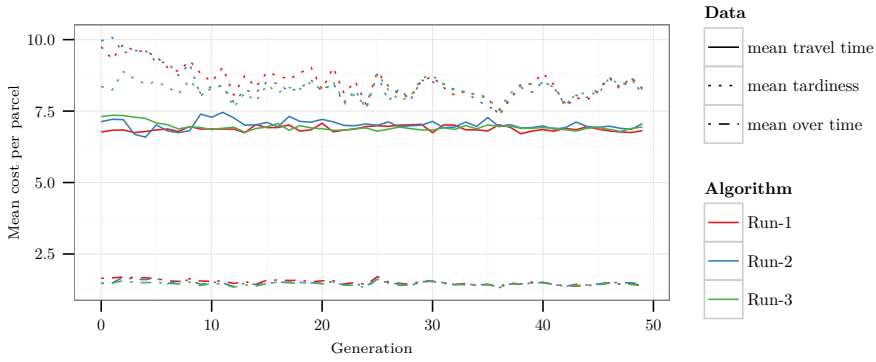


Figure 5.6: Breakdown of cost per generation of three evolutionary runs on a scenario with 50% dynamism, 20 minutes urgency, and scale 1.

evolution is caused by a reduction of tardiness and over time while travel time remains relatively constant. This suggests that it may be worthwhile to emphasize the tardiness in the objective function during evolution. We experimented with two weighted versions of decentralized GP (DGP). DGP-1:1 uses the objective function as defined in eq. 5.4. DGP-1:2 replaces the insertion based GP terminals with weighted versions in favor of tardiness and over time. Figure 5.7 compares the GP runs with two weighted decentralized insertion cost heuristics, DIC-1:2 and DIC-1:4. From Figure 5.7 it can be concluded that DIC-1:2 performs better than the 1:1 objective function while DIC-1:4 performs worse than 1:1. However, replacing the insertion based GP terminals with weighted versions does not benefit evolution, DGP-1:1 outperforms DGP-1:2. This is presumably because evolution already favors heuristics that emphasize reducing tardiness and over time as this yields the greatest performance increase.

5.5 Evaluation

To compare the agent-based hyper-heuristic approach (DGP, Section 5.4) with the MAS using OptaPlanner (DOP, Section 5.3) and the centralized OptaPlanner

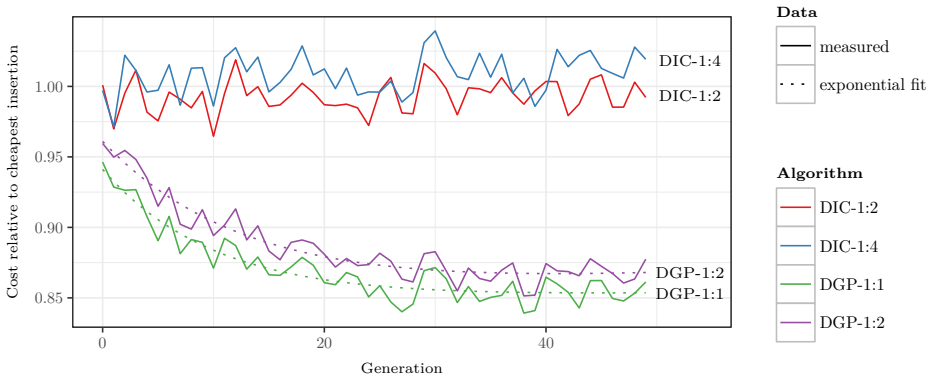


Figure 5.7: Comparison of two evolutionary settings (average of three repetitions each), DGP-1:1 with standard objective function weights of its terminals defined in Table 5.2 and DGP-1:2 with objective functions weights in favor of tardiness and over time. DIC-1:2 and DIC-1:4 are using weighted insertion cost (without evolution) on the same set of scenarios as are used in every generation of the GP.

(COP, Chapter 4) we first need to generate (train) the heuristics that can be used in real-time.

5.5.1 Training

For training we have generated a separate dataset using the same settings (but different random seeds) as used in Chapter 4. During training we use small scale scenarios to save computation time.

Experiment setup

We have opted for four different GP setups (Table 5.4). Three setups are meant to specialize on one specific scenario class, while the DGP-mixed setup aims to generate generalized heuristics that are equally adapted to all scenarios. Because there are nine small scale scenario classes, we use 54, a multiple of nine, evaluations every generation. This ensures that each generation each individual is evaluated on exactly six scenarios of every scenario class.

For the specialized GP runs we need to do $500 \cdot (99 \cdot 50 + 250) = 2{,}600{,}000$ simulations and for the generalized GP run $500 \cdot (99 \cdot 54 + 270) = 2{,}808{,}000$. Since we repeat each setting ten times, the grand total of required simulations is 106,080,000. A single simulation may take from about half a second to

Table 5.4: The four different GP setups. The three specialized setups, DGP-20-35-1, DGP-50-20-1, and DGP-80-5-1, are trained on one specific class of scenarios. DGP-mixed is a setup that is trained on all small scale scenario classes simultaneously.

Dynamism	Urgency	Scale	Num evals	Num last evals	Name
20%	35	1			DGP-20-35-1
50%	20	1	50	250	DGP-50-20-1
80%	5	1			DGP-80-5-1
20%/50%/80%	35/20/5	1	54	270	DGP-mixed

several seconds each on a modern PC. If the average simulation time would be exactly 1 second, the expected total computation time is about 1227 days (3.3 years). Clearly, it is not feasible to run such an experiment on a single computer, therefore we have pooled the resources of about 80 modern quad-core computers to run our simulations. Theoretically, these 80 machines allow us to perform about 320 simulations in parallel. In practice, however, these are shared university machines that may have other processes running or may simply be turned off during an experiment. To utilize these machines we use a feature of RinSim that allows to spread simulations over multiple machines (internally using JPPF (Cohen, 2016)) and that is resistant to single node failures.

Results and analysis

A total of 103,374,996 simulations were computed during the course of the 40 evolutionary runs. The cumulative computation time is 1295 days, using the distributed computing setup, it took slightly more than 10 days. The actual number of simulations that were performed is slightly lower than computed in the previous paragraph because when identical individuals are found within a generation they are evaluated only once.

Figure 5.8 shows the average convergence graphs of each GP variant. For all GP variants, the majority of the improvement occurs in the first 25 generations. It is striking that 80-5-1 shows much less improvement compared to the other variants. This may be explained by the fact that this is probably one of the hardest problems for any algorithm. With 80% dynamism, the problem is changing nearly continuously and with an urgency of 5 minutes, each new order needs to be dealt with swiftly. Based on this graph, it appears that the insertion cost heuristic is performing relatively well in these circumstances. For the 20-35-1 and 50-20-1 settings, GP seems to be able to find the largest improvement relative to the insertion cost heuristic. GP-mixed uses all scenario classes and lies, as expected, somewhere between the others.

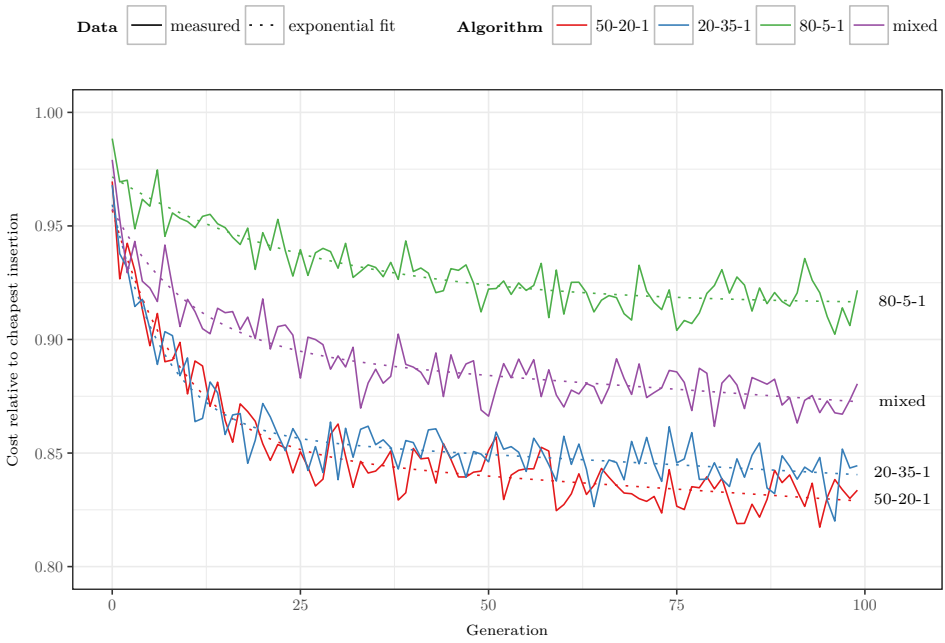


Figure 5.8: Average convergence graphs based on ten repetitions for each of the four GP settings.

