



CHALMERS

Algoritmer och simulation för en dynamiskt uppdaterbar ruttplanerare

En undersökning inom algoritmer för lösning av ett modifierat traveling salesman problem

Examensarbete inom högskoleingenjörsprogrammet Datateknik

OSKAR JAKOBSSON
JONATAN AXETORN

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK

CHALMERS TEKNISKA HÖGSKOLA OCH GÖTEBORGS UNIVERSITET

Göteborg, Sverige 2022

www.chalmers.se www.gu.se

EXAMENSARBETE 2022

Algoritmer och simulation för en dynamiskt uppdaterbar ruttplanerare

En undersökning inom algoritmer för lösning av ett modifierat
traveling salesman problem

OSKAR JAKOBSSON
JONATAN AXETORN



CHALMERS

Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2022

Algoritmer och simulation för en dynamiskt uppdaterbar ruttplanerare
En undersökning inom algoritmer för lösning av ett modifierat traveling salesman
problem
OSKAR JAKOBSSON
JONATAN AXETORN

© OSKAR JAKOBSSON, 2022.

© JONATAN AXETORN, 2022.

Handledare: Jonas Almström Duregård
Examinator: Lars Svensson, Institutionen för data- och informationsteknik

Examensarbete 2022
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet
SE-412 96 Göteborg
Telefon +46 31 772 1000

Skriven i L^AT_EX
Chalmers digitaltryck
Göteborg, Sverige 2022

Förord

Tack till Magnus Gustaver för LaTeX-mallen som användes under rapporten. Vi skulle även vilja rikta ett stort tack till Jonas Almström Duregård för all hjälp och feedback han har givit oss under projektets gång. Vi vill tacka Bzzt AB för chansen att skriva vårt exjobb hos dom. Vi vill också tacka Andreas Ekeroot för hans vägledning under detta projekt.

Oskar Jakobsson, Jonatan Axetorn, Göteborg, Maj 2022

Algoritmer och simulation för en dynamiskt uppdaterbar ruttplanerare
En undersökning inom algoritmer för lösning av ett modifierat traveling salesman problem

OSKAR JAKOBSSON

JONATAN AXETORN

Institutionen för Data- och Informationsteknik

Chalmers tekniska högskola

Göteborgs Universitet

Abstract

Bzzt AB is a logistics company that focuses on optimised routes for deliveries. A problem that occurs when building software to optimise routes is the traveling salesman problem (TSP). TSP can be applied to many areas including the vehicle routing problem (VRP). Both VRP and TSP are two well documented areas of research, and there are multiple variations of the problems. A study of industry-standard solutions was conducted before moving on to development. This report presents and compares three different solutions to a modified version of TSP, a brute-force solution, an euclidean approximation and a greedy approximation. Based on the results derived from testing the algorithms onto four different scenarios the conclusion is that the euclidean approximation consistently calculates suitable routes based on the criterias stated. To conduct this comparison a simulation- and testing environment has also been developed. This software is engineered to represent a real world scenario by generating events and simulating the passing of time. The simulation- and testing software also gathers the information from the calculations and condenses the data into a readable format. New algorithms can be tested through the simulation-software to analyse the efficiency based on stated criteria.

The report is written in Swedish.

Keywords: Vehicle routing problem, Traveling salesman problem, logistics, algorithms, simulation, software.

Innehåll

1	Inledning	1
1.1	Syfte	1
1.2	Mål	1
1.3	Avgränsningar	2
2	Teknisk bakgrund	3
2.1	Traveling salesman	3
2.2	Tidigare forskning	4
2.3	Open route service	4
2.4	Docker	5
2.5	Vanliga ord och begrepp	5
3	Metod	7
3.1	Efterforskning	7
3.2	Utveckling av testning- och simuleringsmiljö	7
3.3	Utveckling av algoritmer	8
3.4	Skapande av scenarion	8
3.5	Lokal installation av ORS	9
4	Konstruktion	11
4.1	Testning- och simuleringsmiljö	11
4.2	Gemensamma egenskaper	12
4.3	Algoritm I - TSP - BruteForce	13
4.4	Approximering	14
4.5	Algoritm II - Euklidisk approximation	15
4.6	Algoritm III - Girig approximation	16
5	Resultat	19
5.1	Scenario 1	19
5.2	Scenario 2	21
5.3	Scenario 3	23
5.4	Scenario 4	26
6	Diskussion	31
6.1	Prestanda och precision	31
6.2	Testning- och simuleringsmiljö	32
6.3	Testdata	33

6.4 Utvärdering av tidsplan	33
7 Slutsats och framtida forskning	35
A Appendix A	I
B Appendix B	V
C Appendix C	VII
D Appendix D	IX
E Appendix E	XIII
F Appendix F	XVII
G Appendix G	XIX
H Appendix H	XXI
I Appendix I	XXIII

1

Inledning

Under COVID-19-pandemin har matleveransföretag ökat sin omsättning eftersom fler människor väljer att beställa sina varor på nätet [1]. Eftersom leveranser hem till privatpersoner ökar är det viktigt att leveranserna är snabba och effektiva. Inte minst för att kunden ska vara nöjd men även för att förminska utsläpp då en effektiv rutt är en kortare rutt och därmed förminskas energiåtgången per leverans. Denna effektivitet uppnås genom att använda en väl optimerad rutt-algoritm. Företaget Bzzts affärsidé är att använda en sådan optimerad algoritm där en förare utöver matleverans kan leverera andra hembeställda paket samt bedriva taxitjänst. I kombination med detta kan föraren ta emot ett större antal beställningar åt gången. Bzzts fordon är utsläppsfria fordon som drivs av förnybar el vilket leder till mindre förorening [2]. Rapporten kommer presentera tre olika algoritmer som angriper effektiviseringen av rutt-algoritmer på olika sätt. Utöver algoritmerna utvecklades en simuleringsmiljö som kommer att nyttjas för att analysera resultat från exekvering. Resultatet kommer analyseras och diskuteras och därefter presenteras en slutsats kring vilken av algoritmerna som är bäst lämpad för uppgiften, samt en utvärdering av den utvecklade simuleringsmiljön.

1.1 Syfte

Syftet med detta projekt är att utveckla och utvärdera tre distinkta routing-algoritmer, samt utveckla en testning- och simuleringsmiljö för att jämföra olika algoritmer.

1.2 Mål

Målen under detta projektet är de följande:

- Utveckla samt jämföra tre olika routing-algoritmer och konkretisera skillnader och fördelar bland dessa.
- Utifrån resultaten sammanställa vilken av de tre algoritmerna som är mest effektiv över flera olika scenarion.
- Utveckla ett ramverk som ska agera som en simuleringsmiljö där det finns möjlighet att testa algoritmer mot olika scenarion. Ramverket ska kunna generera testdata, simulera tidsåtgång, samt sammanställa körningens resultat.

1.3 Avgränsningar

Algoritmerna kommer inte kunna samspela med Bzzts interna mjukvara, utan undersökningen kommer endast att utföras för att utveckla och jämföra olika routing-algoritmer. Upphämtning och avlämning av ordrar kommer att antas ske utan tidsåtgång. Laddningstider för fordon samt förmåga att ta sig till en laddningsstation kommer inte att tas i beaktning, endast att batteriet räcker för nuvarande körning.

Algoritmtid och simulationstid kommer hållas separerade i konstruktionen. Simuleringen kommer alltså inte ta i beaktning den tid som går åt för att avgöra vilken förare som ska tilldelas en order. Utifrån simuleringsmiljöns synvinkel så sker varje orderutdelning omedelbart.

Inga standardbibliotek för lösningar av Traveling Salesman kommer att nyttjas utan algoritmerna kommer att utvecklas från grunden. Den här rapporten kommer heller inte använda evolutionära algoritmer eller någon annan form av artificiell intelligens eller maskininlärnings-modeller. Rapporten kommer endast undersöka möjligheten att approximera fram godtagbara lösningar genom enkel heuristik och sekventiella jämförelser.

Slutprodukten ska inte anses vara marknads-färdig utan endast ett medel för att implementera routing-algoritmer och jämföra resultaten. Ingen beaktning kommer att tas emot vilken sorts hårdvara som algoritmen förväntas köra på.

2

Teknisk bakgrund

I detta kapitel presenteras den tekniska bakgrund som ligger bakom metod- samt konstruktionsdelen. Först presenteras handelsresandeproblemet, sedan presenteras tidigare forskning inom ämnet. Därefter förklaras vilket Vehicle Routing Problem som rapporten kommer att angripa, och sedan vilka API:er och teknologier som har nyttjats. Slutligen listas vanliga ord och begrepp som förekommer.

2.1 Traveling salesman

Resemannaproblemet eller Traveling salesman problem (TSP) är ett känt och välbeforskat problem inom datavetenskap [3]. Problemet uppstår när det utförs en genomsökning av ett antal noder för att hitta den snabbaste rutten mellan dessa. TSP är ett så kallat NP - complete problem. Detta betyder att det saknas en effektiv lösning då existerande lösningar baseras på icke-deterministiska algoritmer med polynomiell tidsåtgång [4].

TSP beskrivs som att en säljare vill besöka ett visst antal städer. Säljaren vill hålla kostnaden för resorna och avståndet mellan städerna till det lägsta möjliga. Problemet uppstår eftersom det finns ett mycket stort antal resor som måste jämföras för att bedömma vilken resa som är mest effektiv.

Den enklaste lösningen som garanterar korrekt resultat är en så kallad brute force-lösning [4]. I en sådan lösning jämförs alla möjliga fall. I exemplet med en säljare undersöker en brute force lösning alla möjliga resor mellan de olika städerna och väljer ut den kortaste. En sådan brute force lösning på TSP har en komplexitet på $O(n!)$, där n är antalet städer. Detta innebär att för ett scenario med 50 städer kommer det finnas 50! olika sätt att besöka dessa. Problemet med denna lösning är att komplexiteten blir som nämnts innan $O(n!)$ vilket resulterar i en mycket långsam algoritm även vid någorlunda låga värden på n . Mer sofistikerade lösningar använder sig utav heuristik för att approximera den bästa lösningen. Detta innebär att tiden det tar för att hitta en väg är kortare men lösningen kan skilja sig från den absolut kortaste vägen. I slutändan är det heuristik som föredras inom datavetenskap eftersom en $O(n!)$ komplexitet kan bli för långsam för att implementera. Mer intressant är därmed den mest effektiva lösningen och inte den absolut korrekta.

2.2 Tidigare forskning

Vehicle Routing Problem (VRP) är en generalisering av TSP och är ett välkänt problem inom datavetenskap där det finns ett brett utbud av tidigare forskning [5][6][7]. VRP syftar på att med en given fordonsflotta hitta den mest effektiva rutten för leveranser och andra fordonsrelaterade uppgifter [8]. Likt TSP kan slutmålet med en VRP-lösning variera. Det kan handla om att minimera bränsleåtgång, minimera restid, eller andra optimeringsmål. Det finns många olika variationer på VRP [8] och den variation som närmast representerar den här rapportens frågeställning är Dynamic Vehicle Routing Problem with Time Window (DVRPTW) [9]. Dynamic syftar här på att kundbehovet på ett eller annat sätt förväntas förändras under körning eller är okänt vid start. Time Window syftar på att det finns någon form av tidsbegränsning för leverans till kund.

Den här rapportens frågeställning skiljer sig från den traditionella DVRPTW-frågeställningen då alla faktorer bortsett från vilka förare som är tillgängliga är okända. Leveranserna utgår inte från en depå med all materiell tillgänglig utan varje order har en unik upphämningsplats samt en unik avlämningsplats som är okänd för algoritmen innan ordern inkommit. Det finns mindre forskning som fokuserar på den här ytterst dynamiska förändringen av frågeställningen. De flesta rapporter som försöker lösa ett DVRPTW problem eller ett VRP med liknande dynamisk karaktär vänder sig till heuristik och evolutionära algoritmer [9][10][11].

Ett annat sätt att närma sig VRP är att använda sig utav föregående körningar för att skapa en typ av kundrepresentation för att optimera framtida beräkningar [5]. Eftersom traditionella lösningar av VRP inte tar hänsyn till föregående exekveringar kommer nya körningar med hög sannolikhet utföra samma beräkningar. Därav argumenterar [5] för att en sådan lösning landar i en god precision och prestanda i många fall. Det förlitar sig på att beställningar sker utifrån samma utgångspunkt för att nå en förbättring. Denna lösning kan dock inte appliceras på VRP with time window.

2.3 Open route service

Open route service (ORS) är en open-source mjukvara som används för att lösa olika typer av routingproblem [12]. Rapporten kommer endast att nyttja ORS möjlighet att presentera vägbeskrivningar för olika fordon mellan två eller fler koordinater. Detta genomförs genom att användaren utför ett API-anrop med de koordinater rutten ska besöka i ordningen de ska besökas. En profil som motsvarar vilket fordon som ska användas skickas även med. Detta kan påverka vägvalen ORS gör då vissa fordon ej får befinna sig på vissa vägar. Det finns profiler för flera olika fordon, till exempel för cyklar, bilar och lastfordon. Rapporten kommer exklusivt att använda sig av profilen "driving-car" som representerar bil. ORS svarar ett API-anrop antingen i JSON eller GEOJSON format med information om den genererade rutten. I JSON-svaret hittar man bland annat information om ruttens sträcka, ruttens

tidsåtgång samt vilka koordinater rutten kommer att besöka.

2.4 Docker

Docker är en linux-baserad virtualiseringsmjukvara som används för att förenkla utvecklingen av egen mjukvara [13]. Docker fungerar likt en virtuell maskin då den arbetar med ett eget operativsystem (Linux ubuntu) som applikationen körs på för att upprätthålla konsistent beteende över olika plattformar. Däremot abstraherar en Docker-instans inte bort den fysiska hårdvaran på samma sätt som en virtual machine [14]. Docker abstraherar istället endast bort applikationslagret. Detta leder till att det krävs mindre minne för att köra en Docker-instans och snabbare boot-up-tider. Detta uppnås genom att en docker-instans är strukturerad i olika paket, benämnda "containers". En utgivare av mjukvara kan då skapa en docker-fil där det inkluderas nödvändiga beroenden som krävs för att exekvera mjukvaran. En annan användare kan sedan ladda ner dockerfilen, skapa samt köra containern lokalt och mjukvaran kommer att fungera identiskt.

2.5 Vanliga ord och begrepp

Orders och Noder: Order refererar till en inkommande beställning. En order består av två geografiska punkter, så kallade noder. En source-nod (src eller s) där orderns innehåll ska hämtas upp, samt en destinations-nod (dest eller d) där orderns innehåll ska levereras. Exempel: Det finns tre godtyckliga orders: order1, order2 och order3. Dessa tre orders kan delas upp i 6 olika noder: s1, d1, s2, d2, s3, d3. Bokstaven representerar source- eller destinations-nod och siffran representerar vilken order noden tillhör.