5.5.2 Testing

In order to evaluate the effectiveness of our GP approach, we test the evolved heuristics using real-time RinSim (van Lon & Holvoet, 2012) on the same dataset as was used in Chapter 4.

Experiment setup

The test dataset has three levels of dynamism, urgency, and scale, resulting in 27 different scenarios classes. For each class, the dataset contains ten scenario instances. The evolutionary runs (Section 5.5.1) produced 40 heuristics, additionally we are also testing the insertion cost heuristic. This means we have 41 algorithms, each of whom we need to test in real-time on the 270 different scenarios in the dataset, resulting in a total of 11,070 real-time simulations. Unlike Chapter 4, we do not repeat the execution of simulations with exactly

the same settings. Instead, we combine the results of the ten heuristics evolved with the same GP settings and compare those with the results of Chapter 4.

To allow direct comparison of the results, we use the same hard- and software as in Chapter 4. The test computer has 24 logical cores (two six core Intel Xeon 2.6GHz E5-2630 v2 processors with hyper threading). A single simulation requires two logical cores, one for the simulator and one for the solver computations. At least one core needs to be available for the operating system, resulting in a maximum of 11 simulations that can be run in parallel. As in Chapter 4, we warm up the JVM for 30 seconds before starting the real-time experiment.

Results

Table 5.5 lists the algorithms that we compare. Similar to Chapter 4, we

Table 5.5: Algorithm names with their meaning and number of simulations per class that were performed. For COP and DOP, three repetitions were done for each of the ten scenarios in a class. For the other algorithms, no repetitions were done. For the DGP variants, each of the ten evolved heuristics were simulated on each scenario.

Algorithm	Description	Simulations per class
COP	Centralized OptaPlanner (from Chapter 4)	30
DOP	Decentralized OptaPlanner (from Chapter 4)	30
DIC	Decentralized insertion cost	10
DGP-20-35-1	Decentralized GP trained on 20-35-1 class	100
DGP-50-20-1	Decentralized GP trained on 50-20-1 class	100
DGP-80-5-1	Decentralized GP trained on 80-5-1 class	100
DGP-mixed	Decentralized GP trained on all small scale classes	100

apply Welch's t-test for testing the significance of the differences between the algorithms. In the following analysis we refer to this test by mentioning the p-values (when relevant) that were observed. The significance threshold was set at $p = .01$. For pairs of algorithms that have the same number of simulations we perform a paired t-test instead of an unpaired t-test. The total experiment computation time of the 11,070 real-time simulations was about 551.9 hours ($\approx$ 23 days), during this time 11 simulations were run in parallel. Table 5.7 shows all simulation results.

5.5.3 Analysis

The first hypothesis (Section 5.1) states that hyper-heuristics (DGP) can outperform DOP. We can accept this hypothesis as the results indicate that there is always at least one of the DGP variants that outperform DOP (Table 5.6).

Table 5.6: Summary of relative performance of DGP variants to DOP. Each number indicates the number of classes on which the algorithm is (significantly) better or worse compared to DOP.

Algorithm	sign. better	better (not sign.)	worse (not sign.)	sign. worse
DIC	0	6	9	12
DGP-20-35-1	18	0	5	4
DGP-50-20-1	21	8	0	0
DGP-80-5-1	27	0	0	0
DGP-mixed	27	0	0	0

In fact, DGP-mixed, DGP-80-5-1, and DGP-50-20-1 are better than DOP for all scenario classes. Table 5.6 shows that DGP-20-35-1 also often outperforms DOP but not as often. It's also noteworthy that DIC outperforms DOP in several (mostly small scale) classes, indicating that in some cases even a simple heuristic can be better than OptaPlanner.

Table 5.7: Average results for each setting. The 'Best' column indicates which algorithms has the best performance, the rank of each value is indicated by the number in superscript, a † appended to a value with rank n indicates that the difference between the value of rank n and rank $n+1$ is not statistically significant ($p < 0.01$). The results of the four evolved algorithms also report their standard deviation as the numbers are the average of the different heuristics produced by GP.

Class	COP	DOP	DIC	DGP-20-35-1	DGP-50-20-1	DGP-80-5-1	DGP-mixed	Best
20-5-1	$25.100^{1\dagger}$	$28.550^{6\dagger}$	$27.042^{5\dagger}$	$31.025^{7} \pm 13.876$	$26.480^{4\dagger} \pm 1.169$	$25.754^{3} \pm 0.579$	$25.611^{2\dagger} \pm 0.392$	COP†
50-5-1	$22.276^{3\dagger}$	$23.902^{6\dagger}$	$23.218^{5\dagger}$	$25.835^{7} \pm 8.489$	$22.898^{4\dagger} \pm 1.462$	$21.665^{1\dagger} \pm 0.372$	$21.912^{2\dagger} \pm 0.398$	DGP-80-5-1†
80-5-1	$21.481^{2\dagger}$	$23.511^{6\dagger}$	$22.782^{5\dagger}$	$25.969^{7} \pm 10.079$	$22.658^{4\dagger} \pm 1.259$	$21.281^{1\dagger} \pm 0.380$	$21.755^{3} \pm 0.334$	DGP-80-5-1†
20-20-1	$17.692^{1\dagger}$	21.661^{7}	$20.748^{6\dagger}$	$18.987^{4\dagger} \pm 0.363$	$18.705^{2\dagger} \pm 0.300$	$19.152^{5\dagger} \pm 0.330$	$18.869^{3\dagger} \pm 0.306$	COP†
50-20-1	$14.852^{1\dagger}$	17.575^{7}	$16.878^{6\dagger}$	$15.267^{4\dagger} \pm 0.376$	$15.223^{3\dagger} \pm 0.243$	$15.507^{5\dagger} \pm 0.678$	$15.181^{2\dagger} \pm 0.272$	COP†
80-20-1	$14.438^{1\dagger}$	$17.168^{6\dagger}$	17.750^{7}	$15.018^{4} \pm 0.338$	$14.800^{2\dagger} \pm 0.155$	$15.335^{5} \pm 0.491$	$14.866^{3\dagger} \pm 0.296$	COP†
20-35-1	14.520^{1}	19.396^{7}	$18.748^{6\dagger}$	$16.477^{3\dagger} \pm 0.340$	$16.373^{2\dagger} \pm 0.277$	$17.021^{5\dagger} \pm 0.818$	$16.569^{4} \pm 0.278$	COP
50-35-1	12.921^{1}	$17.359^{6\dagger}$	17.636^{7}	$14.436^{2\dagger} \pm 0.453$	$14.543^{3\dagger} \pm 0.321$	$14.963^{5} \pm 0.458$	$14.598^{4\dagger} \pm 0.283$	COP
80-35-1	12.395^{1}	$15.743^{6\dagger}$	16.303^{7}	$13.742^{3\dagger} \pm 0.239$	$13.610^{2\dagger} \pm 0.181$	$14.178^{5} \pm 0.547$	$13.792^{4} \pm 0.342$	COP
20-5-5	$18.809^{3\dagger}$	$20.068^{5\dagger}$	$20.229^{6\dagger}$	$22.217^{7} \pm 9.248$	$19.101^{4\dagger} \pm 2.516$	$17.781^{1\dagger} \pm 0.231$	$17.883^{2\dagger} \pm 0.288$	DGP-80-5-1†
50-5-5	17.131^{5}	16.565^{4}	$18.590^{6\dagger}$	$18.616^{7} \pm 6.244$	$16.005^{3\dagger} \pm 1.834$	$14.762^{1\dagger} \pm 0.180$	$14.884^{2} \pm 0.291$	DGP-80-5-1†
80-5-5	$17.249^{5\dagger}$	16.402^{4}	18.549^{7}	$18.485^{6\dagger} \pm 6.164$	$15.912^{3\dagger} \pm 1.723$	$14.664^{1\dagger} \pm 0.160$	$14.790^{2} \pm 0.231$	DGP-80-5-1†
20-20-5	$13.987^{3\dagger}$	$16.904^{6\dagger}$	17.656^{7}	$13.941^{2\dagger} \pm 0.303$	$13.833^{1\dagger} \pm 0.139$	$14.690^{5} \pm 0.730$	$14.034^{4} \pm 0.263$	DGP-50-20-1†
50-20-5	$10.198^{4\dagger}$	11.615^{6}	14.176^{7}	$9.756^{3\dagger} \pm 0.186$	$9.497^{1} \pm 0.115$	$10.297^{5} \pm 0.725$	$9.749^{2\dagger} \pm 0.147$	DGP-50-20-1
80-20-5	$10.329^{4\dagger}$	11.837^{6}	14.851^{7}	$10.082^{3} \pm 0.238$	$9.823^{1} \pm 0.163$	$10.613^{5} \pm 0.729$	$10.044^{2\dagger} \pm 0.196$	DGP-50-20-1
20-35-5	$10.967^{1\dagger}$	$14.097^{6\dagger}$	15.555^{7}	$11.083^{2} \pm 0.184$	$11.289^{4} \pm 0.188$	$11.938^{5} \pm 0.660$	$11.277^{3\dagger} \pm 0.220$	COP†
50-35-5	$8.677^{1\dagger}$	11.326^{6}	14.443^{7}	$8.884^{2\dagger} \pm 0.203$	$8.973^{3\dagger} \pm 0.246$	$9.718^{5} \pm 0.674$	$9.057^{4} \pm 0.201$	COP†
80-35-5	8.877^{1}	11.206^{6}	14.817^{7}	$9.099^{2} \pm 0.247$	$9.251^{4} \pm 0.220$	$9.922^{5} \pm 0.614$	$9.247^{3\dagger} \pm 0.202$	COP
20-5-10	17.587^{4}	$17.929^{5\dagger}$	$18.926^{6\dagger}$	$20.166^{7} \pm 7.779$	$17.156^{3\dagger} \pm 2.665$	$15.858^{1\dagger} \pm 0.142$	$15.929^{2} \pm 0.313$	DGP-80-5-1†
50-5-10	$15.681^{5\dagger}$	14.588^{4}	17.517^{7}	$16.828^{6\dagger} \pm 6.557$	$13.917^{3} \pm 1.828$	$12.835^{2} \pm 0.181$	$12.828^{1\dagger} \pm 0.328$	DGP-mixed†
80-5-10	$15.898^{5\dagger}$	14.446^{4}	17.783^{7}	$16.518^{6\dagger} \pm 5.572$	$14.100^{3\dagger} \pm 1.828$	$12.895^{1\dagger} \pm 0.160$	$12.935^{2} \pm 0.373$	DGP-80-5-1†
20-20-10	11.588^{5}	$13.320^{6\dagger}$	15.043^{7}	$10.758^{1\dagger} \pm 0.221$	$10.776^{2} \pm 0.116$	$11.554^{4\dagger} \pm 0.717$	$10.939^{3} \pm 0.198$	DGP-20-35-1†
50-20-10	$9.329^{4\dagger}$	10.649^{6}	14.260^{7}	$8.799^{2\dagger} \pm 0.247$	$8.585^{1} \pm 0.067$	$9.478^{5} \pm 0.808$	$8.799^{3} \pm 0.149$	DGP-50-20-1
80-20-10	$9.167^{4\dagger}$	10.469^{6}	14.149^{7}	$8.710^{2\dagger} \pm 0.253$	$8.489^{1} \pm 0.114$	$9.347^{5} \pm 0.763$	$8.727^{3} \pm 0.136$	DGP-50-20-1
20-35-10	$9.787^{4\dagger}$	$11.990^{6\dagger}$	13.983^{7}	$9.158^{1} \pm 0.186$	$9.437^{3\dagger} \pm 0.313$	$10.069^{5} \pm 0.641$	$9.405^{2\dagger} \pm 0.234$	DGP-20-35-1
50-35-10	$7.827^{1\dagger}$	10.053^{6}	14.004^{7}	$7.838^{2} \pm 0.203$	$8.060^{4} \pm 0.409$	$8.744^{5} \pm 0.695$	$8.044^{3\dagger} \pm 0.212$	COP†
80-35-10	$7.870^{2\dagger}$	9.879^{6}	14.078^{7}	$7.767^{1\dagger} \pm 0.187$	$7.984^{4} \pm 0.389$	$8.582^{5} \pm 0.683$	$7.913^{3\dagger} \pm 0.196$	DGP-20-35-1†
Avg. rank	2.7	5.74	6.56	3.81	2.74	3.74	2.7	

The differences between DGP-mixed and DOP and between DGP-80-5-1 and DOP are always significant, even for large scale scenarios. This is interesting because the heuristics were never trained on large scale scenarios. It appears that the evolved heuristic has no problem scaling up to large problem instances. To investigate whether the heuristic's scalability can be explained by its supposed computational efficiency, we have measured the computational runtimes within a single simulation of both the DOP as well as the DGP-50-20-1 on a scenario with class 50-20-10 (Figure 5.9). The big gap between DGP-50-20-1 and DOP

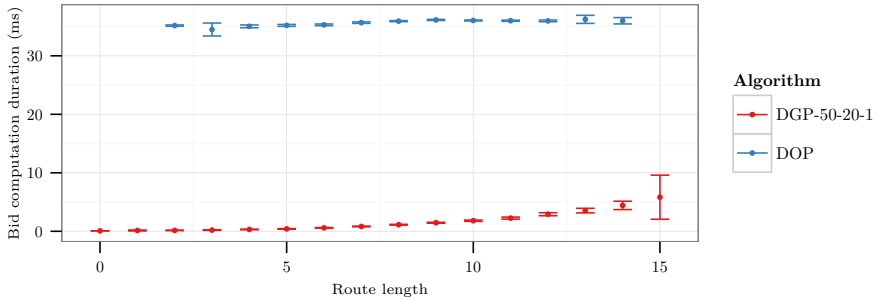


Figure 5.9: Average bid computation times for both the DGP-50-20-1 and DOP on a single scenario of class 50-20-10. The error bars indicate the 95% confidence interval. There are no values for route length 0 and 1 for DOP because it is unnecessary to let the OptaPlanner solver compute an insertion in this case. Note that up until route length 7, the average computation time for DGP-50-20-1 is below 1 ms.

is caused by the unimproved time parameter of the OptaPlanner solver. This parameter determines the period the solver keeps searching while it has not found an improving solution. In Chapter 4, unimproved time is set to 20 ms, which explains why the fastest computation time of DOP is always higher than 20 ms. Additionally, Figure 5.9 shows that the DGP heuristic is fast and growing at a low rate (averages range from 0.074 to 5.827 milliseconds between route length 0 and 15).

To investigate the influence of the bid computation time on the performance of the DGP approach, we conducted an additional experiment where we artificially delayed the computation of the heuristic. This experiment was carried out with a single DGP-50-20-1 heuristic and on all ten scenarios in the 50-20-10 class. Table 5.8 indicates that the computational efficiency of the DGP approach is a contributing factor for its relatively good performance. With a delay of 100 milliseconds, DGP-50-20-1 still outperforms DOP. However, with a delay of 200 milliseconds, DGP-50-20-1 performs worse compared to DOP. Based on this observation we conclude that the quality of cost estimations made by the

Table 5.8: Average cost of DOP with a single heuristic from DGP-50-20-1 with and without an artificial bid computation delay.

Class	Algorithm	Cost
50-20-10	DOP	10.649
50-20-10	DGP-50-20-1 no delay	8.634
50-20-10	DGP-50-20-1 100 ms delay	9.251
50-20-10	DGP-50-20-1 200 ms delay	11.102

evolved heuristics must be higher than the estimations made by DOP.

The second hypothesis states that DGP can outperform COP. This hypothesis can be accepted since the evolved heuristics regularly outperform COP (Table 5.9). However, COP still performs best in 11 of the 27 classes. The scale and urgency of a problem seem to be good indicators of the relative performance of the DGP approaches and COP. The more urgent and large scale a problem is, the better the DGP approaches perform.

Table 5.9: Summary of relative performance of DGP variants to COP. Each number indicates the number of classes on which the algorithm is (significantly) better or worse compared to COP.

Algorithm	sign. better	better (not sign.)	worse (not sign.)	sign. worse
DIC	0	0	8	19
DGP-20-35-1	3	5	11	8
DGP-50-20-1	8	4	10	5
DGP-80-5-1	5	4	10	8
DGP-mixed	8	5	10	4

The third hypothesis states that the evolved heuristics perform especially well in more urgent circumstances. Based on Table 5.7 it is clear that evolved heuristics outperform COP in eight of the nine very urgent classes (urgency of five minutes), we can therefore accept this hypothesis. The class where COP is better than the evolved heuristics is 20-5-1. In this class, COP is not significantly different from DGP-mixed ($p \approx .45$), DGP-80-5-1 ($p \approx .34$), and DGP-50-20-1 ($p \approx .05$). Additionally, we expect that the fact that the heuristics have relatively short computation times is a stronger factor in more urgent scenarios, because the available computation time is shorter.