Förare: Förare refererar både till den person som utför leveransen samt personens fordon. När en förare benämns förflytta sig representeras objektet som ett fordon som berörs av dem restriktionerna som tillkommer. Detta kan exempelvis vara batteriliv eller fordonets möjlighet att leverera varor som kräver temperaturreglering.

Väg: En väg är en rad source och destinations-noder som utgör vägen en förare ska ta för att leverera alla ordrar som är tilldelade den föraren.

Prestanda och Precision: När resultat för algoritmerna presenteras kommer det analyseras ur två perspektiv: prestanda och precision.

- God prestanda syftar på så korta algoritmtider som möjligt. För att en algoritm ska anses ha duglig prestanda bör algoritmen ej överskrida ~ 1 sekund algoritmtid per order. Prestandan varierar beroende på exekveringsplattform.
- God precision syftar på så kort leveranstid som möjligt.

3

Metod

I detta kapitlet kommer rapporten att presentera projektets arbetsgång. Projektet inleddes med en period av efterforskning kring VRP och relaterad teori för att sedan röra sig emot utveckling. Först utvecklades testnings- och simuleringsmiljön, sedan lades fokus på att utveckla ruttplanerar-algoritmerna. Efter det skapades teoretiska scenarion för att kunna testa algoritmerna. Slutligen gjordes ett förbättringsarbete med en lokal installation av ORS-API:t.

3.1 Efterforskning

Projektet inleddes med akademisk efterforskning. Som tidigare nämnt så är VRP ett populärt problem inom både matematik och datavetenskap. Därav fanns det även ett stort utbud av akademiska rapporter. Många av dessa rapporter behandlar dock inte den ytterst dynamiska naturen av denna rapportens problemformulering. Utöver detta angriper många rapporter VRP med hjälp av artificiell intelligens och maskininlärning [10][11]. Detta går emot rapportens mål och därav användes inte den metodiken som många andra rapporter anammar. Däremot togs inspiration när det kom till heuristik från flertalet artiklar, exempelvis [9][10]. Heuristik syftar på en metod eller exekvering som baserat på teorier eller andra obeprövade metoder används för att ge icke-exakta, men tillräckliga svar på frågeställningar [15]. Heuristiska metoder och heuristiska urval implementerades i algoritmerna för att förbättra prestanda.

Eftersom projektet ville simulera scenarion baserat på riktiga koordinater krävdes en algoritm som kunde avläsa kartor och hitta en giltig väg mellan en eller flera koordinater. Projektet valde att nyttja ett API för detta och det utfördes efterforskning kring vilket API som skulle passa arbetet bäst. Toppkandidater i detta fallet blev ORS och Googles Google Maps API. På grund av möjlighet till lokal installation samt att det är ett gratis open source API valdes ORS före Google Maps API:t.

3.2 Utveckling av testning- och simuleringsmiljö

För att jämföra de olika algoritmernas effektivitet utvecklades det en testning- och simuleringsmiljö. Denna miljö hade som ansvar att utföra tre huvudsakliga uppgifter:

1. Generera testdata i syfte att mata in i simulationen under exekveringen av programmet. För detta krävs det först en initiering av scenariot som algoritmen

ska appliceras på. Därefter skapandet av olika händelser som ska hanteras under simulationens gång.

2. Simulera en tidsåtgång som ska representera ett verkligt scenario.
3. I slutet av varje körning generera och organisera resultatet i koncist och lättläst format, samt överföra detta till en resultatfil som sedan kan användas för att jämföra resultat från olika algoritmer och körningar.

Utöver detta ska miljön kunna modifieras. En ny algoritm ska relativt enkelt kunna utnyttja denna simuleringsmiljö.

3.3 Utveckling av algoritmer

Efter att testning- och simuleringsmiljön utvecklats påbörjades utvecklingen av algoritmerna. Algoritmernas uppgift är att för varje order dela ut ordern till den förare som har lägst total leveranstid. Detta inkluderar alltså både den inkommande ordern, samt tidigare utdelade ordrar. Denna beräkning sker för varje inkommande order. Varje algoritm ska ha en distinkt egenskap som skiljer den från de andra algoritmerna. Utifrån dessa krav skapades en grund för hur algoritmerna skulle fungera:

- **TSP - BruteForce:** Algoritmen jämför leveranstiden för alla möjliga permutationer av vägar för varje förare och väljer den snabbaste utav dessa. Detta kommer troligtvis innebära en hög precision men också en hög prestandakostnad.
- **Euklidisk approximation:** Algoritmen jämför endast de fem förare som är närmast src-noden enligt den euklidiska sträckan. Den jämför varje möjlig permutation per förare men istället för att exekvera API anrop för alla permutationer räknas det euklidiska avståndet ut vilket kräver betydligt mindre prestanda.
- **Girig approximation:** Algoritmen jämför precis som euklidisk approximation de fem förare som är närmast src-noden. Därefter väljer den väg genom att fråga ORS vilken av de giltiga noderna som är närmast. Detta pågår tills algoritmen har byggt upp en giltig väg. Teorin är att girig approximation ska både god precision och god prestanda.

Utifrån detta utvecklades algoritmerna. I kapitel 4 förklaras hur den färdigutvecklade mjukvaran fungerar.

3.4 Skapande av scenarion

För att testa algoritmerna krävdes det verklighetstrogen testdata i form av antal förare och deras egenskaper, samt ordrar som kunde matas in vid givna tidsintervaller. Utifrån dessa förutsättningar skapades fyra olika scenarion:

- Scenario 1: Scenario 1 genererar 30 olika ordrar över 17,5 minuter och har 10 förare tillgängliga. Dessa förare har olika förutsättningar i form av batteri vid simulationsstart, kapacitet i bilen, samt startposition. Det finns 30 unika

restauranger och 30 unika kundpositioner genom hela scenariot. Varje order har en max leveranstid på 30 minuter. Komplette scenariokod finns i appendix A.

- Scenario 2: Scenario 2 genererar 5 ordrar och 1 förare. Den första ordern som utfärdas har en längre leveranssträcka än resterande ordrar. Varje order har en max leveranstid på 60 minuter. Komplette scenariokod finns i appendix B.
- Scenario 3: Scenario 3 genererar 13 olika ordrar, där flertalet ordrar utgår från samma restaurang men vid olika tidpunkter. Scenariot har 4 förare. Varje order har en max leveranstid på 25 minuter. Komplette scenariokod finns i appendix C.
- Scenario 4: Scenario 4 är en exakt kopia av scenario 1 förutom att max leveranstid för orderarna har ändrats från 30 minuter till 20 minuter. Komplette scenariokod finns i appendix D.

3.5 Lokal installation av ORS

ORS-api:t finns tillgängligt att anropa gratis över internet men med vissa begränsningar [16]. Det går endast att göra 2000 “Directions” API-anrop per 24 timmar, samt 40 “Directions” API-anrop per minut. Utöver dessa begränsningar så finns det även en risk att API-anropen tar olika lång tid på sig då informationen måste skickas fram och tillbaka över internet. Detta kan i sin tur leda till opålitlig testdata. En lokal installation ger även bättre prestanda då ett API-anrop till en localhost går betydligt snabbare än ett API-anrop över internet. Därför togs beslutet att installera ORS-biblioteket lokalt genom att nyttja Docker. Med en lokal installation finns det inte längre några begränsningar på hur många anrop man får göra, vare sig per minut eller per 24 timmar. En lokal installation eliminerar även slump-faktorn som medkommer när man gör anrop över internet.

För att installera ORS lokalt skapades en Docker-container genom att följa ORS egna guide [17]. Efter installation hostades API:t lokalt på localhost port 8080 och algoritmen kunde göra API-anropen för vägbeskrivningar lokalt.

4

Konstruktion

I detta kapitel kommer rapporten att introducera samt förklara konstruktionen av den mjukvara som har utvecklats. Först presenteras den testning- och simuleringsmiljö som används för att kunna testa samt utvärdera algoritmerna. Sedan förklarar kapitlet de tre algoritmer som har utvecklats. Då presenteras först de gemensamma filosofierna och heuristiska urvalen som algoritmerna delar, för att sedan förklara varje enskild algoritms konstruktion.

4.1 Testning- och simuleringsmiljö

Testning- och simuleringsmiljön har som rapporten tidigare nämnt tre huvudsakliga uppgifter: Generera scenariot, simulera tidsåtgång, samt sammanställa resultatet av körningen i en fil. Framöver i kapitlet kommer rapporten referera till testning- och simuleringsmiljön som motor.

Generera scenariot: När motorn ska initieras kräver den två inparametrar: vilken algoritm som ska testas samt vilket scenario som algoritmen ska testas emot. Motorn skapar sedan en lista av order-objekt och en lista av förar-objekt. Listorna fylls med olika ordrar med tillhörande utdelningstider, samt förare med tillhörande startpositioner. En viktig aspekt av motorn är att det endast är just motorn som vet i vilken ordning ordrar ska delas ut och vid vilka tider de ska beställas. Algoritmerna kan alltså inte utforma en förarens vägval för att anpassa sig för eventuella framtida ordrar utan den informationen finns strikt tillgänglig i motorn.

Simulera tidsåtgång: Tidssimulationen fungerar som följande: När ett scenario startar hoppar motorn fram i tiden till den första händelsen, exempelvis den tidpunkt när den första ordern blir beställd. Därefter frågar motorn den algoritm som körs vilken förare som ska bli tilldelad ordern. När en order har blivit tilldelad jämförs tiden för två händelser:

1. Vid vilken tidpunkt kommer det beställas en order igen?
2. Vid vilken tidpunkt kommer en av de utdelade orderarna vara levererade?

Den händelse som är närmast i tid kommer att simuleras först. Om då exempelvis nästa beställning kommer att ske innan nästa leverans, kommer motorn hoppa fram i tiden till den beställningen. När motorn har hoppat fram kommer den även simulera körsträcka för alla tillgängliga förare. Förarna förflyttar sig baserat på den tidsåtgång mellan föregående förflyttning och nuvarande tidpunkt. När förarna har

förflyttat sig och befinner sig på deras uppdaterade positioner får algoritmen tillgång till förarobjekten samt inkommande order. Därefter delar algoritmen ut nästa order till en av förarna och simulationen jämför tidpunkten för nästa leverans med tidpunkten för nästa order ännu en gång. Detta fortsätter tills alla ordrar har blivit utdelade. När alla ordrar har blivit utdelade hoppar simulationen fram till tiden för nästa leverans för varje order tills dess att alla ordrar för scenariot har blivit levererade.

Det är även viktigt att poängtera att den tid som algoritmen tar för att dela ut en order ej kommer påverka motorns nuvarande simulationstid. Det är alltså två skilda tider och ur motorns synvinkel sker varje orderutdelning omedelbart.

Sammanställa resultatet: Efter att scenariot har körts igenom och hanterats sammanställs den väsentliga datan i ett excel dokument. Där varje order representerar en rad, och raderna kommer i ordning av ordrar, se exempelvis appendix E. Ur Excel-dokumentet utläses data som när en order kom in, vilken förare som ansvarade för ordern, när ordern hämtades upp, samt när den levererades. Excel-dokumentet visar även hur lång tid det tog för algoritmen att dela ut varje order. Utifrån denna datan kan även grafer skapas för att visuellt representera resultaten från en körning.

4.2 Gemensamma egenskaper

Då alla dessa tre algoritmer försöker uppnå samma mål finns det vissa likheter mellan dem. I detta delkapitel presenteras alla de gemensamma egenskaper och filosofier som algoritmerna utgår ifrån.

För varje inkommande order finns det en given mängd av tillgängliga förare. Varje algoritm kommer iterera över alla förare för att sortera ut, enligt olika parametrar, den mängd av förare som anses vara lämpliga att jämföra. De förare som anses vara lämpliga refereras till som förarkandidater. De krav som alla algoritmer har för att en förare ska vara en godkänd kandidat är:

- Föraren har tillräckligt med plats för att lasta in den nuvarande ordern.
- Föraren har möjlighet att behålla orderns temperatur.

Algoritmen kommer sedan iterera över varje förarkandidat och kolla: Om den här föraren får ordern, hur lång tid tar då den förarens totala väg? När en förarens totala tidsåtgång är uträknad utförs det en batterikontroll för att försäkra att föraren har tillräckligt med batteri för att köra hela den nya vägen. En förarkandidat som inte har tillräckligt med batteri kan ej bli tilldelad ordern. Om föraren har tillräckligt med batteri för att slutföra körningen kontrollerar algoritmen slutligen om någon av orderna kommer att överstiga deras tillåtna leveranstid för den genererade vägen. Om så är fallet kommer föraren inte längre vara en kandidat för ordern.

Den förarkandidat som klarar dessa kontroller samt har kortast totala tidsåtgång blir tilldelad ordern. Om ingen förarkandidat klarar av batteri- och leveranstidskontrollen blir ordern ej utdelad.

Gemensamt urval

Input: allDrivers: alla tillgängliga förare**Output:** driverCandidates: alla giltiga förarkandidater

```

1: for each driver in allDrivers do
2:   if driver.cantRegulateTemp then
3:     continue
4:   end if
5:   if order.capacityNeeded > driver.capacityLeft then
6:     continue
7:   end if
8:   driverCandidates.add(driver)
9: end for

```

Den största skillnaden mellan de tre algoritmerna är på vilket sätt de beräknar den totala vägen för en given order och förare. Det är alltså hur de löser TSP delen av VRP som skiljer dem åt.

4.3 Algoritm I - TSP - BruteForce

När Algoritm I - TSP BruteForce når punkten i sin exekvering där den totala tiden för leverans ska beräknas sker ett antal beräkningar. Först sammanställs alla möjliga kombinationer av noder utifrån vilka orders som redan är tilldelade en förare.

getAllRoutes(r, nodesNotInRoute), rekursiv metod för att generera alla permutationer

Input: r : en väg, vid första anrop en tom lista**Input:** nodesNotInroute = alla noder som ska besökas**Output:** allRoutes : Alla giltiga vägar

```

1: if !nodesNotInRoute.isEmpty then
2:   for int i = 0; i < nodesNotInRoute.size; i++ do
3:     justRemoved = nodesNotInRoute.remove(0)
4:     newRoute = r.clone()
5:     newRoute.add(justRemoved)
6:     getAllRoutes(newRoute, nodesNotInRoute)
7:     nodesNotInRoute.add(justRemoved)
8:   end for
9: else
10:  if !allRoutes.contains(r) then
11:    allRoutes.add(r)
12:  end if
13: end if

```

En sådan lista kan se ut som följande: [1, 0, 1, 0] där 1 och 0 är orderID för respektive order. ID 1 och 0 uppenbarar sig vid 2 tillfällen i listan eftersom den första siffran

representerar orderns source-nod och den andra representerar orderns destination-nod. En restriktion som algoritmen behandlar är att en väg inte kan innebära att en förare besöker en destinations-nod förrän den tillhörande source-noden är besökt. I ett verkligt scenario kan inte en förare anlända på sin destination utan att ha hämtat upp vad det är som ska levereras.

I en traditionell TSP finns det $n!$ olika sätt att traversera en graf [4]. Med rapportens restriktiva version utav TSP kommer ett antal av dessa permutationer ej vara giltiga. För varje uppsättning av n antal noder existerar det $\frac{n}{2}$ antal par av src- och destinations-noder. Varje enskilt par är korrekt ordnat i hälften av permutationerna, alltså att src-noden besöks innan destinations-noden. För att vara en giltig väg måste alla par vara korrekt ordnade. Detta innebär att för varje tillkommet par halveras andelen vägar som är giltiga utifrån en vanlig TSP. Alltså $\frac{n!}{2^{n/2}}$, där n är antalet noder, $\frac{n}{2}$ representerar då antal par och $n!$ är antalet permutationer i en traditionell TSP.

När alla listor är skapade beräknas leveranstiden för varje kombination av ordrar. Alla resultat jämförs med varandra och kombinationen med kortast leveranstid returneras. TSP - Brute Force använder sig av denna metod för att utläsa vilken förare är mest lämpad att hantera en inkommande order då alla förare undersöks med den inkommande ordern för att beräkna vilken förare som uppnår bäst precision. Föraren med bäst precision blir sedan tilldelad ordern och kör den kombination av src och dst-noder som har lägst leveranstid.

4.4 Approximering

Att beräkna den bästa möjliga sträckan för alla dessa förare kräver mycket datorkraft och exekveringen kommer ta lång tid. Då TSP - BruteForce jämför varje giltig permutation av noder så blir tidskomplexiteten för varje enskild förare $O((n!/2^{n/2}) \cdot O(ORS(n)))$, där n är antalet noder och $O(ORS(n))$ representerar tidskomplexiteten för ett anrop till ORS-api:t. Totala tidskomplexiteten för en TSP - BruteForce körning blir alltså: $O(M \cdot ((n!/2^{n/2}) \cdot O(ORS(n))))$, där M representerar antal förare. Algoritmtiden kommer då öka exponentiellt för större värden på n och riskerar dålig prestanda. Ett sätt att förminska tidskomplexiteten är att förminska antalet förarkandidater. De två nästkommande algoritmerna gör båda detta på samma vis, genom att välja ut de fem förarkandidater som har det kortaste euklidiska avståndet mellan förarens nuvarande position och orderns src-nod.

Det tillkommer även en extra heuristisk batterikoll för att försöka undvika att slösa datorkraft på en förare som eventuellt inte har tillräckligt med batteri för att genomföra en order. Först beräknas det euklidiska avståndet mellan förarens position och orderns src-nod. Sedan multipliceras avståndet med två för att kompensera för skillnader mellan den verkliga vägen och den euklidiska. Sedan divideras denna sträcka med 14, som är fordonens snitthastighet i m/s. Slutligen multipliceras kvoten med 0,0138 som är den batterikonstant som dras för varje sekund en förare är i körning. Pseudokod för urval av förarkandidater:

Gemensamt urval vid approximering

Input: allDrivers: alla tillgängliga förare**Output:** driverCandidates: alla giltiga förarkandidater

```

1: for each driver in allDrivers do
2:   if driver.cantRegulateTemp then
3:     continue
4:   end if
5:   if order.capacityNeeded > driver.capacityLeft then
6:     continue
7:   end if
8:   distance = euclidianDistance(driverPos, order.src)
9:   if distance*2/14 * 0.0138 > driver.getBattery then
10:    continue
11:  end if
12:  if driverCandidates.size < 5 then
13:    driverCandidates.add(driver)
14:  else
15:    if distance < longestDistance(driverCandidates) then
16:      drivareCandidates.remove(longestDistanceDriver)
17:      driverCandidates.add(driver)
18:    end if
19:  end if
20: end for

```

4.5 Algoritm II - Euklidisk approximation

Efter det heuristiska urvalet börjar Euklidisk approximation precis som TSP - Brute-Force med att generera alla möjliga permutationer av de tillgängliga noderna. För att spara tid gör inte Euklidisk approximation ett API-anrop per permutation utan räknar istället ut det euklidiska avståndet mellan noderna. De tre permutationer som har kortast euklidiskt avstånd används för att göra 3 API-anrop till ORS. Den permutation som har minst tidsåtgång enligt ORS blir då förarens valda väg, och tidsåtgången den tid som returneras för att jämföras med de andra 5 förarkandidaterna.

Pseudokod för Algoritm II - Euklidisk approximation

Algorithm II: Euklidisk approximation

```
1: allPermutations = generateAllPermutations(allNodes)
2: for each permutation in allPermutations do
3:   distance = euclidianDistance(permutation)
4:   if bestPermutations.size < 3 then
5:     bestPermutations.add(permutation)
6:   else
7:     if distance < longestDistance(bestPermutations) then
8:       bestPermutations.remove(longestDistance(bestPermutations))
9:       bestPermutations.add(permutation)
10:    end if
11:  end if
12: end for
13: bestTime = 0
14: for each permutation in bestPermutations do
15:   time = ors.call(permutation)
16:   if time < bestTime OR bestTime == 0 then
17:     bestTime = time
18:     bestPermutation = permutation
19:   end if
20: end for
21: if !driver.hasEnoughBattery() OR !allOrdersMeetTimeLimit() then
22:   return -1
23: end if
24: return bestTime
```

4.6 Algorithm III - Girig approximation

Algorithm III räknar ut förarens väg samt tidsåtgång på 2 olika sätt beroende på hur många hittills tilldelade orders föraren redan har.

1. Exakt uträkning, om föraren inte har någon tilldelad order räknas sträckans tid, alltså nuvarande position -> src -> dest, direkt ut genom ett API-anrop till ORS. Om föraren redan har en tilldelad order som inte har plockats upp finns det endast 4 olika noder att besöka. Detta resulterar i 6 olika permutationer ($\frac{4!}{2^{4/2}} = 6$). Algoritmen anropar då ORS för alla de sex möjliga permutationerna för att se vilken som har kortast restid.
2. Närmaste nod, om föraren redan har 2 eller mer tilldelade orders kommer algoritmen konsekvent välja den nod som är närmast tills alla noder är besökta för att bygga upp en giltig väg.

Pseudokod för Algorithm III:

Algorithm III: Girig approximation

Input: CurrentPos : förarens nuvarande position**Input:** numberOfOrders : Antal ordrar föraren har innan nuvarande ordern blir tilldelad**Input:** Order : Den beställda ordern

```
1: if numberOfOrders == 0 then
2:   path = currentPos->order.src->order.dest
3: else if numberOfOrders == 1 then
4:   allPermutations = findAllPermutations(s1,s2,d1,d2)
5:   path = findQuickest(allPermutations)
6: else if numberOfOrders == 2 then
7:   options : Alla giltiga noder som föraren kan besöka
8:   currentNode = currentPos
9:   while options > 0 do
10:    nextNode = findClosest(currentNode, options)
11:    options.remove(nextNode)
12:    if nextNode == a src node then
13:      options.add(correspondingDest(nextNode))
14:    end if
15:    path.add(nextNode)
16:    currentNode = nextNode
17:   end while
18: end if
19: if !driver.hasEnoughBattery() OR !allOrdersMeetTimeLimit() then
20:   return -1
21: end if
22: return (timeOf(path))
```

5

Resultat

I detta kapitel presenteras resultatet från de fyra scenarion rapporten tidigare har introducerat. Varje delkapitel kommer att presentera resultat från ett specifikt scenario, där algoritm-tid och leveranstid är de två viktigaste faktorerna. För varje scenario kommer den mest intressanta datan att presenteras, men all genererad rådata finns tillgängligt i tillhörande Appendix. Dessa refereras till i varje delkapitel.

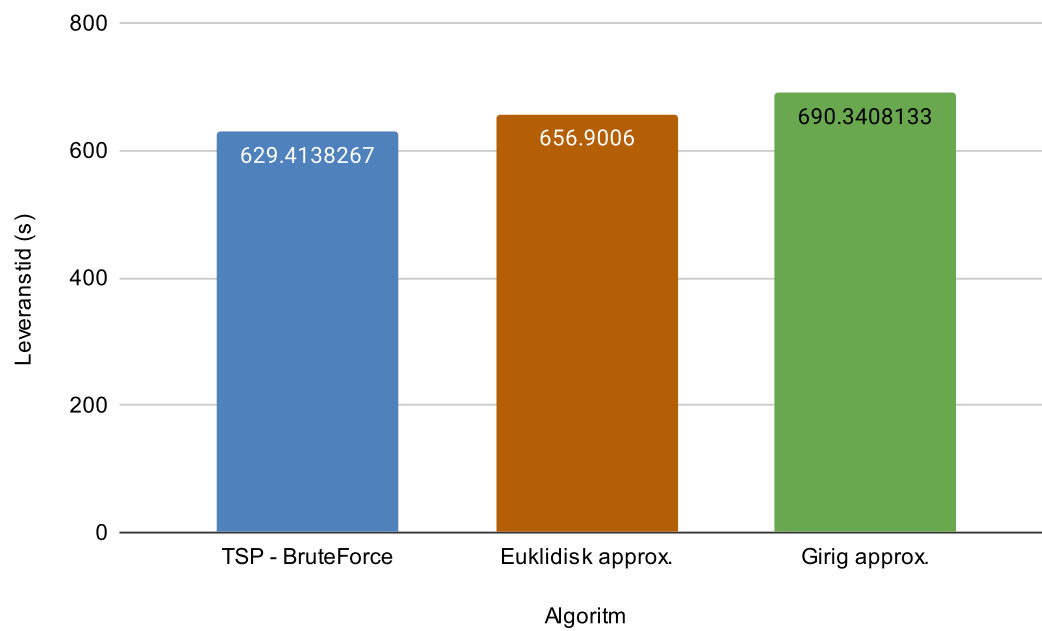
5.1 Scenario 1

All rådata för scenario 1 finns tillgänglig i appendix E.

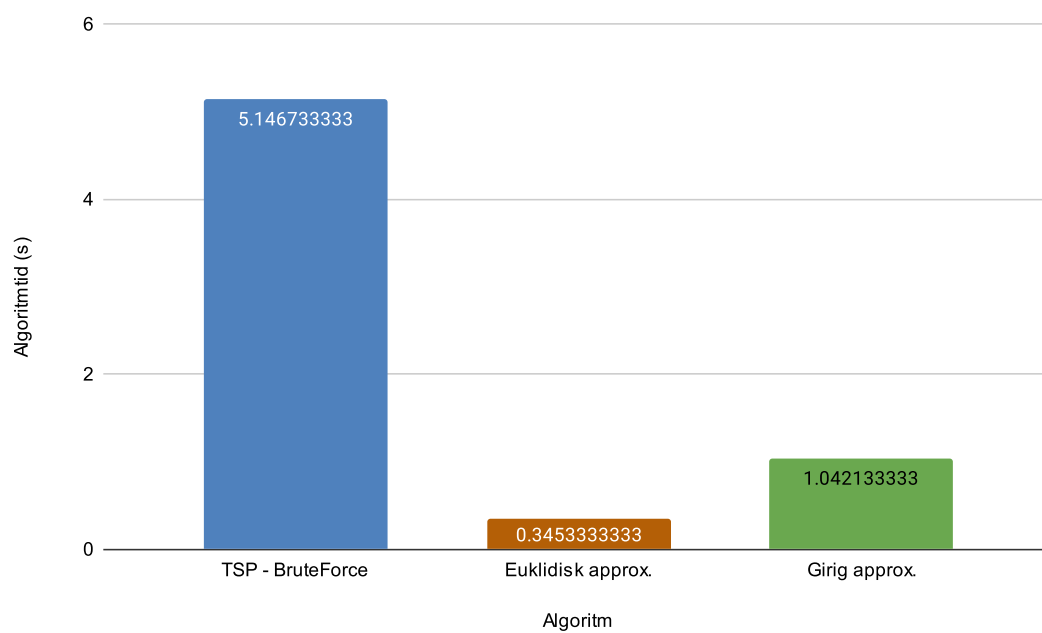
För scenario 1 så kan algoritmernas genomsnittliga precision rangordnas enligt resultaten uppvisade i figur 5.1.

1. TSP - BruteForce når bäst precision med en genomsnittlig leveranstid på ~ 629.4 sekunder.
2. Den euklidiska approximeringen har en precision på ~ 656.9 sekunder.
3. Den giriga approximeringen har en precision på ~ 690.34 sekunder

Prestandamässigt så presterar den euklidiska approximeringen bäst med ett genomsnitt på 0.35 sekunder algoritmtid per order, se figur 5.2. Den giriga approximeringen visar sig något långsammare och når en genomsnittlig algoritmtid per order på 1.0 sekunder. Långsammast är TSP BruteForce som uppnår ett genomsnitt på 5.1 sekunder algoritmtid per order.

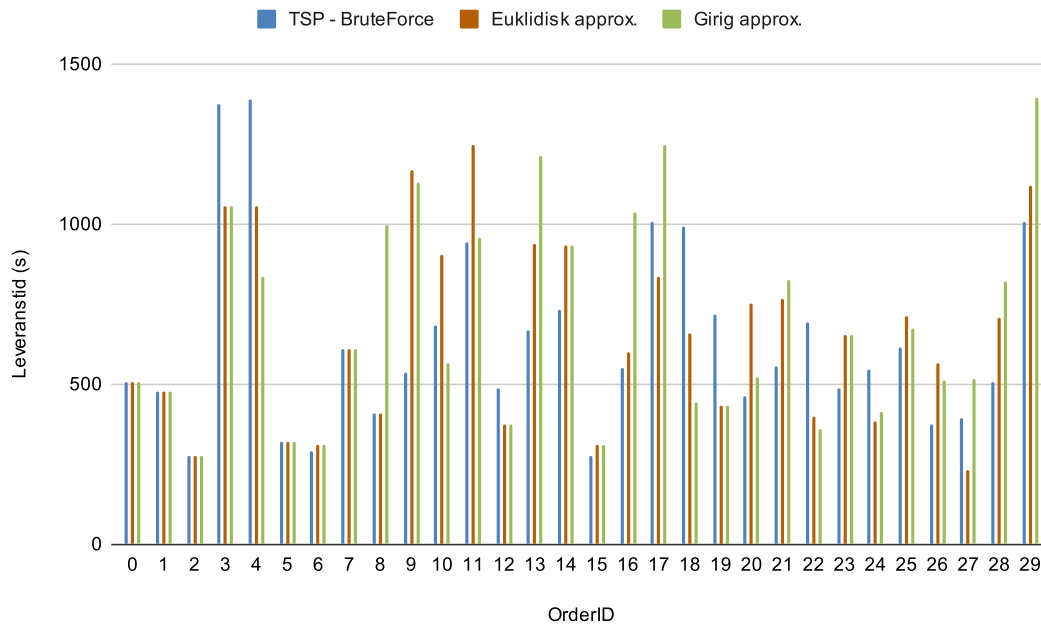


Figur 5.1: Genomsnittlig leveranstid scenario 1



Figur 5.2: Genomsnittlig algoritmtid scenario 1

I figur 5.3. presenteras hur leveranstiden är fördelad över de 30 olika orderarna för varje respektive algoritm. Ingen algoritm misslyckas att upprätthålla den tidsgräns på 30 minuter som varje order har i scenario 1. Som grafen indikerar har de tre algoritmerna ofta samma leveranstid för de första orderarna. TSP - BruteForce förskjuter order 3 och 4 men får därför en lägre genomsnittlig leveranstid då de mittersta orderarna har lägre leveranstid för TSP - BruteForce än de två approximeringarnas.