Hypothesis four states that specialized heuristics outperform general heuristics on scenarios for which they are specialized. DGP-20-35-1 outperforms DGP-mixed on class 20-35-1 (but not significantly, $p \approx .54$), however, DGP-50-20-1 performs best of the evolved heuristics on this class. Surprisingly, DGP-mixed outperforms DGP-50-20-1 on its training class, 50-20-1, although

not significantly ($p \approx .76$). A possible explanation for the good performance of DGP-mixed in this case is that it is trained on all small scale scenarios and 50-20-1 is an 'average' scenario, it has medium dynamism and medium urgency. DGP-80-5-1 performs best on its training class 80-5-1, the difference with DGP-mixed is significant ($p \approx .004$). So, for all three classes on which was trained explicitly, the specialized heuristic significantly outperforms the general heuristic only once. The difference is not significant in two other cases, therefore we reject the hypothesis. We conclude that in some situations a general heuristic can perform comparably to a specialized heuristic. Our results seem to conform to the 'no free lunch theorem' (Wolpert & Macready, 1997). The urgency on which a heuristic was trained is a strong indicator of how well it will perform on a scenario class. We created a summary of the relative performance of the DGP variants, grouped by urgency (Table 5.10). The table shows that each

Table 5.10: Summary of relative performance of DGP variants per urgency level. Each number indicates the number of classes on which the algorithm is the best DGP approach for that urgency level.

Urgency	DGP-20-35-1	DGP-50-20-1	DGP-80-5-1	DGP-mixed
5	0	0	7	2
20	1	7	0	1
35	7	2	0	0

specialized heuristic performs best on seven out of nine classes that have the urgency level on which the heuristic was trained.

The fifth hypothesis states that generalized heuristics outperform specialized heuristics on scenarios for which they are not specialized. Based on Table 5.10 we can reject this hypothesis. There are only three classes, 20-5-1, 50-20-1, and 50-5-10, where DGP-mixed outperforms the other evolved heuristics. This result is somewhat surprising, especially since the number of evaluations for mixed is slightly more than for the specialized heuristics (54 vs 50 evaluations). Nevertheless, the DGP-mixed method produces heuristics of good quality as is demonstrated by its average rank of 2.7 which is the best average rank shared by COP. However, when computing the average ranks of only the four DGP heuristics, DGP-50-20-1 has the same rank as DGP-mixed (average rank 2.19).

As expected, DIC is on average the worst performing algorithm. There are, however, several cases where DGP-20-35-1 performs worse compared to DIC. The data in Table 5.7 shows that DGP-20-35-1 performs among the worst in the most urgent scenarios. This is expected considering that it was trained on the least urgent scenarios. The results of this heuristic are made even worse by one instance that performs especially bad (as can be seen by the larger than usual standard deviations). When removing this badly performing heuristic

from the analysis, the ranks for the DGP-20-35-1 are still among the worst for the very urgent classes, but the values are much closer to that of DIC. This indicates that the bad performance is not just explained by this one outlier.

5.5.4 Reproducibility

Following the policy in (van Lon & Holvoet, 2013) we open-sourced all software that was written for the research described in this chapter and made it available online. The scripts for running each experiment described in this chapter can be found in (van Lon, 2017c). All resulting data, including the scripts that we used for the analysis, as well as visualizations of all heuristics, are available in (van Lon, 2017d). This code depends on several other open source projects that we developed. For all simulations we used RinSim version 4.3.0 (van Lon, 2016d). The code for the OptaPlanner based algorithms is part of RinLog version 3.2.0 (van Lon, 2016c), the evolutionary algorithms related code can be found in (van Lon, 2017b,e). The scenario files that we generated for training were generated using our dataset generator (van Lon, 2016b).

5.6 Conclusion

Agents in a multi-agent system typically compute decisions using traditional optimization algorithms. We have investigated an alternative approach based on hyper-heuristics. The present chapter is the first to evaluate the performance of an agent-based hyper-heuristic approach on a real-time logistics problem that systematically varies the dynamism, urgency, and scale of the problem. The results show that our hyper-heuristic outperforms a reference algorithm, based on the OptaPlanner optimization library, in all scenarios. In addition, the decentralized hyper-heuristic approach even outperforms the centralized reference algorithm in most situations. The hyper-heuristic approach performs relatively better on more urgent and larger scale problems. The hyper-heuristic approach has the additional advantage that it can specialize on certain problem characteristics, increasing its performance even further.

We see three interesting directions for future work. The first direction is to make the logistics simulator even more realistic. Examples that will improve realism are, using a road layout of a city, using real-world customer data, having a heterogeneous fleet of vehicles, or, imposing fuel constraints. The goal of increasing realism is to evaluate whether the hyper-heuristic agent approach can outperform traditional algorithms in real-world conditions, hopefully leading to their eventual deployment. The second research direction is to investigate

different team compositions and level of selection in the evolutionary process. The work done by Waibel et al. (2009) seems to be applicable to evolutionary designing multi-agent systems for logistics problems. A possible hypothesis in a heterogeneous setup could be the emergence of agent specializations. For example, agents could optimize towards pickup and deliveries in a specific geographical area, such as inner city versus rural areas. Thirdly, in the current hyper-heuristic setup, the parts of agents that are subject to evolution are relatively small. The agent behavior as well as the contract-net coordination protocol are predetermined. A very interesting line of work would be to give evolution more freedom. A challenge would be to determine a set of basic coordination or communication building blocks. Using these building blocks, evolution could start exploring in the space of possible coordination mechanisms. It would be interesting to see if evolution would create novel coordination mechanisms or if it would reinvent existing coordination mechanisms. Since we made all our algorithms and results freely available, we believe that the present chapter provides an ideal starting point for any of these future research directions.

Chapter 6

Conclusion

A widely held belief in multi-agent systems literature is that MAS's are advantageous in operational research problems that are very dynamic and/or large scale. However, such claims were never supported by evidence based on a systematic empirical study. This dissertation is the first to systematically investigate and quantify the influence of dynamism, urgency, and scale on the performance of both MAS's and centralized algorithms. In the remainder of this chapter we summarize the contributions in more detail (Section 6.1), we reflect on the lessons learned (Section 6.2), and we look forward to interesting directions for future work (Section 6.3).

6.1 Summary of contributions

The five main contributions of this dissertation and their conclusions are discussed in this section.

6.1.1 Measures of dynamic pickup-and-delivery problems

We argue that urgency and dynamism are conceptually different and we propose separate measures for both concepts. In support of this conceptual separation, the experimental results show that the degree of dynamism and urgency have a different influence on the solution quality in dynamic logistic problems. Interestingly, the degree of dynamism is negatively correlated with operating costs while more urgent scenarios are correlated with significantly

higher operating costs. Additionally, we define scale as a multiplier applied to the number of vehicles and orders in a problem. These three formal definitions allow to design experiments that investigate the influence of one property in isolation of the others. The effect of variations of a problem property on algorithm performance can therefore be quantified independently, enabling insight into algorithm performance.

6.1.2 Dataset

Based on the formal definitions of dynamism, urgency, and scale, a benchmark dataset and problem instance generator were presented. To avoid any interactions between the variables, the dataset generator is constructed meticulously. The generated benchmark dataset allows systematic comparison of algorithms. By open sourcing the dataset generator, other researchers are enabled to create their own datasets and conduct new investigations.

6.1.3 A realistic simulation platform

The real-time logistics simulator, RinSim, is a technical contribution. The simulator has support for both decentralized MAS's as well as centralized algorithms and supports the dataset with different levels of dynamism, urgency, and scale. Both the centralized as well as the decentralized interface of RinSim provide the same software limitations and hardware constraints, thereby providing a fair environment for comparing performance. We have demonstrated that fluctuations caused by the real-time nature of the simulator have a minimal impact on the end result. RinSim is entirely open-source to support reproducibility of all experiments and to allow extensibility of all components.