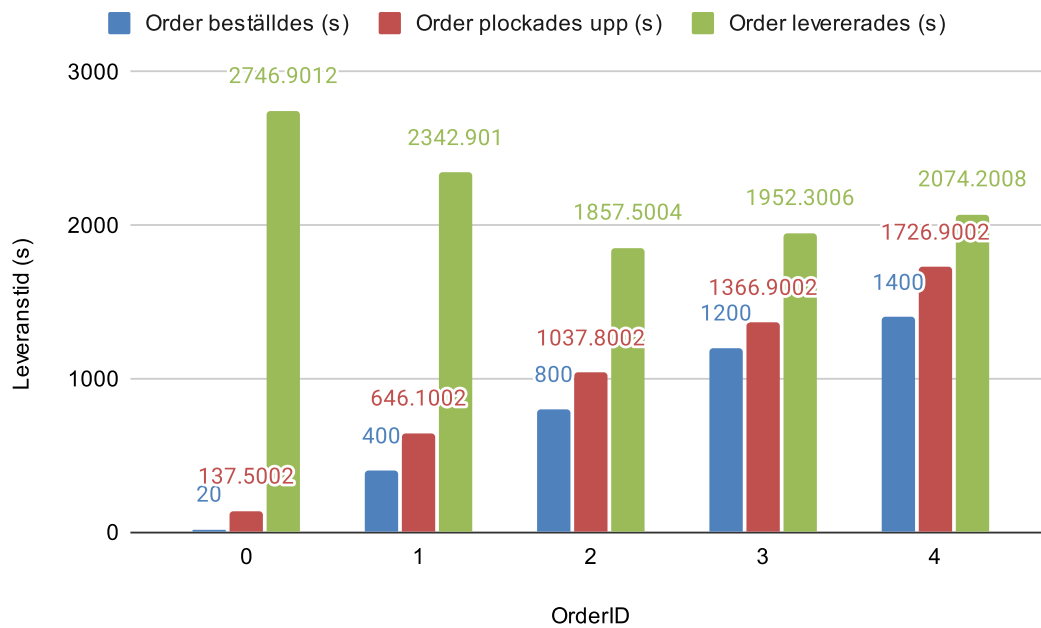


Figur 5.3: Leveranstid scenario 1

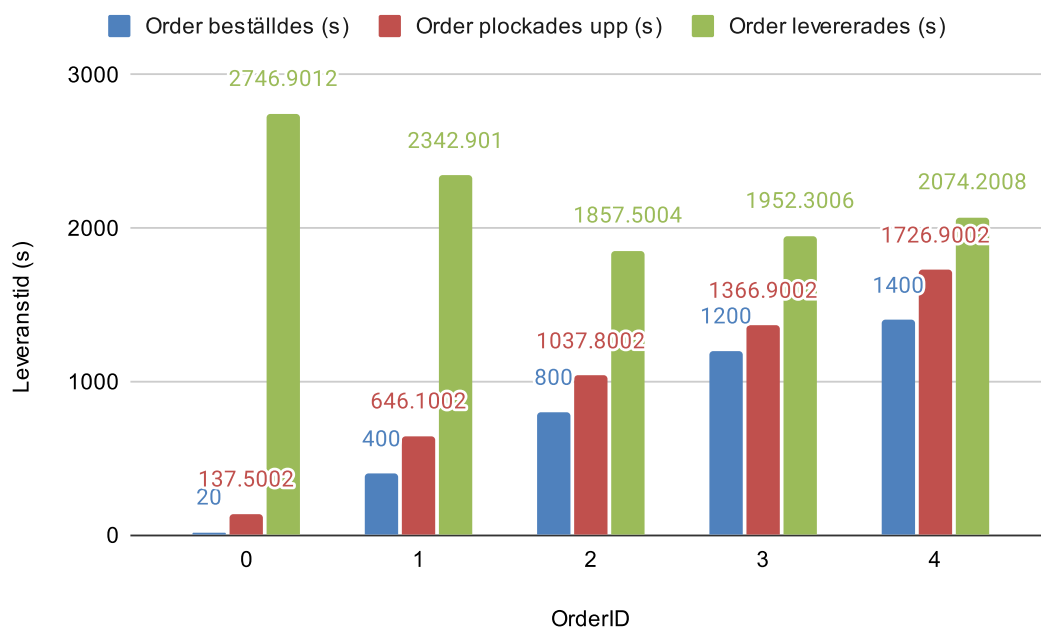
5.2 Scenario 2

All rådata för scenario 2 finns tillgänglig i appendix F.

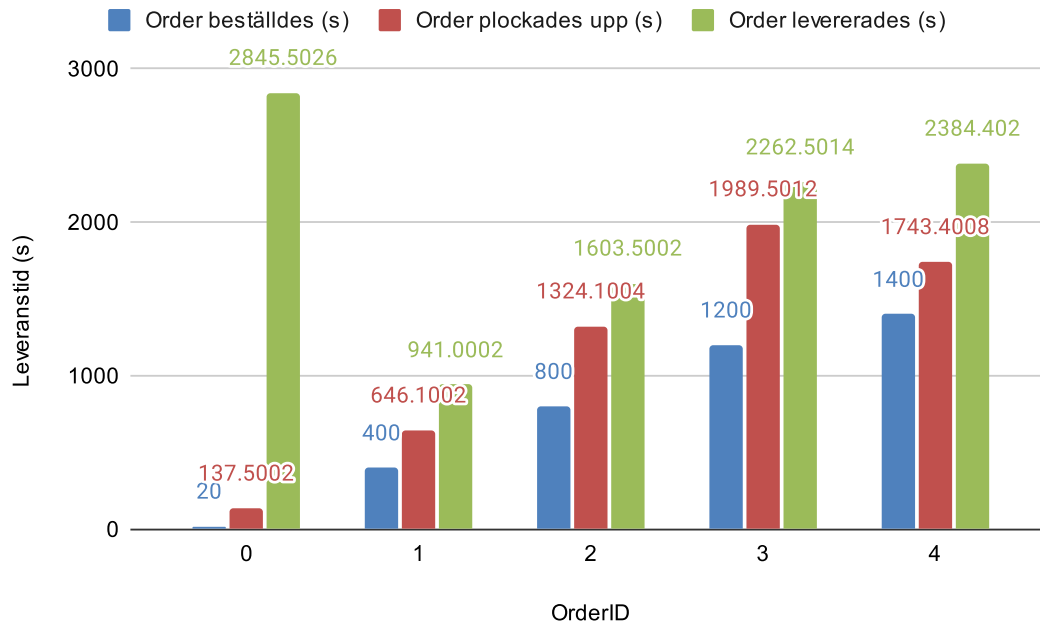
I följande grafer representeras exekvering av Scenario 2, se figur 5.4, 5.5 samt figur 5.6. Som graferna indikerar registreras orders vid tidpunkterna 20, 400, 800, 1200 och 1400. Tydligt är likheten mellan TSP-BruteForce och Euklidisk approximation. Siffran under följande grafer representerar orderns identifikationsnummer(ID).



Figur 5.4: TSP - BruteForce



Figur 5.5: Euklidisk Approximation



Figur 5.6: Girig Approximation

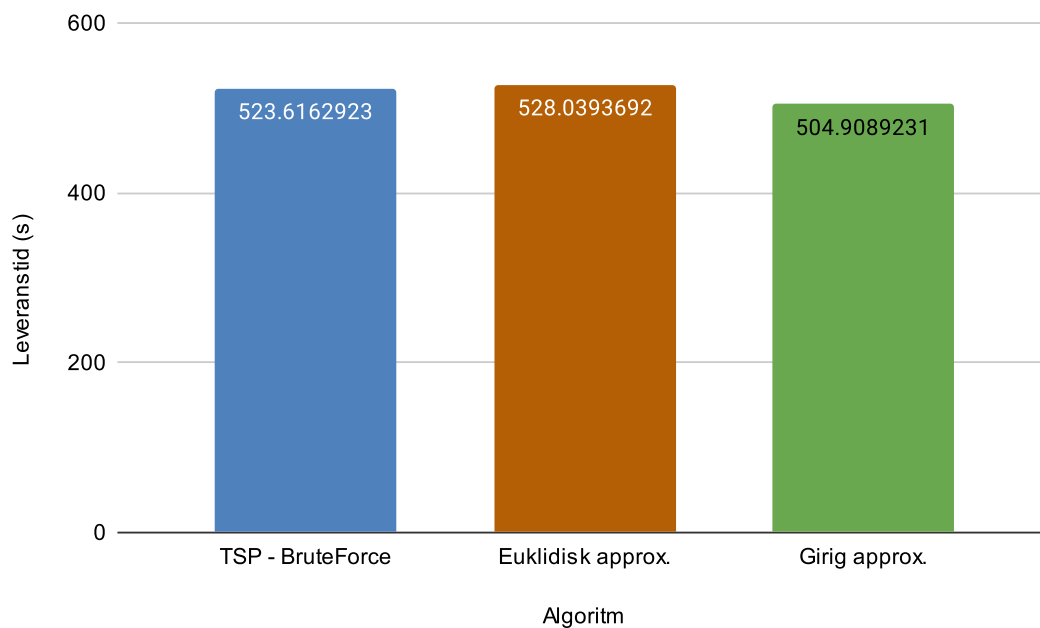
5.3 Scenario 3

All rådata för scenario 3 finns tillgänglig i appendix G.

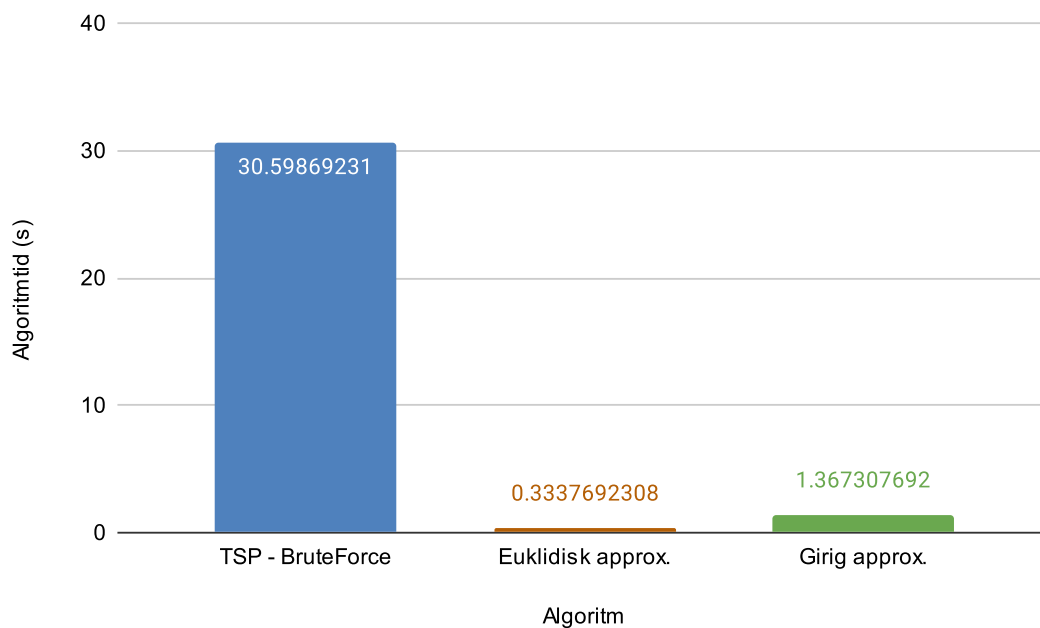
För scenario 3 så kan algoritmernas genomsnittliga precision rangordnas enligt resultaten uppvisade i figur 5.7.

1. Den giriga approximationen uppnår bäst precision med en genomsnittlig leveranstid på ~ 504.91 sekunder.
2. TSP - BruteForce uppnår en precision på ~ 523.6 sekunder.
3. Den euklidiska approximationen har en nästan jämlig precision på ~ 528.0 sekunder

Prestandamässigt så presterar den euklidiska approximationen bäst med ett genomsnitt på ~ 0.3 sekunder algoritmtid per order, se figur 5.2. Den giriga approximationen visar sig något långsammare och når en genomsnittlig algoritmtid per order på ~ 1.4 sekunder. Långsammast är TSP BruteForce som uppnår ett högt genomsnitt på ~ 30.6 sekunder algoritmtid per order.



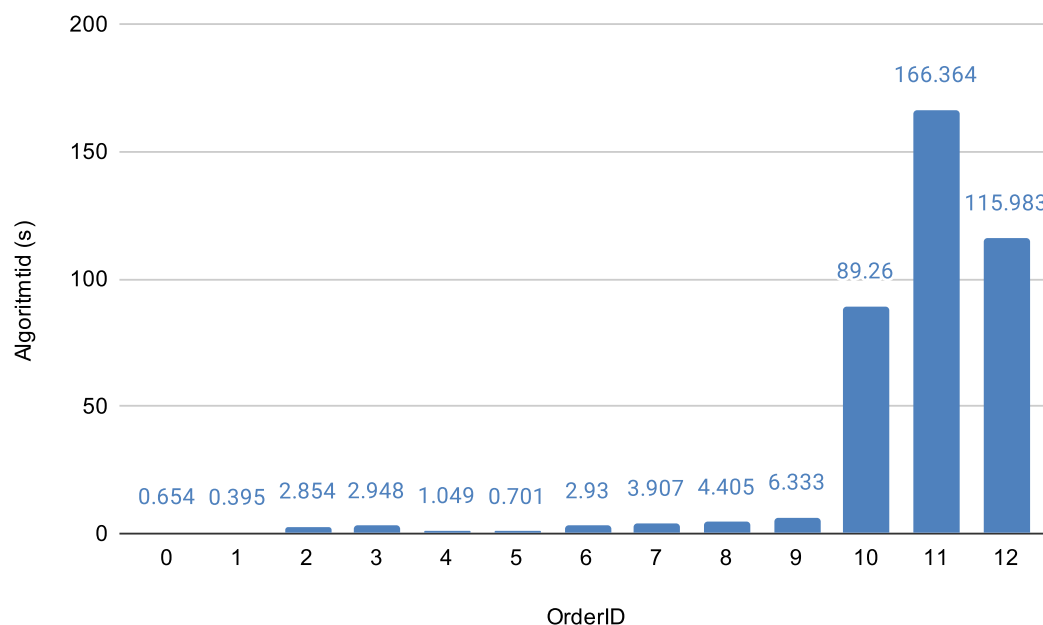
Figur 5.7: Genomsnittlig leveranstid scenario 3



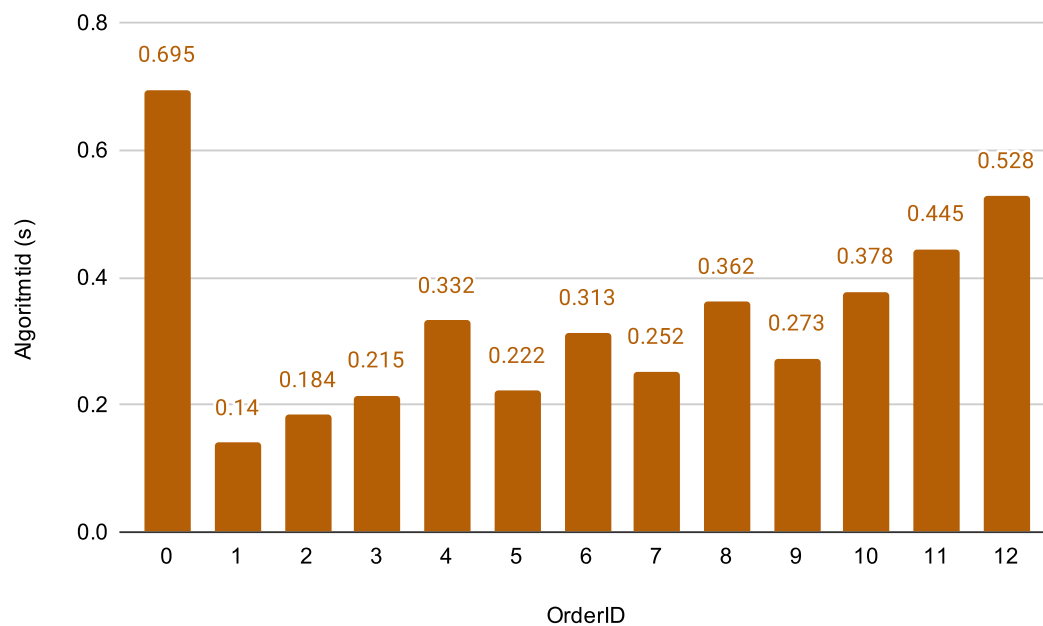
Figur 5.8: Genomsnittlig algoritmtid scenario 3

I figur 5.9 påvisas den exponentiella aspekten av algoritmtid för TSP - BruteForce. Efter order 9 försämras prestandan markant och algoritmtiden ökar från 6.3 sekunder till 89.2 sekunder. Den euklidiska approximeringen håller en jämnare algoritmtid för sina ordrar, se figur 5.10. Den euklidiska approximeringen ser också en ökning från order 9 och framåt, men denna ökning är betydligt mindre signifikant och

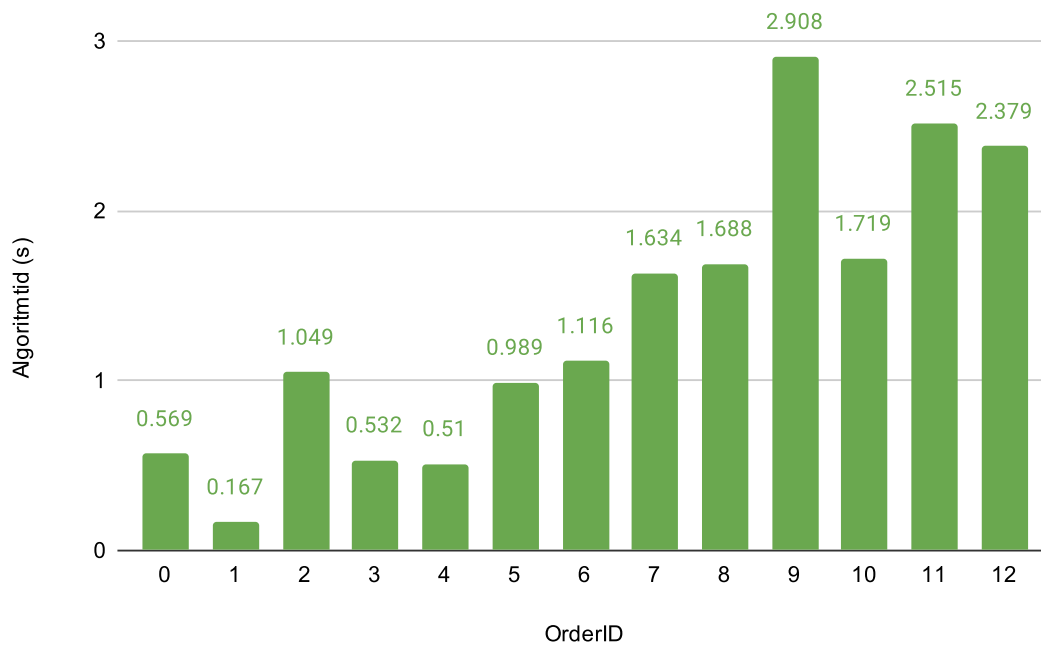
rör sig om en genomsnittlig ökning på ~ 0.1 sekunder per order. Även den giriga approximationen följer ett liknande mönster och förlorar prestanda mot de sista fyra orderarna i scenario 3, se figur 5.11.



Figur 5.9: Algoritmtid TSP BruteForce Scenario 3



Figur 5.10: Algoritmtid Euklidisk approximering Scenario 3



Figur 5.11: Algoritmtid Girig approximering Scenario 3

5.4 Scenario 4

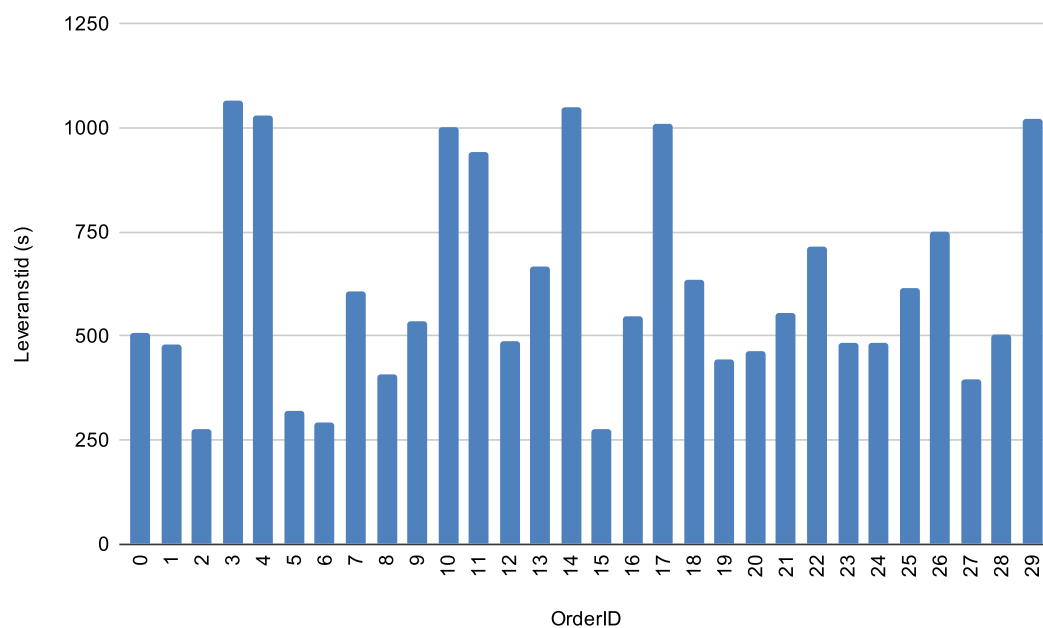
All rådata för scenario 4 finns tillgänglig i appendix H.

Scenario 4 är samma scenario som scenario 1 men med en nedsatt maxtid för leverans från 30 minuter till 20 minuter. I figur 5.12, 5.13 samt figur 5.14 kan vi se att alla tre algoritmer klarar av att optimera rutterna för att detta krav ska hållas för varje order genom scenariot.

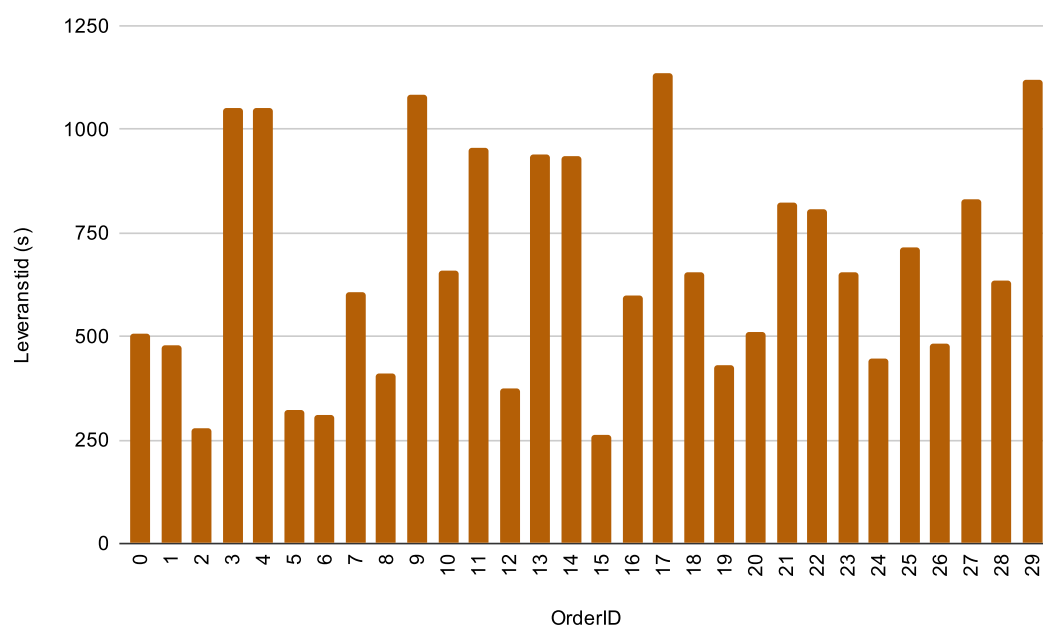
För scenario 4 så kan algoritmernas genomsnittliga precision rangordnas enligt resultaten uppvisade i figur 5.15.

1. TSP - BruteForce uppnår bäst precision med en genomsnittlig leveranstid på ~ 618.3 sekunder.
2. Den giriga approximeringen uppnår bäst precision med en genomsnittlig leveranstid på $\sim 661,6$ sekunder.
3. Den euklidiska approximeringen har en nästan jämlik precision på ~ 669.1 sekunder

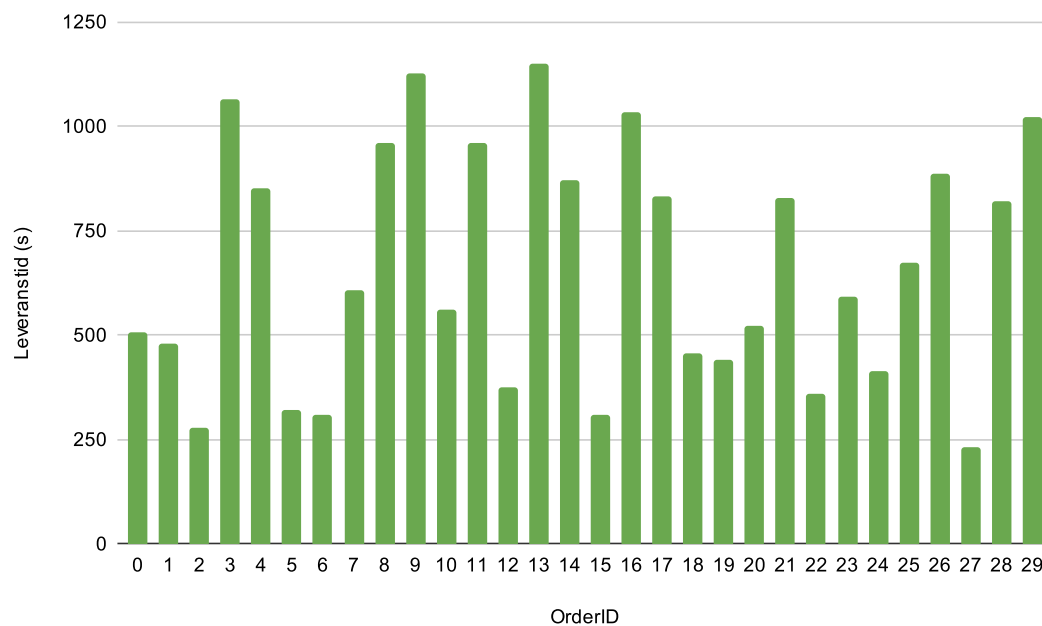
Den genomsnittliga algoritmtiden följer liknande mönster som tidigare scenarion, se figur 5.16.