6.1.4 Systematic evaluation of centralized algorithms and decentralized multi-agent systems

A MAS based on CNET and a centralized tabu search algorithm, based on the OptaPlanner optimization library, are implemented. Using the measures, dataset, and RinSim, a systematic evaluation of these two implementations is conducted. This evaluation experiment is the first of its kind to compare the influence of dynamism, urgency, and scale on the performance of two classes of algorithms in such a systematic, thorough, and fair manner. The results of the comparison show that the solutions found by the centralized algorithm cost, on average, only 94.2% of the cost of the solutions found by the MAS. This

indicates that the centralized algorithm generally performs better compared to the multi-agent system. However, for scenarios that are medium to very dynamic, very urgent, and medium to large scale, the average relative cost of the centralized algorithm is 112.3%, indicating that under these circumstances, the multi-agent system performs better compared to the centralized algorithm. When assessing the performance of the algorithms individually per scenario property, there is not one algorithm that generally outperforms the other on that dimension. The code of the algorithms as well as the experiment results data are published to allow complete reproducibility of the evaluation. Because all components are completely open source, this evaluation provides a baseline of performance comparisons between centralized and decentralized algorithms.

6.1.5 Genetic programming of multi-agent systems

Based on the evaluation of centralized and decentralized algorithms, an investigation on optimizing MAS was conducted. The main hypothesis is that hyper-heuristics, more specifically GP, can be used to improve agents decentrally coordinated via CNET. The heuristic that is evolved by GP is used as agent bid function in the auction process of CNET. The results show that our hyper-heuristic outperforms a reference algorithm, based on the OptaPlanner optimization library, in all scenarios. In addition, the decentralized hyper-heuristic approach even outperforms the centralized reference algorithm in most situations. The hyper-heuristic approach performs relatively better on more urgent and larger scale problems. The hyper-heuristic approach has the additional advantage that it can specialize on certain problem characteristics, increasing its performance even further. This contribution, the combination of hyper-heuristics and MAS, provides a first step towards the automatic design of MAS's.

6.2 Lessons learned and discussion

Conducting an empirical study for comparing distinct algorithms is a tedious task. We have formally defined the pickup-and-delivery problem, including the scenario properties: dynamism, urgency, and scale. For the algorithms we used OptaPlanner, a well known satisfaction solver library. A tuning experiment was conducted to find the best performing OptaPlanner algorithm for this problem. The best algorithm was incorporated in an online centralized algorithm and a multi-agent system based on the dynamic contract-net protocol. In order to perform a fair empirical study it is imperative to use a real-time simulator that assigns the same processing power to the approaches. For this reason

we have extended the RinSim logistics simulator and have demonstrated that fluctuations caused by the real-time nature of the simulator have a minimal impact on the end result.

Reproducibility is one of the main principles of the scientific method. Unfortunately, this principle is not yet the default practice among researchers in multi-agent systems and operational research. The work described in this dissertation would have greatly benefited from an existing benchmark of a dynamic logistic problem that could be targeted by both centralized as well as decentralized algorithms. Unfortunately such a benchmark did not exist. Additionally, attempts at reusing algorithms based on existing papers was not successful due to their lack of reproducibility. Although facilitating reproducibility can be tedious, it would become much less so were this the default practice. Due to the focus on reproducibility, we believe that this dissertation provides many points of extensibility and is therefore an ideal starting point for researchers interested in dynamic logistics, empirical evaluation, multi-agent systems, or genetic programming.

In practice, a multi-agent system can be deployed in various ways. A natural deployment of a CNET MAS is to let customers and vehicle drivers use local hardware (such as a mobile phone) for running their agents. Using a decentralized communication middleware it is then possible to create a decentralized and distributed system. An alternative deployment is to use a centralized computer which simulates the agents and controls all vehicles remotely. A disadvantage of this approach is that it has a single point of failure (the centralized computer), when this server goes down, the entire system would fail. This is contrary to a distributed deployment which is more robust to hardware failures. It would be interesting to study the effect of deployment on centralized and decentralized algorithms.

Because of the real-time nature of the experiments described in this dissertation, the results are influenced by the speed of the hardware. The results indicate that the CNET MAS performs better when there is little time for computations (urgent problems) and the amount of computations that need to be done is large (larger scale problems). It seems likely that when hardware capabilities increase over time, the relative performance of the CNET MAS will decrease when compared to the centralized OptaPlanner algorithm. When the available computing power increases, the absolute performance of the algorithms is likely to converge to the optimum, reducing the performance difference between the algorithms.

6.3 Future work

While working on the research described in this dissertation, new ideas for interesting follow-up research emerged. In this section we describe the most promising directions for future work.

- The research described in this dissertation constitutes an important step towards more realistic experiments with multi-agent systems in logistics. There is, however, still room for improvement. There are several ways in which RinSim can be made even more realistic, e.g.: using a road layout of a city, using real-world customer data, having a heterogeneous fleet of vehicles, imposing cargo constraints, imposing fuel constraints, and simulating realistic deployment scenarios. Such a realistic evaluation would provide evidence necessary for a deployment by a commercial company.
- Many other MAS coordination protocols exist, such as Delegate MAS and Gradient Field, that can be evaluated and compared to the algorithms used in the present dissertation. Similarly, there are many more centralized algorithms and libraries that implement them. This dissertation provides a benchmark, an ideal starting point for further research into more advanced algorithms.
- In the experiments described in this dissertation, it is assumed that all vehicles are part of a single transportation company. Therefore, all agents have an incentive to cooperate. In a setup with multiple competing companies, agents might make selfish choices that are detrimental to the interests of all agents. Creating a MAS for the dynamic PDPTW in a competitive setup requires a game-theoretic approach that ensures that agents have incentives for cooperation.
- The work done by Waibel et al. (2009) on team composition and level of selection in the evolutionary process seems to be relevant in the context of MAS for logistics. In a heterogeneous setup, agents in a MAS can have different control programs, a possible hypothesis could be the emergence of agent specializations. For example, agents could optimize towards pickup and deliveries in a specific geographical area, such as inner city versus rural areas.
- The experiments in Chapter 5 show that the agent-based hyper-heuristic approach outperforms the agent-based OptaPlanner approach and also often outperforms the centralized OptaPlanner approach. It would be interesting to investigate whether a centralized hyper-heuristic approach (or other machine learning approach) could be devised and if that

could perform even better. A simple strategy would be to execute the decentralized approach centrally. The expected performance of such a system is equivalent to the decentralized hyper-heuristic approach. This approach could then be used as a baseline to further improve the centralized hyper-heuristic algorithm.

- In the current hyper-heuristic setup, the parts of agents that are subject to evolution is relatively small. The agent behavior as well as the contract-net coordination protocol are predetermined. A very interesting line of work would be to give evolution more freedom. A challenge would be to determine a set of basic coordination and communication building blocks. Using these building blocks, evolution could start exploring in the space of possible coordination mechanisms. It would be interesting to see if evolution would create novel coordination mechanisms or if it would reinvent existing coordination mechanisms.

Bibliography

Arlitt, M. & Williamson, C. (1997). Internet Web servers: workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5), 631–645. doi:10.1109/90.649565.

Beham, A., Kofler, M., Wagner, S., & Affenzeller, M. (2009). Agent-Based Simulation of Dispatching Rules in Dynamic Pickup and Delivery Problems. *2009 2nd International Symposium on Logistics and Industrial Informatics*, (pp. 1–6). doi:10.1109/LINDI.2009.5258763.

Benyahia, I. & Potvin, J. (1998). Decision support for vehicle dispatching using genetic programming. *IEEE Transactions On Systems Man and Cybernetics Part A - Systems And Humans*, 28(3), 306–314. doi:10.1109/3468.668962.

Berbeglia, G., Cordeau, J.-F., & Laporte, G. (2010). Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1), 8–15. doi:10.1016/j.ejor.2009.04.024.

Borndörfer, R., Grötschel, M., Klostermeier, F., & Küttner, C. (1999). *Telebus Berlin: Vehicle scheduling in a dial-a-ride system*, volume 471 of *Lecture Notes in Economics and Mathematical Systems*. doi:10.1007/978-3-642-85970-0_19.

Branke, J., Nguyen, S., Pickardt, C. W., & Zhang, M. (2016). Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 20(1), 110–124. doi:10.1109/TEVC.2015.2429314.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12), 1695–1724. doi:10.1057/jors.2013.71.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Woodward, J. R. (2010). *A Classification of Hyper-heuristic Approaches*, (pp. 449–468). Springer US: Boston, MA.

Burke, E. K., Hyde, M. R., & Kendall, G. (2006). *Evolving Bin Packing Heuristics with Genetic Programming*, (pp. 860–869). Springer Berlin Heidelberg: Berlin, Heidelberg. doi:10.1007/11844297_87.

Cohen, L. (2016). JPPF, the open source grid computing solution. `http://jppf.org/`.

Cordeau, J.-F. & Laporte, G. (2003). The dial-a-ride problem (darp): Variants, modeling issues and algorithms. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(2), 89–101. doi:10.1007/s10288-002-0009-8.