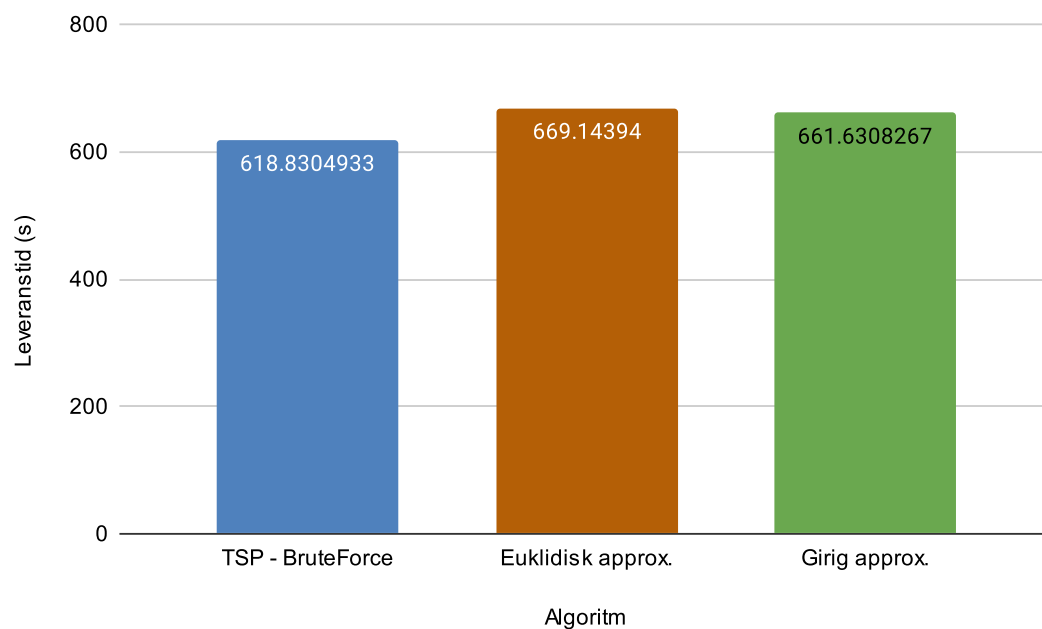
Figur 5.12: TSP - BruteForce leveranstid scenario 4



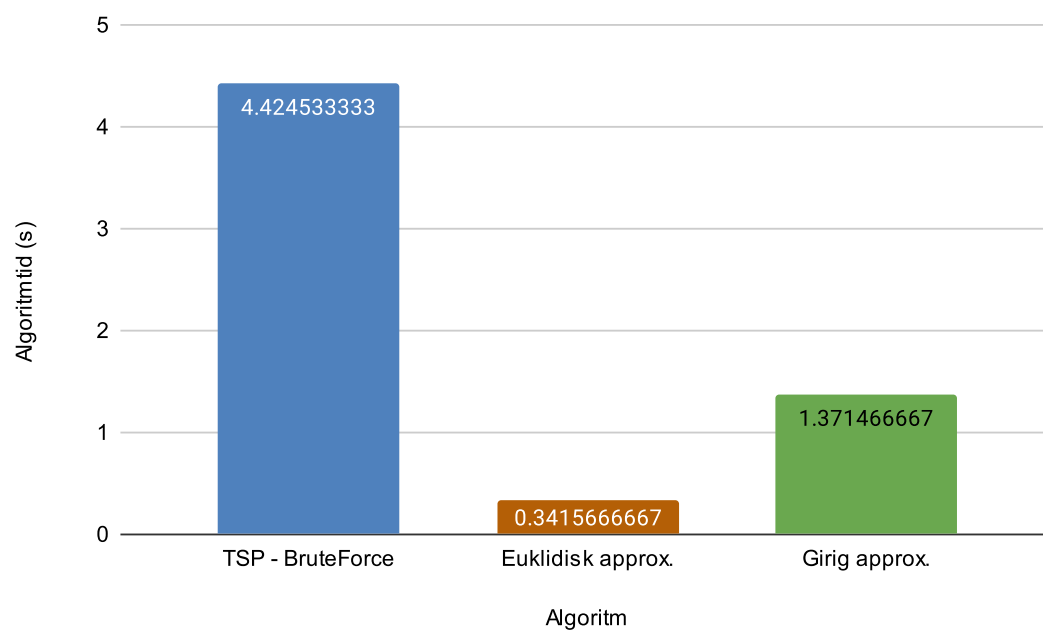
Figur 5.13: Euklidisk approximation leveranstid scenario 4



Figur 5.14: Girig approximation leveranstid scenario 4



Figur 5.15: Genomsnittlig leveranstid scenario 4



Figur 5.16: Genomsnittlig algoritmtid scenario 4

6

Diskussion

I det här kapitlet diskuteras resultaten som framställts. Först diskuteras prestandan och precisionen kring algoritmerna, sedan testning- och simuleringsmiljön. Därefter analyseras testdatan som framställts i de olika scenariorna och slutligen presenterar kapitlet en reflektion och utvärdering kring projektets tidsplan.

6.1 Prestanda och precision

Alla algoritmer följer samma princip när det kommer till utdelning av orders. När en förare har blivit tilldelad en order så tillhör den ordern den föraren resten utav körningen. Detta förbättrar prestanda då det finns mindre jämförelser att ta ställning till för varje given order, men riskerar också sämre precision. Eftersom varje algoritm endast tar ställning till nuvarande order så finns det en teoretisk möjlighet att en algoritm som genererar sämre vägar tidigt i ett scenario, får bättre leveranstid än en algoritm som genererade mer optimerade vägar tidigt i ett scenario. En förare som tar en sämre optimerad väg kan befinna på positioner vid tidpunkter där en bättre optimerad väg ej hade befunnit sig. I ett teoretiskt scenario är det då möjligt att den sämre optimerade föraren av en slump befinner sig nära två nya orders som föraren snabbt kan plocka upp och fortsätta sin körning. Ett liknande fenomen ser vi exempel på i scenario 3 då den giriga approximeringen får en bättre leveranstid än TSP BruteForce, se figur 5.7. Detta sker trots att TSP BruteForce jämför alla möjliga permutationer för varje order.

Det blir även tydligt att algoritmerna inte gör alla de optimeringar som är möjliga, beroende på vad en orders tidsbegränsning är. I scenario 1 har TSP - BruteForce exempelvis en genomsnittlig leveranstid på 629.41 sekunder, se figur 5.1. I scenario 4 som består av samma förare, samma ordrar vid samma tillfällen, men med kortare tidsbegränsning så uppnår TSP - BruteForce en kortare genomsnittlig leveranstid på 618.8 sekunder, se figur 5.15. Även detta är en konsekvens av att ordrar ej kan byta förare efter tilldelning.

Eftersom prestanda mäts i hur lång tid det tar för en algoritm att dela ut en given order så är resultaten oundvikligen något hårdvarubaserade. Rangordningen av algoritmernas prestanda per scenario bör däremot vara densamma oberoende av vilken hårdvara som används. Men det finns en teoretisk möjlighet att alla algoritmer får en betydligt lägre algoritmtid för ett givet scenario på kraftfullare hårdvara. Detta kan påverka eventuella slutsatser. Exempelvis kan då TSP - BruteForce, som kräver

mer prestanda än de andra algoritmerna, fortfarande anses vara tillräckligt effektiv i exekvering för att föredras.

Något som uppenbarar sig i scenario tvås resultat är framskjutningen av order med ID 0 i alla tre algoritmer. Anledningen till att ordern resulterar i en lång leveranstid är på grund av att senare leveranser har en kortare sträcka. Resultatet blir att alla tre algoritmer optimerar vägen till att prioritera dem kortare beställningarna till att levereras innan order 0. I detta scenariot utför endast en förare leveranserna vilket resulterar i detta fallet då en order tar lång tid. Utifall scenariot innehöll fler förare hade detta problemet inte uppstått eftersom en uträkning på total tid för alla orders hade räknats ut och därmed hade ej förare0 tilldelats fler orders. I ett scenario där det är en hög belastning av orders kan fortfarande problemet uppstå då förare med långa leverans sträckor kan bli tilldelade ytterligare orders. En lösning på detta dilemma är att implementera en gräns på hur lång tid en orders leverans kan dröja. Med detta kan en förare inte bli tilldelad orders som resulterar i att en annan leverans överskrider sin maximala leveranstid. I detta teoretiska fall finns det möjlighet att inga förare kan hantera en inkommande order då den överskrider den maximala tiden. Utifall att detta sker hade ingen förare blivit tilldelat ordern och därmed inte levererats.

6.2 Testning- och simuleringsmiljö

Då TSP och VRP är två komplicerade samt ständigt utvecklande ämnen med mycket forskningsintresse så framställs det frekvent nya lösningsförslag och teorier. Den utvecklade simuleringsmiljön har möjlighet att agera som en bra grund för en tredje part att utveckla egna algoritmer. En ny algoritm behöver kommunicera med simulationen och meddela vid vilken tidpunkt nästa order kommer att levereras (om det finns orders utdelade). Detta för att simulationen ska virtuellt passera tiden på ett korrekt sätt vid exekvering. Utvecklingen av en ny algoritm kan även underlättas genom att utnyttja den redan existerande uppbyggnaden av klasser som Orders, Förare, Scenarion och liknande. Användandet av denna grund innebär även att simuleringsmiljön enkelt kan sammanställa körningens resultat i excel-format för senare analys. Den sammanställda datan kan sedan användas för att jämföra med andra algoritmer som körts på samma scenario. Därefter kan en slutsats dras om den nytillkomna algoritmen är en förbättring eller försämring av precision och prestanda. Utifall att den nytillkomna algoritmen vill undersökas utan utomstående faktorer, exempelvis batteriliv, kan dessa stängas av utan att ha någon effekt på simulationens gång. Därmed finns det en flexibilitet att testa nya algoritmer i simulationen.

Som tidigare nämnts är algoritmtiden strikt separerad från den simulerade leveranstiden. Simuleringsmiljön antar alltså att varje order har blivit tilldelad utan tidsfördröjning. I ett verklighetsbaserat scenario är det dock teoretiskt möjligt att en lång algoritmtid kan komma att påverka leveranstiden för en given order, alltså att prestandan påverkar precisionen. Dessa fall kan ej fångas upp av simuleringsmiljöns sätt att modellera prestanda och precision.

Vid större scenarion är det svårt att utläsa exakt vilka val den testade algoritmen har gjort för en given förare. Den genererade resultat-filen visar vilken väg en förare har tagit genom att skriva ut i vilken ordning föraren har besökt order-noderna, men ingen information ges om när vägen genererades eller hur ofta vägen har ändrats under körning. Det som däremot är tillgängligt för analys är den genomsnittliga precisionen samt precisionen per order. Även prestandan går tydligt att se som genomsnitt samt per order.

6.3 Testdata

Rapporten har presenterat resultat från fyra olika scenarion. Dessa scenarion utformades för att testa olika styrkor samt svagheter mellan algoritmerna. Det hade möjligtvis varit bra om man hade haft ännu ett scenario med betydligt fler ordrar över en längre tid. Både TSP - BruteForce och den giriga approximeringen når vid flera scenarion den punkt då deras prestanda når över 1 sekund algoritmtid. Det hade varit intressant att se ifall en sådan punkt existerar för den euklidiska approximeringen. Då den euklidiska approximeringen endast anropar ORS ett fast antal gånger är det däremot inte garanterat att en sådan brytpunkt finns, särskilt inte under ett scenario med tidsbegränsade ordrar.

Stora scenarion innebär dock en risk ur ett analysperspektiv. För större scenarion är det svårare att analysera vad algoritmen gör och vilka beslut den tar. Man kan fortfarande analysera ur ett prestanda mot precision perspektiv, men analysen kring varför en algoritm presterade som den gjorde kan öka i svårighetsgrad.

Varje scenario hade också genomgående samma tidsgräns för alla ordrar. Alltså scenario 1 hade 30 minuter, scenario 2 60 minuter, scenario 3 25 och scenario 4 20 minuter. Det hade varit intressant att analysera ett scenario där tidsbegränsningen per order fluktuerade mellan olika ordrar.

6.4 Utvärdering av tidsplan

Innan arbetet startade framställdes ett GANTT-schema att jobba utefter, se appendix I. Eftersom det var svårt att bedöma exakt vad projektet skulle innehålla innan arbetet startat var de olika momenten i GANTT-schemat mycket generella. I efterhand hade schemat behövts delats upp ytterligare. Exempelvis borde utveckling vara uppdelat i de olika algoritmerna, testning och simulations-miljö osv.

Utvecklingen påbörjades något senare än vad som detaljeras i tidsplanen. Detta berodde på att forskningen kring ämnet tog längre tid än förväntat. Rapportskrivningen påbörjades i tid men inga tydliga framsteg togs förens 3-4 veckor in i planeringen. Förutom detta stämde rapportskrivningen överens med vad som planerats i

GANTT-schemat. Något som inte stämde överens var förberedelsen av presentationen. Detta planerades att ta två veckor men tiden ockuperades av rapportskrivning i slutskedet och därmed flyttades presentationen upp. Presentationen tog en vecka att förbereda i slutändan och detta visade sig vara tillräckligt med tid.

I framtiden bör planeringen vara mer detaljerad. Stora ämnen som utveckling och utvärdering bör delas upp ytterligare för att bättre klargöra vad de innebär. Tiden som krävs för att skapa en presentation kan vara en vecka istället för två och arbetet kring rapportskrivningen bör delas upp mer jämnt kring de veckor som planerats.

7

Slutsats och framtida forskning

Den giriga approximeringen samt TSP - BruteForce uppnår bäst precision på de scenarion algoritmerna har testats mot. TSP - BruteForce har dock en hög kostnad i prestanda med algoritmtider uppemot 160 sekunder i scenario 3, se figur 5.9. Prestandan för TSP - BruteForce har genom de olika scenarion varit för dålig för att anses som ett genomförbart alternativ. Den giriga approximeringen har desto bättre prestanda men i alla scenarion förutom scenario 2 uppnår den vid flera tillfällen en algoritmtid över 1 sekund. I scenario 1, 3 samt 4 överstiger den giriga approximeringens algoritmtid 1 sekund på minst hälften av alla orders. Den euklidiska approximeringen har i alla scenarion bäst prestanda, men precisionen är något sämre än de andra algoritmerna. Då precisionen för den euklidiska approximationen är tillräcklig för att leverera alla ordrar inom tidsgränsen för alla scenarion, anses precisionen vara godtagbar. Utifrån dessa faktorer kan slutsatsen dras att den algoritm som ur ett helhetsperspektiv, med hänsyn till både precision och prestanda, presterat bäst under de givna testscenarierna är den euklidiska approximationen.

Den utvecklade testning- och simuleringsmiljön har möjlighet att nyttjas av en tredje part för att testa ytterligare algoritmer. Detta sker med relativ enkelhet och existerande begränsningar kan enkelt stängas av. Efter exekvering sammanställer simuleringen resultatet i ett lättläst format som kan analyseras för att dra slutsatser kring en testad algoritm, se Appendix E.

En intressant fortsättning på projektet hade varit att förbättra testning- och simuleringsmiljön så att den simulerade tiden tar hänsyn till algoritmtider under körning. Det hade även varit intressant att jämföra olika API:er och dra en slutsats kring vad skillnaden i prestanda och precision blir.

Källhänvisning

- [1] SverigesRadio, “Branscher pekas ut som vinnare under pandemin,” Accessed May. 22, 2022. [Online]. Available: <https://sverigesradio.se/artikel/7516968>
- [2] Bzzt, “Vilka är vi?” Accessed May. 22, 2022. [Online]. Available: <https://www.bzzt.se/vilka-r-vi>
- [3] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem : A Computational Study*. Princeton University Press, 2007.
- [4] C. Chase, harrison Chen, A. Neoh, and M. Wilder-Smith, “An evaluation of the traveling salesman problem,” *California State Polytechnic University, Pomona*, 2020. [Online]. Available: <https://scholarworks.calstate.edu/concern/theses/8g84mp499?locale=en>
- [5] L. Feng, Y. Huang, I. W. Tsang, A. Gupta, K. Tang, K. C. Tan, and Y.-S. Ong, “Towards faster vehicle routing by transferring knowledge from customer representation,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 952–965, 2022. [Online]. Available: 10.1109/TITS.2020.3018903
- [6] M. Zhang, S. Pratap, Z. Zhao, D. Prajapati, and G. Q. Huang, “Forward and reverse logistics vehicle routing problems with time horizons in b2c e-commerce logistics,” *International Journal of Production Research*, vol. 59, no. 20, pp. 6291–6310, 2021. [Online]. Available: 10.1080/00207543.2020.1812749
- [7] L. Zhu and D. Hu, “Study on the vehicle routing problem considering congestion and emission factors.” *International Journal of Production Research*, vol. 57, no. 19, pp. 6115 – 6129, 2019. [Online]. Available: <https://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=bsu&AN=138647979&authtype=sso&custid=s3911979&site=ehost-live&scope=site&authtype=sso&custid=s3911979>
- [8] M. Dziak, “Vehicle routing problem (vrp),” *Salem Press Encyclopedia*, 2020. 2p, 2020. [Online]. Available: <https://search.ebscohost.com/login.aspx?direct=true&db=ers&AN=146920896&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds>
- [9] F. Wang, F. Liao, Y. Li, X. Yan, and X. Chen, “An ensemble learning based multi-objective evolutionary algorithm for the dynamic vehicle routing

- problem with time windows,” *Computers and Industrial Engineering*, vol. 154, p. 107131, 2021. [Online]. Available: <https://doi.org/10.1016/j.cie.2021.107131>
- [10] S. Liu, K. Tang, and X. Yao, “Memetic search for vehicle routing with simultaneous pickup-delivery and time windows,” *Swarm and Evolutionary Computation*, vol. 66, p. 100927, 2021. [Online]. Available: <https://doi.org/10.1016/j.swevo.2021.100927>
- [11] Y. Niu, D. Kong, R. Wen, Z. Cao, and J. Xiao, “An improved learnable evolution model for solving multi-objective vehicle routing problem with stochastic demand,” *Knowledge-Based Systems*, vol. 230, p. 107378, 2021. [Online]. Available: <https://doi.org/10.1016/j.knosys.2021.107378>
- [12] Open Route Service, “Api-documentation,” Accessed May. 22, 2022. [Online]. Available: <https://openrouteservice.org/dev/#/api-docs>
- [13] Docker, “Developers bring their ideas to life with docker,” Accessed May. 17, 2022. [Online]. Available: <https://www.docker.com/why-docker/>
- [14] J. Muli, “Beginning devops with docker.” Birmingham B3 2PB, UK: Packt Publishing, Limited, 2018, pp. 2–4.
- [15] E. Arrington, “Heuristics.” *Salem Press Encyclopedia*, 2021. [Online]. Available: <https://search.ebscohost.com/login.aspx?direct=true&db=ers&AN=89164251&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds>
- [16] Open Route Service, “Plans,” Accessed May. 9, 2022. [Online]. Available: <https://openrouteservice.org/plans/>
- [17] —, “Running-with-docker,” Accessed May. 9, 2022. [Online]. Available: <https://giscience.github.io/openrouteservice/installation/Running-with-Docker.html>

A

Appendix A

Appendix A Scenariokod för Scenario 1. Nedanför listar appendix A komplett scenariokod för scenario 1. För varje order ges det input i ordning av: OrderId, src-nod, dest-nod, tidsgräns, värmeKrav, kylaKrav, platskrav, och slutligen tidpunkt då ordern beställs. För föraren ges det input i ordning av: FörarId, positionsnod, startbatteri, maxkapacitet, nuvarande last, möjlighet till att hålla mat varm, och slutligen möjlighet till att hålla mat kall.

Ordrar:

```
ArrayList<Order> orders = new ArrayList<Order>();
orders.add(new Order(0, new Node("11.97522", "57.6994"), new Node("12.0002", "57.71623"),
30, false, false, 15, 20));
orders.add(new Order(1, new Node("11.98478", "57.69753"), new Node("11.98485",
"57.68894"), 30, false, false, 15, 45));
orders.add(new Order(2, new Node("11.97796", "57.692314"), new Node("11.96601",
"57.68978"), 30, false, false, 15, 70));
orders.add(new Order(3, new Node("12.01285", "57.69969"), new Node("11.97028",
"57.69584"), 30, false, false, 15, 120));
orders.add(new Order(4, new Node("11.95252", "57.69089"), new Node("11.97512",
"57.70033"), 30, false, false, 15, 140));
orders.add(new Order(5, new Node("11.97718", "57.69922"), new Node("11.96531",
"57.6982"), 30, false, false, 15, 180));
orders.add(new Order(6, new Node("11.96688", "57.69842"), new Node("11.97042",
"57.69364"), 30, false, false, 15, 270));
orders.add(new Order(7, new Node("11.98389", "57.68247"), new Node("11.99117",
"57.69078"), 30, false, false, 15, 320));
orders.add(new Order(8, new Node("11.95133", "57.6937"), new Node("11.96665",
"57.69312"), 30, false, false, 15, 350));
orders.add(new Order(9, new Node("11.95687", "57.69902"), new Node("11.96189",
"57.69395"), 30, false, false, 15, 390));
orders.add(new Order(10, new Node("11.96566", "57.70531"), new Node("11.96452",
"57.6985"), 30, false, false, 15, 420));
orders.add(new Order(11, new Node("11.97047", "57.70486"), new Node("11.98331",
"57.70328"), 30, false, false, 15, 430));
orders.add(new Order(12, new Node("11.97767", "57.69401"), new Node("11.97549",
"57.69819"), 30, false, false, 15, 450));
orders.add(new Order(13, new Node("11.95769", "57.70455"), new Node("11.97145",
"57.70136"), 30, false, false, 15, 460));
```

```

orders.add(new Order(14, new Node("11.96515", "57.70867"), new Node("11.97559",
"57.7033"), 30, false, false, 15, 520));
orders.add(new Order(15, new Node("11.95787", "57.70554"), new Node("11.96077",
"57.70101"), 30, false, false, 15, 530));
orders.add(new Order(16, new Node("11.97809", "57.70402"), new Node("11.98369",
"57.69938"), 30, false, false, 15, 540));
orders.add(new Order(17, new Node("11.97129", "57.69651"), new Node("11.98467",
"57.69867"), 30, false, false, 15, 580));
orders.add(new Order(18, new Node("11.95762", "57.69494"), new Node("11.9764",
"57.69993"), 30, false, false, 15, 630));
orders.add(new Order(19, new Node("11.97957", "57.70005"), new Node("11.96328",
"57.69942"), 30, false, false, 15, 645));
orders.add(new Order(20, new Node("11.96291", "57.69712"), new Node("11.97166",
"57.70056"), 30, false, false, 15, 700));
orders.add(new Order(21, new Node("11.96276", "57.70268"), new Node("11.97226",
"57.70638"), 30, false, false, 15, 740));
orders.add(new Order(22, new Node("11.95122", "57.69998"), new Node("11.95013",
"57.69392"), 30, false, false, 15, 790));
orders.add(new Order(23, new Node("11.96677", "57.70583"), new Node("11.98093",
"57.70309"), 30, false, false, 15, 850));
orders.add(new Order(24, new Node("11.95625", "57.69846"), new Node("11.96554",
"57.69775"), 30, false, false, 15, 890));
orders.add(new Order(25, new Node("11.95087", "57.69761"), new Node("11.93586",
"57.69958"), 30, false, false, 15, 940));
orders.add(new Order(26, new Node("11.9492", "57.69627"), new Node("11.96171",
"57.69581"), 30, false, false, 15, 945));
orders.add(new Order(27, new Node("11.96885", "57.69793"), new Node("11.97702",
"57.69515"), 30, false, false, 15, 990));
orders.add(new Order(28, new Node("11.97142", "57.70585"), new Node("11.98545",
"57.70137"), 30, false, false, 15, 1020));
orders.add(new Order(29, new Node("11.96499", "57.70923"), new Node("11.95852",
"57.70295"), 30, false, false, 15, 1060));
return orders;

```

Förare:

```

ArrayList<Driver> returnList = new ArrayList<Driver>();
returnList.add(new Driver(0, new Node("11.97853", "57.70654"), 100.0, 100, 10, true,
true));
returnList.add(new Driver(1, new Node("11.96452", "57.7029"), 100.0, 100, 10, true,
true));
returnList.add(new Driver(2, new Node("11.96979", "57.69861"), 100.0, 1000, 0, true,
true));
returnList.add(new Driver(3, new Node("11.97286", "57.70837"), 100.0, 1000, 0, true,
true));
returnList.add(new Driver(4, new Node("11.98225", "57.70184"), 100.0, 1000, 0, true,
true));

```



```
returnList.add(new Driver(5, new Node("11.96321", "57.69754"), 100.0, 1000, 0, true,
true));
returnList.add(new Driver(6, new Node("11.97809", "57.70037"), 100.0, 1000, 0, true,
true));
returnList.add(new Driver(7, new Node("11.97262", "57.69456"), 100.0, 1000, 0, true,
true));
returnList.add(new Driver(8, new Node("11.9793", "57.69747"), 100.0, 1000, 0, true,
true));
returnList.add(new Driver(9, new Node("11.96852", "57.69541"), 100.0, 1000, 0, true,
true));
return returnList;
```


B

Appendix B

Appendix B Scenariokod för Scenario 2. Nedanför listar appendix B komplett scenariokod för scenario 2. För varje order ges det input i ordning av: OrderId, src-nod, dest-nod, tidsgräns, värmeKrav, kylaKrav, platskrav, och slutligen tidpunkt då ordern beställs. För föraren ges det input i ordning av: FörarId, positionsnod, startbatteri, maxkapacitet, nuvarande last, möjlighet till att hålla mat varm, och slutligen möjlighet till att hålla mat kall.

Ordrar:

```
ArrayList<Order> orders = new ArrayList<Order>();
orders.add(new Order(0, new Node("11.97522", "57.6994"), new Node("12.0002", "57.71623"),
60, false, false, 15, 20));
orders.add(new Order(1, new Node("11.97718", "57.69922"), new Node("11.97226",
"57.70638"), 60, false, false, 15, 400));
orders.add(new Order(2, new Node("11.96566", "57.70531"), new Node("11.96452",
"57.6985"), 60, false, false, 15, 800));
orders.add(new Order(3, new Node("11.95769", "57.70455"), new Node("11.97145",
"57.70136"), 60, false, false, 15, 1200));
orders.add(new Order(4, new Node("11.95762", "57.69494"), new Node("11.9764",
"57.69993"), 60, false, false, 15, 1400));
return orders;
```

Förare:

```
ArrayList<Driver> returnList = new ArrayList<Driver>();
returnList.add(new Driver(0, new Node("11.962725", "57.697525"), 100.0, 100, 10,
true, true));
return returnList;
```


C

Appendix C

Appendix C Scenariokod för Scenario 3. Nedanför listar appendix C komplett scenariokod för scenario 3. För varje order ges det input i ordning av: OrderId, src-nod, dest-nod, tidsgräns, värmeKrav, kylaKrav, platskrav, och slutligen tidpunkt då ordern beställs. För föraren ges det input i ordning av: FörarId, positionsnod, startbatteri, maxkapacitet, nuvarande last, möjlighet till att hålla mat varm, och slutligen möjlighet till att hålla mat kall.

Ordrar:

```
ArrayList<Order> orders = new ArrayList<Order>();
orders.add(new Order(0, new Node("11.97522", "57.6994"), new Node("11.96861",
"57.69564"), 25, false, false, 15, 20));
orders.add(new Order(1, new Node("11.97718", "57.69922"), new Node("11.96531",
"57.6982"), 25, false, false, 15, 25));
orders.add(new Order(2, new Node("11.97388", "57.70179"), new Node("11.96452",
"57.6985"), 25, false, false, 15, 30));
orders.add(new Order(3, new Node("11.97522", "57.6994"), new Node("11.98331",
"57.70328"), 25, false, false, 15, 31));
orders.add(new Order(4, new Node("11.97809", "57.70402"), new Node("11.98369",
"57.69938"), 25, false, false, 15, 70));
orders.add(new Order(5, new Node("11.97718", "57.69922"), new Node("11.98467",
"57.69867"), 25, false, false, 15, 250));
orders.add(new Order(6, new Node("11.97957", "57.70005"), new Node("11.96328",
"57.69942"), 25, false, false, 15, 280));
orders.add(new Order(7, new Node("11.97718", "57.69922"), new Node("11.97226",
"57.70638"), 25, false, false, 15, 300));
orders.add(new Order(8, new Node("11.97243", "57.70162"), new Node("11.98545",
"57.70137"), 25, false, false, 15, 310));
orders.add(new Order(9, new Node("11.97399", "57.6989"), new Node("11.97369",
"57.69387"), 25, false, false, 15, 320));
orders.add(new Order(10, new Node("11.9744", "57.70128"), new Node("11.98123",
"57.70026"), 25, false, false, 15, 325));
orders.add(new Order(11, new Node("11.97388", "57.70179"), new Node("11.96663",
"57.6997"), 25, false, false, 15, 330));
orders.add(new Order(12, new Node("11.97522", "57.70076"), new Node("11.96484",
"57.7029"), 25, false, false, 15, 390));
return orders;
```

Förare:

```
ArrayList<Driver> returnList = new ArrayList<Driver>();  
returnList.add(new Driver(0, new Node("11.97853", "57.70654"), 100.0, 100, 10, true,  
true));  
returnList.add(new Driver(1, new Node("11.96452", "57.7029"), 100.0, 100, 10, true,  
true));  
returnList.add(new Driver(2, new Node("11.96979", "57.69861"), 100.0, 1000, 0, true,  
true));  
returnList.add(new Driver(3, new Node("11.97286", "57.70837"), 100.0, 1000, 0, true,  
true));  
return returnList;
```

D

Appendix D

Appendix D Scenariokod för Scenario 4. Nedanför listar appendix D komplett scenariokod för scenario 4. För varje order ges det input i ordning av: OrderId, src-nod, dest-nod, tidsgräns, värmeKrav, kylaKrav, platskrav, och slutligen tidpunkt då ordern beställs. För föraren ges det input i ordning av: FörarId, positionsnod, startbatteri, maxkapacitet, nuvarande last, möjlighet till att hålla mat varm, och slutligen möjlighet till att hålla mat kall.