Coslovich, L., Pesenti, R., & Ukovich, W. (2006). A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. *European Journal of Operational Research*, 175(3), 1605 – 1615. doi:10.1016/j.ejor.2005.02.038.

Cruz, C., González, J. R., & Pelta, D. A. (2011). Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*, 15(7), 1427–1448. doi:10.1007/s00500-010-0681-0.

Dantzig, G. & Ramser, J. (1959). The truck dispatching problem. *Management Science*, 6(1), 80–91. doi:10.1287/mnsc.6.1.80.

De Smet et al., G. (2016). OptaPlanner User Guide. OptaPlanner is an open source constraint satisfaction solver in Java.

Dinh, H. T., van Lon, R. R. S., & Holvoet, T. (2016). Multi-Agent Route Planning Using Delegate MAS. In *ICAPS Proceedings of the 4th Workshop on Distributed and Multi-Agent Planning (DMAP-2016)* (pp. 24–32).

Dorer, K. & Calisti, M. (2005). An adaptive solution to dynamic transport optimization. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05 (pp. 45–51). New York, NY, USA: ACM. doi:10.1145/1082473.1082803.

Eiben, A. E. & Smith, J. E. (2007). *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer-Verlag Berlin Heidelberg, corrected edition. doi:10.1007/978-3-662-05094-1.

Eksioglu, B., Vural, A. V., & Reisman, A. (2009). The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4), 1472–1483. doi:10.1016/j.cie.2009.05.009.

Fischer, K., Müller, J. P., & Pischel, M. (1995). A model for cooperative transportation scheduling. In *Proc. of the 1st Int. Conf. on Multiagent Systems (ICMAS'95)* (pp. 109–116). San Francisco.

Flood, M. (1956). The Traveling-salesman problem. *Operations Research*, 4(1), 61–75. doi:10.1287/opre.4.1.61.

Fu, L. (2002). Scheduling dial-a-ride paratransit under time-varying, stochastic congestion. *Transportation Research Part B-Methodological*, 36(6), 485–506. doi:10.1016/S0191-2615(01)00014-5.

Gendreau, M., Guertin, F., Potvin, J.-Y., & Séguin, R. (2006). Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, 14(3), 157–174. doi:10.1016/j.trc.2006.03.002.

Gendreau, M. & Potvin, J.-Y. (1998). Dynamic vehicle routing and dispatching. In T. Crainic & G. Laporte (Eds.), *Fleet Management and Logistics*, Centre for Research on Transportation (pp. 115–126). Springer US. doi:10.1007/978-1-4615-5755-5_5.

Glaschenko, A., Ivaschenko, A., Rzevski, G., & Skobelev, P. (2009). Multi-agent real time scheduling system for taxi companies. In *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)* (pp. 29–36).

Gunther, N. J. (2006). *Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services*. Secaucus, NJ, USA: Springer-Verlag New York, Inc. doi:10.1007/978-3-540-31010-5.

Hanif, S., van Lon, R. R. S., Gui, N., & Holvoet, T. (2011). Delegate mas for large scale and dynamic pdp: A case study. In *Intelligent Distributed Computing V*, volume 5 (pp. 23–33).

Holvoet, T., Weyns, D., & Valckenaers, P. (2009). Patterns of delegate mas. In *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems* (pp. 1–9). doi:10.1109/SASO.2009.31.

Ince, D. C., Hatton, L., & Graham-Cumming, J. (2012). The case for open computer programs. *Nature*, 482(7386), 485–488. doi:10.1038/nature10836.

Keller, R. E. & Poli, R. (2008). *Cost-Benefit Investigation of a Genetic-Programming Hyperheuristic*, (pp. 13–24). Springer Berlin Heidelberg: Berlin, Heidelberg. doi:10.1007/978-3-540-79305-2_2.

Kilby, P., Prosser, P., & Shaw, P. (1998). *Dynamic VRPs: A study of scenarios*. Technical report, University of Strathclyde.

Koza, J. R. (1994). *Genetic programming II: Automatic discovery of reusable subprograms*. MIT Press.

Larsen, A. (2000). *The Dynamic Vehicle Routing Problem.* PhD thesis, Technical University of Denmark (DTU).

Larsen, A., Madsen, O., & Solomon, M. (2002). Partially dynamic vehicle routing - models and algorithms. *Journal of the Operational Research Society*, 53(6), 637–646. doi:10.1057/palgrave.jors.2601352.

Law, A. M. (2007). *Simulation modeling and analysis.* McGraw-Hill, fourth edition.

Lewis, P. A. W. & Shedler, G. S. (1979). Simulation of nonhomogeneous poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3), 403–413. doi:10.1002/nav.3800260304.

Li, H. & Lim, A. (2001). A metaheuristic for the pickup and delivery problem with time windows. In *Tools with Artificial Intelligence, Proceedings of the 13th International Conference on* (pp. 160–167). doi:10.1109/ICTAI.2001.974461.

Luke, S., Panait, L., Balan, G., Paus, S., Skolicki, Z., Kicinger, R., Popovici, E., Sullivan, K., Harrison, J., Bassett, J., Hubley, R., Desai, A., Chircop, A., Compton, J., Haddon, W., Donnelly, S., Jamil, B., Zelibor, J., Kangas, E., Abidi, F., Mooers, H., O'Beirne, J., Talukder, K. A., McKay, S., & McDermott, J. (2011). ECJ 20: a java-based evolutionary computation and genetic programming research system. `https://cs.gmu.edu/~eclab/projects/ecj/`.

Lund, K., Madsen, O. B. G., & Rygaard, J. M. (1996). *Vehicle routing problems with varying degrees of dynamism.* Technical report, IMM Institute of Mathematical Modelling.

Madsen, O. B. G., Ravn, H. F., & Rygaard, J. M. (1995). A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60(1), 193–208. doi:10.1007/BF02031946.

Máhr, T., Srour, J. F., de Weerdt, M., & Zuidwijk, R. (2008). Agent performance in vehicle routing when the only thing certain is uncertainty. In *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)* Estorial, Portugal.

Máhr, T., Srour, J. F., de Weerdt, M., & Zuidwijk, R. (2010). Can agents measure up? A comparative study of an agent-based and on-line optimization approach for a drayage problem with uncertainty. *Transportation Research: Part C*, 18(1), 99–119. doi:10.1016/j.trc.2009.04.018.

Merlevede, J., van Lon, R. R. S., & Holvoet, T. (2014). Neuroevolution of a multi-agent system for the dynamic pickup and delivery problem. In *International Joint Workshop on Optimisation in Multi-Agent Systems and Distributed Constraint Reasoning (OptMAS, co-located with AAMAS)*.

Merriam Webster (2014). "dynamic." *Merriam-Webster.com*.

Mes, M., van der Heijden, M., & Schuur, P. (2010). Look-ahead strategies for dynamic pickup and delivery problems. *OR Spectrum*, 32(2), 395–421. doi:10.1007/s00291-008-0146-3.

Mes, M., van der Heijden, M., & Schuur, P. (2013). Interaction between intelligent agent strategies for real-time transportation planning. *Central European Journal of Operations Research*, 21(2), 337–358. doi:10.1007/s10100-011-0230-7.

Mes, M., van der Heijden, M., & van Harten, A. (2007). Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. *European Journal of Operational Research*, 181(1), 59–75. doi:10.1016/j.ejor.2006.02.051.

Mitrović-Minić, S., Krishnamurti, R., & Laporte, G. (2004). Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B - Methodological*, 38(8), 669–685. doi:10.1016/j.trb.2003.09.001.

Mitrović-Minić, S. & Laporte, G. (2004). Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B - Methodological*, 38(7), 635–655. doi:10.1016/j.trb.2003.09.002.

Nguyen, T. T., Yang, S., & Branke, J. (2012). Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6, 1 – 24. doi:10.1016/j.swevo.2012.05.001.

Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2008). A survey on pickup and delivery problems. Part II: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58(2), 81–117. doi:10.1007/s11301-008-0036-4.

Pěchouček, M. & Mařík, V. (2008). Industrial deployment of multi-agent technologies: review and selected case studies. *Autonomous Agents and Multi-Agent Systems*, 17(3), 397–431. doi:10.1007/s10458-008-9050-0.

Pillac, V., Gendreau, M., Gueret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1), 1–11. doi:10.1016/j.ejor.2012.08.015.

Potvin, J., Dufour, G., & Rousseau, J. (1993). Learning Vehicle Dispatching with Linear-Programming Models. *Computers & Operations Research*, 20(4), 371–380. doi:10.1016/0305-0548(93)90081-S.

Preisler, T., Dethlefs, T., & Renz, W. (2015). Data-adaptive simulation: Cooperativeness of users in bike-sharing systems. In W. Kersten, T. Blecker, & C. M. Ringle (Eds.), *Innovations and Strategies for Logistics and Supply Chains* (pp. 1765–1772).

Preisler, T., Dethlefs, T., & Renz, W. (2016). Self-organizing redistribution of bicycles in a bike-sharing system based on decentralized control. In *Federated Conference on Computer Science and Information Systems*, volume 8 (pp. 1471–1480).: ACSIS. doi:10.15439/2016F126.

Psaraftis, H. (1980). A dynamic-programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2), 130–154. doi:10.1287/trsc.14.2.130.

Psaraftis, H. (1995). Dynamic vehicle routing: Status and prospects. *Annals of Operations Research*, 61, 143–164. doi:10.1007/BF02098286.

Psaraftis, H. N. (1983). k-Interchange procedures for local search in a precedence-constrained routing problem. *European Journal of Operational Research*, 13(4), 391–402. doi:10.1016/0377-2217(83)90099-1.

Pureza, V. & Laporte, G. (2008). Waiting and Buffering Strategies for the Dynamic Pickup and Delivery Problem with Time Windows. *INFOR*, 46(3), 165–175. doi:10.3138/infor.46.3.165.

Savelsbergh, M. W. P. & Sol, M. (1995). The General Pickup and Delivery Problem. *Transportation Science*, 29(1), 17–29. doi:10.1287/trsc.29.1.17.

Shen, W., Hao, Q., Yoon, H. J., & Norrie, D. H. (2006). Applications of agent-based systems in intelligent manufacturing: An updated review. *Advanced Engineering Informatics*, 20(4), 415 – 431. doi:10.1016/j.aei.2006.05.004.

Shen, Y., Potvin, J., Rousseau, J., & Roy, S. (1995). A computer assistant for vehicle dispatching with learning capabilities. *Annals of Operations Research*, 61, 189–211. doi:10.1007/BF02098288.

Smith, R. G. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12), 1104–1113. doi:10.1109/TC.1980.1675516.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2), 254–265. doi:10.1287/opre.35.2.254.

van den Haak, P., van Lon, R. R. S., van der Meer, J., & Rothkrantz, L. (2010). Stress assessment of car-drivers using eeg-analysis. In *Proceedings of the 11th International Conference on Computer Systems and Technologies* (pp. 473–477).

van der Heijden, M., Ebben, M., Gademann, N., & van Harten, A. (2002). Scheduling vehicles in automated transportation systems algorithms and case study. *OR Spectrum*, 24(1), 31–58. doi:10.1007/s291-002-8199-x.

Van Essche, S., Ferrante, E., Turgut, A. E., van Lon, R. R. S., Holvoet, T., & Wenseleers, T. (2015). Environmental factors promoting the evolution of recruitment strategies in swarms of foraging robots. In *Proceedings of the First International Symposium on Swarm Behavior and Bio-Inspired Robotics* (pp. 1–8).

van Lon, R. R. (2017a). RinLog v3.2.2. `https://github.com/rinde/RinLog/tree/v3.2.2`. doi:10.5281/zenodo.571180.

van Lon, R. R. S. (2014a). RinLog: v1.0.0. `https://github.com/rinde/RinLog/tree/v1.0.0`. doi:10.5281/zenodo.13344.

van Lon, R. R. S. (2014b). RinSim: v3.0.0. `https://github.com/rinde/RinSim/tree/v3.0.0`. doi:10.5281/zenodo.13343.

van Lon, R. R. S. (2015a). Code and results, PRIMA 2015. `https://github.com/rinde/vanLon15-PRIMA-code/tree/v1.0.0`. doi:10.5281/zenodo.27365.

van Lon, R. R. S. (2015b). Dynamism, urgency, and scale dataset. doi:10.5281/zenodo.27364.

van Lon, R. R. S. (2015c). PDPTW dataset generator: v1.0.0. `https://github.com/rinde/pdptw-dataset-generator/tree/v1.0.0`. doi:10.5281/zenodo.27362.

van Lon, R. R. S. (2015d). RinLog: v2.0.0. `https://github.com/rinde/RinLog/tree/v2.0.0`. doi:10.5281/zenodo.27361.

van Lon, R. R. S. (2015e). RinSim: v4.0.0. `https://github.com/rinde/RinSim/tree/v4.0.0`. doi:10.5281/zenodo.27360.

van Lon, R. R. S. (2016a). On dynamism and urgency - code. `https://github.com/rinde/vanLon16-EJOR-code`. doi:10.5281/zenodo.48219.

van Lon, R. R. S. (2016b). PDPTW dataset dataset: v1.1.0. `https://github.com/rinde/pdptw-dataset-generator/tree/v1.1.0`. doi:10.5281/zenodo.59259.

van Lon, R. R. S. (2016c). RinLog v3.2.0. https://github.com/rinde/RinLog/tree/v3.2.0. doi:10.5281/zenodo.192111.

van Lon, R. R. S. (2016d). RinSim v4.3.0. https://github.com/rinde/RinSim/tree/v4.3.0. doi:10.5281/zenodo.192106.

van Lon, R. R. S. (2017b). evo4mas v0.3.0. https://github.com/rinde/evo4mas/tree/v0.3.0. doi:10.5281/zenodo.248966.

van Lon, R. R. S. (2017c). Optimizing agents with genetic programming - An evaluation of hyper-heuristics in dynamic real- time logistics - code. https://github.com/rinde/vanLon17-GPEM-code/tree/v1.0.0. doi:10.5281/zenodo.260130.

van Lon, R. R. S. (2017d). Optimizing agents with genetic programming - An evaluation of hyper-heuristics in dynamic real-time logistics - datasets and results. doi:10.5281/zenodo.259774.

van Lon, R. R. S. (2017e). RinECJ v4.3.0. https://github.com/rinde/RinECJ/tree/v0.3.0. doi:10.5281/zenodo.259718.

van Lon, R. R. S. (2017f). When do agents outperform centralized algorithms? - A systematic empirical evaluation in logistics - code v1.1.0. https://github.com/rinde/vanLon17-JAAMAS-code. doi:10.5281/zenodo.576389.

van Lon, R. R. S. (2017g). When do agents outperform centralized algorithms? - A systematic empirical evaluation in logistics - datasets and results v1.1.0. doi:10.5281/zenodo.576345.

van Lon, R. R. S., Branke, J., & Holvoet, T. (2017). Optimizing agents with genetic programming: an evaluation of hyper-heuristics in dynamic real-time logistics. *Genetic Programming and Evolvable Machines*, (pp. 1–28). doi:10.1007/s10710-017-9300-5.

van Lon, R. R. S., Ferrante, E., Turgut, A. E., Wenseleers, T., Vanden Berghe, G., & Holvoet, T. (2016). Measures of dynamism and urgency in logistics. *European Journal of Operational Research*, 253(3), 614–624. doi:10.1016/j.ejor.2016.03.021.

van Lon, R. R. S., Ferrante, E., Turgut, A. E., Wenseleers, T., Vanden Berghe, G., & Holvoet, T. (August 2015). Measures for dynamism and urgency in logistics. In *CW Reports*, volume CW686. Department of Computer Science, KU Leuven.

van Lon, R. R. S. & Holvoet, T. (2012). RinSim: A simulator for collective adaptive systems in transportation and logistics. In *Proceedings of the 6th*

IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012) (pp. 231–232). Lyon, France. doi:10.1109/SASO.2012.41.

van Lon, R. R. S. & Holvoet, T. (2013). Evolved multi-agent systems and thorough evaluation are necessary for scalable logistics. In *2013 IEEE Workshop on Computational Intelligence In Production And Logistics Systems (CIPLS)* (pp. 48–53). doi:10.1109/CIPLS.2013.6595199.

van Lon, R. R. S. & Holvoet, T. (2015). Towards systematic evaluation of multi-agent systems in large scale and dynamic logistics. In Q. Chen, P. Torroni, S. Villata, J. Hsu, & A. Omicini (Eds.), *PRIMA 2015: Principles and Practice of Multi-Agent Systems: 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings* (pp. 248–264). Cham: Springer International Publishing. doi:10.1007/978-3-319-25524-8_16.

van Lon, R. R. S. & Holvoet, T. (2017). When do agents outperform centralized algorithms? A systematic empirical evaluation in logistics. *Autonomous Agents and Multi-Agent Systems*. Under review. The first submitted version is published as a technical report (van Lon & Holvoet, 2016).

van Lon, R. R. S. & Holvoet, T. (October 2016). When do agents outperform centralized algorithms? A systematic empirical evaluation in logistics. In *CW Reports*. Department of Computer Science, KU Leuven.

van Lon, R. R. S., Holvoet, T., Vanden Berghe, G., Wenseleers, T., & Branke, J. (2012). Evolutionary Synthesis of Multi-Agent Systems for Dynamic Dial-a-Ride Problems. In *GECCO Companion '12 Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion* (pp. 331–336). Philadelphia, USA. doi:10.1145/2330784.2330832.

van Lon, R. R. S., Wiggers, P., Rothkrantz, L., & Holvoet, T. (2011). Design of evolvable biologically inspired classifiers. In *Fifth International Workshop on Nature Inspired Cooperative Strategies for Optimization*.

Vonolfen, S., Beham, A., Kommenda, M., & Affenzeller, M. (2013). *Structural Synthesis of Dispatching Rules for Dynamic Dial-a-Ride Problems*, (pp. 276–283). Springer Berlin Heidelberg: Berlin, Heidelberg. doi:10.1007/978-3-642-53856-8_35.

Waibel, M., Keller, L., & Floreano, D. (2009). Genetic team composition and level of selection in the evolution of cooperation. *IEEE Transactions on Evolutionary Computation*, 13(3), 648–660. doi:10.1109/TEVC.2008.2011741.

Weiss, G. (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press.

Weyns, D., Boucké, N., & Holvoet, T. (2006). Gradient field-based task assignment in an agv transportation system. In *Proc. of 5th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)* (pp. 842–849). doi:10.1145/1160633.1160785.

Weyns, D., Boucké, N., Holvoet, T., & Demarsin, B. (2007). DynCNET: A protocol for dynamic task assignment in multiagent systems. *First International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2007*, (pp. 281–284). doi:10.1109/SASO.2007.20.

Weyns, D., Schelfthout, K., Holvoet, T., & Lefever, T. (2005). Decentralized control of e'gv transportation systems. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05 (pp. 67–74). New York, NY, USA: ACM. doi:10.1145/1082473.1082806.

Willkomm, D., Machiraju, S., Bolot, J., & Wolisz, A. (2009). Primary user behavior in cellular networks and implications for dynamic spectrum access. *Communications Magazine, IEEE*, 47(3), 88–95. doi:10.1109/MCOM.2009.4804392.

Wilson, N. H. M. & Colvin, N. J. (1977). *Computer control of the Rochester dial-a-ride system.* Massachusetts Institute of Technology, Center for Transportation Studies.

Wolpert, D. H. & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1), 67–82. doi:10.1109/4235.585893.

Wooldridge, M. (2002). *An introduction to multiagent systems.* Wiley.

Yang, J., Jaillet, P., & Mahmassani, H. (2004). Real-time multivehicle truckload pickup and delivery problems. *Transportation Science*, 38(2), 135–148. doi:10.1287/trsc.1030.0068.

Yang, S., Jiang, Y., & Nguyen, T. T. (2012). Metaheuristics for dynamic combinatorial optimization problems. *IMA Journal of Management Mathematics*, 24(4), 451. doi:10.1093/imaman/dps021.

Curriculum vitae

Rinde Roelf Sebastiaan van Lon was born in Delft, The Netherlands, on May 10th 1986. After receiving the HAVO high school degree in 2003, he obtained the propedeuse diploma in technical informatics at Hogeschool van Amsterdam in 2004. Later that year, he enrolled at Delft University of Technology (TU Delft) where he obtained a B.Sc. in computer science in 2008. During his master studies at the same university, Rinde got in touch with bio-inspired algorithms, which prompted him to follow courses about evolutionary computing and swarm intelligence at Vrije Universiteit Amsterdam. In 2010 he graduated at TU Delft and joined the imec-DistriNet research group at KU Leuven, Leuven, Belgium. In Leuven he was supervised by Prof. dr. Tom Holvoet where he did research on multi-agent systems, dynamic logistics, and bio-inspired algorithms.

List of publications

Articles in internationally reviewed scientific journals

van Lon, R. R. S., Ferrante, E., Turgut, A. E., Wenseleers, T., Vanden Berghe, G., & Holvoet, T. (2016). Measures of dynamism and urgency in logistics. *European Journal of Operational Research*, 253(3), 614–624. doi:10.1016/j.ejor.2016.03.021

van Lon, R. R. S. & Holvoet, T. (2017). When do agents outperform centralized algorithms? A systematic empirical evaluation in logistics. *Autonomous Agents and Multi-Agent Systems*. Under review. The first submitted version is published as a technical report (van Lon & Holvoet, 2016)

van Lon, R. R. S., Branke, J., & Holvoet, T. (2017). Optimizing agents with genetic programming: an evaluation of hyper-heuristics in dynamic real-time logistics. *Genetic Programming and Evolvable Machines*, (pp. 1–28). doi:10.1007/s10710-017-9300-5

Articles at international conferences and symposia, published in full proceedings

van den Haak, P., van Lon, R. R. S., van der Meer, J., & Rothkrantz, L. (2010). Stress assessment of car-drivers using eeg-analysis. In *Proceedings of the 11th International Conference on Computer Systems and Technologies* (pp. 473–477)

van Lon, R. R. S., Wiggers, P., Rothkrantz, L., & Holvoet, T. (2011). Design of evolvable biologically inspired classifiers. In *Fifth International Workshop on Nature Inspired Cooperative Strategies for Optimization*

Hanif, S., van Lon, R. R. S., Gui, N., & Holvoet, T. (2011). Delegate mas for large scale and dynamic pdp: A case study. In *Intelligent Distributed Computing*

V, volume 5 (pp. 23–33)

van Lon, R. R. S., Holvoet, T., Vanden Berghe, G., Wenseleers, T., & Branke, J. (2012). Evolutionary Synthesis of Multi-Agent Systems for Dynamic Dial-a-Ride Problems. In *GECCO Companion '12 Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion* (pp. 331–336). Philadelphia, USA. doi:10.1145/2330784.2330832

van Lon, R. R. S. & Holvoet, T. (2012). RinSim: A simulator for collective adaptive systems in transportation and logistics. In *Proceedings of the 6th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012)* (pp. 231–232). Lyon, France. doi:10.1109/SASO.2012.41

van Lon, R. R. S. & Holvoet, T. (2013). Evolved multi-agent systems and thorough evaluation are necessary for scalable logistics. In *2013 IEEE Workshop on Computational Intelligence In Production And Logistics Systems (CIPLS)* (pp. 48–53). doi:10.1109/CIPLS.2013.6595199

Merlevede, J., van Lon, R. R. S., & Holvoet, T. (2014). Neuroevolution of a multi-agent system for the dynamic pickup and delivery problem. In *International Joint Workshop on Optimisation in Multi-Agent Systems and Distributed Constraint Reasoning (OptMAS, co-located with AAMAS)*

van Lon, R. R. S. & Holvoet, T. (2015). Towards systematic evaluation of multi-agent systems in large scale and dynamic logistics. In Q. Chen, P. Torroni, S. Villata, J. Hsu, & A. Omicini (Eds.), *PRIMA 2015: Principles and Practice of Multi-Agent Systems: 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings* (pp. 248–264). Cham: Springer International Publishing. doi:10.1007/978-3-319-25524-8_16

Van Essche, S., Ferrante, E., Turgut, A. E., van Lon, R. R. S., Holvoet, T., & Wenseleers, T. (2015). Environmental factors promoting the evolution of recruitment strategies in swarms of foraging robots. In *Proceedings of the First International Symposium on Swarm Behavior and Bio-Inspired Robotics* (pp. 1–8)

Dinh, H. T., van Lon, R. R. S., & Holvoet, T. (2016). Multi-Agent Route Planning Using Delegate MAS. In *ICAPS Proceedings of the 4th Workshop on Distributed and Multi-Agent Planning (DMAP-2016)* (pp. 24–32)

Technical reports

van Lon, R. R. S., Ferrante, E., Turgut, A. E., Wenseleers, T., Vanden Berghe, G., & Holvoet, T. (August 2015). Measures for dynamism and urgency in logistics. In *CW Reports*, volume CW686. Department of Computer Science, KU Leuven

van Lon, R. R. S. & Holvoet, T. (October 2016). When do agents outperform centralized algorithms? A systematic empirical evaluation in logistics. In *CW Reports*. Department of Computer Science, KU Leuven

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
IMEC-DISTRINET
Celestijnenlaan 200A box 2402
B-3001 Leuven
first.name@dept.kuleuven.be
http://www.distrinet.cs.kuleuven.be