Ordrar:

```
ArrayList<Order> orders = new ArrayList<Order>();
orders.add(new Order(0, new Node("11.97522", "57.6994"), new Node("12.0002", "57.71623"),
20, false, false, 15, 20));
orders.add(new Order(1, new Node("11.98478", "57.69753"), new Node("11.98485",
"57.68894"), 20, false, false, 15, 45));
orders.add(new Order(2, new Node("11.97796", "57.692314"), new Node("11.96601",
"57.68978"), 20, false, false, 15, 70));
orders.add(new Order(3, new Node("12.01285", "57.69969"), new Node("11.97028",
"57.69584"), 20, false, false, 15, 120));
orders.add(new Order(4, new Node("11.95252", "57.69089"), new Node("11.97512",
"57.70033"), 20, false, false, 15, 140));
orders.add(new Order(5, new Node("11.97718", "57.69922"), new Node("11.96531",
"57.6982"), 20, false, false, 15, 180));
orders.add(new Order(6, new Node("11.96688", "57.69842"), new Node("11.97042",
"57.69364"), 20, false, false, 15, 270));
orders.add(new Order(7, new Node("11.98389", "57.68247"), new Node("11.99117",
"57.69078"), 20, false, false, 15, 320));
orders.add(new Order(8, new Node("11.95133", "57.6937"), new Node("11.96665",
"57.69312"), 20, false, false, 15, 350));
orders.add(new Order(9, new Node("11.95687", "57.69902"), new Node("11.96189",
"57.69395"), 20, false, false, 15, 390));
orders.add(new Order(10, new Node("11.96566", "57.70531"), new Node("11.96452",
"57.6985"), 20, false, false, 15, 420));
orders.add(new Order(11, new Node("11.97047", "57.70486"), new Node("11.98331",
"57.70328"), 20, false, false, 15, 430));
orders.add(new Order(12, new Node("11.97767", "57.69401"), new Node("11.97549",
"57.69819"), 20, false, false, 15, 450));
orders.add(new Order(13, new Node("11.95769", "57.70455"), new Node("11.97145",
"57.70136"), 20, false, false, 15, 460));
```

```
orders.add(new Order(14, new Node("11.96515", "57.70867"), new Node("11.97559",
"57.7033"), 20, false, false, 15, 520));
orders.add(new Order(15, new Node("11.95787", "57.70554"), new Node("11.96077",
"57.70101"), 20, false, false, 15, 530));
orders.add(new Order(16, new Node("11.97809", "57.70402"), new Node("11.98369",
"57.69938"), 20, false, false, 15, 540));
orders.add(new Order(17, new Node("11.97129", "57.69651"), new Node("11.98467",
"57.69867"), 20, false, false, 15, 580));
orders.add(new Order(18, new Node("11.95762", "57.69494"), new Node("11.9764",
"57.69993"), 20, false, false, 15, 630));
orders.add(new Order(19, new Node("11.97957", "57.70005"), new Node("11.96328",
"57.69942"), 20, false, false, 15, 645));
orders.add(new Order(20, new Node("11.96291", "57.69712"), new Node("11.97166",
"57.70056"), 20, false, false, 15, 700));
orders.add(new Order(21, new Node("11.96276", "57.70268"), new Node("11.97226",
"57.70638"), 20, false, false, 15, 740));
orders.add(new Order(22, new Node("11.95122", "57.69998"), new Node("11.95013",
"57.69392"), 20, false, false, 15, 790));
orders.add(new Order(23, new Node("11.96677", "57.70583"), new Node("11.98093",
"57.70309"), 20, false, false, 15, 850));
orders.add(new Order(24, new Node("11.95625", "57.69846"), new Node("11.96554",
"57.69775"), 20, false, false, 15, 890));
orders.add(new Order(25, new Node("11.95087", "57.69761"), new Node("11.93586",
"57.69958"), 20, false, false, 15, 940));
orders.add(new Order(26, new Node("11.9492", "57.69627"), new Node("11.96171",
"57.69581"), 20, false, false, 15, 945));
orders.add(new Order(27, new Node("11.96885", "57.69793"), new Node("11.97702",
"57.69515"), 20, false, false, 15, 990));
orders.add(new Order(28, new Node("11.97142", "57.70585"), new Node("11.98545",
"57.70137"), 20, false, false, 15, 1020));
orders.add(new Order(29, new Node("11.96499", "57.70923"), new Node("11.95852",
"57.70295"), 20, false, false, 15, 1060));
return orders;
```

Förare:

```
ArrayList<Driver> returnList = new ArrayList<Driver>();
returnList.add(new Driver(0, new Node("11.97853", "57.70654"), 100.0, 100, 10, true,
true));
returnList.add(new Driver(1, new Node("11.96452", "57.7029"), 100.0, 100, 10, true,
true));
returnList.add(new Driver(2, new Node("11.96979", "57.69861"), 100.0, 1000, 0, true,
true));
returnList.add(new Driver(3, new Node("11.97286", "57.70837"), 100.0, 1000, 0, true,
true));
returnList.add(new Driver(4, new Node("11.98225", "57.70184"), 100.0, 1000, 0, true,
true));
```



```
returnList.add(new Driver(5, new Node("11.96321", "57.69754"), 100.0, 1000, 0, true,
true));
returnList.add(new Driver(6, new Node("11.97809", "57.70037"), 100.0, 1000, 0, true,
true));
returnList.add(new Driver(7, new Node("11.97262", "57.69456"), 100.0, 1000, 0, true,
true));
returnList.add(new Driver(8, new Node("11.9793", "57.69747"), 100.0, 1000, 0, true,
true));
returnList.add(new Driver(9, new Node("11.96852", "57.69541"), 100.0, 1000, 0, true,
true));
return returnList;
```


E

Appendix E

Appendix E innehåller den kompletta datan för de tre algoritmerna efter en körning genom scenario 1. Kolumnerna beskriver vad varje cell innehåller och varje rad representerar en order. Längst ner i leveranstid och algoritmtid kolumnerna finns genomsnittet av respektive kolumn.

OrderID	Förare	Order beställdes (s)	Order delades ut (s)	Order plockades upp (s)	Order levererades (s)	Leveranstid (s)	Från	Till	Algoritmtid (s)	Förare	Väg
0	Driver2	20	20	62.3002	525.6004	505.6004	[11.97522,57.6994]	[12.0002,57.71623]	0.673	Driver0	S9->D9->S20->D20->S25->D25
1	Driver4	45	45	154.3002	523.3002	478.3002	[11.98478,57.69753]	[11.98485,57.68894]	0.505	Driver1	S10->S14->D10->D14->S29->D29
2	Driver8	70	70	155.4002	346.6004	276.6004	[11.97796,57.692314]	[11.96601,57.68978]	0.476	Driver2	S0->D0->S16->D16->S27->D27
3	Driver6	120	120	465.7002	1495.3008	1375.3008	[12.01285,57.69969]	[11.97028,57.69584]	0.808	Driver3	S11->S23->D23->D11
4	Driver5	140	140	334.2002	1529.5008	1389.5008	[11.95252,57.69089]	[11.97512,57.70033]	0.752	Driver4	S1->D1->S7->D7
5	Driver7	180	180	281.8002	501.2002	321.2002	[11.97718,57.69922]	[11.96531,57.6982]	0.539	Driver5	S4->S8->D8->S18->S26->D26->D4->D18
6	Driver9	270	270	323.6002	561.8002	291.8002	[11.96688,57.69842]	[11.97042,57.69364]	0.673	Driver6	S3->S19->S24->D19->D24->D3
7	Driver4	320	320	719.8004	927.3006	607.3006	[11.98389,57.68247]	[11.99117,57.69078]	0.778	Driver7	S5->D5->S13->S15->D15->S21->D13->D21
8	Driver5	350	350	455.8002	759.2002	409.2002	[11.95133,57.6937]	[11.96665,57.69312]	1.239	Driver8	S2->D2->S22->D22
9	Driver0	390	390	730.9002	926.5004	536.5004	[11.95687,57.69902]	[11.96189,57.69395]	2.72	Driver9	S6->D6->S12->S17->D12->S28->D28->D17
10	Driver1	420	420	500.7002	1103.0002	683.0002	[11.96566,57.70531]	[11.96452,57.6985]	3.603		
11	Driver3	430	430	698.1002	1370.6006	940.6006	[11.97047,57.70486]	[11.98331,57.70328]	2.426		
12	Driver9	450	450	688.9002	936.0006	486.0006	[11.97767,57.69401]	[11.97549,57.69819]	2.603		
13	Driver7	460	460	677.8002	1127.5006	667.5006	[11.95769,57.70455]	[11.97145,57.70136]	2.355		
14	Driver1	520	520	708.4002	1253.0004	733.0004	[11.96515,57.70867]	[11.97559,57.7033]	2.487		
15	Driver7	530	530	695.6004	804.3002	274.3002	[11.95787,57.70554]	[11.96077,57.70101]	3.805		
16	Driver2	540	540	981.9002	1088.9002	548.9002	[11.97809,57.70402]	[11.98369,57.69938]	4.499		
17	Driver9	580	580	821.3004	1587.8006	1007.8006	[11.97129,57.69651]	[11.98467,57.69867]	4.382		
18	Driver5	630	630	994.1002	1623.101	993.101	[11.95762,57.69494]	[11.9764,57.69993]	7.865		
19	Driver6	645	645	844.4002	1360.7004	715.7004	[11.97957,57.70005]	[11.96328,57.69942]	11.292		
20	Driver0	700	700	1014.8002	1164.1004	464.1004	[11.96291,57.69712]	[11.97166,57.70056]	8.648		
21	Driver7	740	740	889.6004	1296.3008	556.3008	[11.96276,57.70268]	[11.97226,57.70638]	8.731		
22	Driver8	790	790	1363.1002	1480.5004	690.5004	[11.95122,57.69998]	[11.95013,57.69392]	8.381		
23	Driver3	850	850	1101.2002	1335.3004	485.3004	[11.96677,57.70583]	[11.98093,57.70309]	4.166		
24	Driver6	890	890	1287.2002	1433.6006	543.6006	[11.95625,57.69846]	[11.96554,57.69775]	3.206		
25	Driver0	940	940	1425.7006	1556.1008	616.1008	[11.95087,57.69761]	[11.93586,57.69958]	6.78		
26	Driver5	945	945	1119.7004	1321.0006	376.0006	[11.9492,57.69627]	[11.96171,57.69581]	8.308		
27	Driver2	990	990	1270.9004	1385.3006	395.3006	[11.96885,57.69793]	[11.97702,57.69515]	24.453		
28	Driver9	1020	1020	1270.7002	1525.1004	505.1004	[11.97142,57.70585]	[11.98545,57.70137]	12.911		
29	Driver1	1060	1060	1603.5006	2068.9008	1008.9008	[11.96499,57.70923]	[11.95852,57.70295]	14.338		
						629.4138267			5.146733333		

Figur E.1: BruteForce resultat scenario 1

E. Appendix E

OrderID	Förare	Order beställides (s)	Order delades ut (s)	Order plockades upp (s)	Order levererades (s)	Leveranstid (s)	Från	Till	Algoritmtid (s)	Förare	Väg
0	Driver2	20	20	62.3002	525.6004	505.6004	[11.97522,57.6994]	[12.0002,57.71623]	0.706	Driver0	S10->S24->S20->D24->D10->D20
1	Driver4	45	45	154.3002	523.3002	478.3002	[11.98478,57.69753]	[11.98485,57.68894]	0.129	Driver1	S15->D15->S22->S26->D22->S9->D26->D9
2	Driver8	70	70	155.4002	346.6004	276.6004	[11.97796,57.692314]	[11.96601,57.68978]	0.115	Driver2	S0->D0
3	Driver6	120	120	465.7002	1173.7006	1053.7006	[12.01285,57.69969]	[11.97028,57.69584]	0.21	Driver3	S11->S21->S28->D21->D11->D28
4	Driver5	140	140	334.2002	1193.3002	1053.3002	[11.95252,57.69089]	[11.97512,57.70033]	0.169	Driver4	S1->D1->S7->D7
5	Driver7	180	180	281.8002	501.2002	321.2002	[11.97718,57.69922]	[11.96531,57.6982]	0.352	Driver5	S4->S8->D8->S18->D4->D18->S29->D29
6	Driver9	270	270	323.6002	579.2002	309.2002	[11.96688,57.69842]	[11.97042,57.69364]	0.18	Driver6	S3->S19->D19->D3
7	Driver4	320	320	719.8004	927.3006	607.3006	[11.98389,57.68247]	[11.99117,57.69078]	0.304	Driver7	S5->D5->S13->S14->S23->D13->D14->D23
8	Driver5	350	350	455.8002	759.2002	409.2002	[11.95133,57.6937]	[11.96665,57.69312]	0.229	Driver8	S2->D2->S17->S27->D27->D17
9	Driver1	390	390	1362.9008	1558.5012	1168.5012	[11.95687,57.69902]	[11.96189,57.69395]	0.24	Driver9	S6->D6->S12->D12->S16->D16->S25->D25
10	Driver0	420	420	666.0002	1321.7008	901.7008	[11.96566,57.70531]	[11.96452,57.6985]	0.194		
11	Driver3	430	430	698.1002	1676.0008	1246.0008	[11.97047,57.70486]	[11.98331,57.70328]	0.234		
12	Driver9	450	450	690.2004	825.9006	375.9006	[11.97767,57.69401]	[11.97549,57.69819]	0.232		
13	Driver7	460	460	667.8002	1399.3006	939.3006	[11.95769,57.70455]	[11.97145,57.70136]	0.263		
14	Driver7	520	520	888.0002	1454.4008	934.4008	[11.96515,57.70867]	[11.97559,57.7033]	0.273		
15	Driver1	530	530	684.2002	837.8002	307.8002	[11.95787,57.70554]	[11.96077,57.70101]	0.305		
16	Driver9	540	540	1031.0002	1137.0004	597.0004	[11.97809,57.70402]	[11.98369,57.69938]	0.311		
17	Driver8	580	580	1028.3002	1413.9008	833.9008	[11.97129,57.69651]	[11.98467,57.69867]	0.297		
18	Driver5	630	630	954.0004	1286.9004	656.9004	[11.95762,57.69494]	[11.9764,57.69993]	0.326		
19	Driver6	645	645	844.4002	1076.0004	431.0004	[11.97957,57.70005]	[11.96328,57.69942]	0.298		
20	Driver0	700	700	1243.8004	1453.701	753.701	[11.96291,57.69712]	[11.97166,57.70056]	0.333		
21	Driver3	740	740	1079.3002	1508.1006	768.1006	[11.96276,57.70268]	[11.97226,57.70638]	0.363		
22	Driver1	790	790	1051.1002	1189.1006	399.1006	[11.95122,57.69998]	[11.95013,57.69392]	0.494		
23	Driver7	850	850	1202.1004	1503.701	653.701	[11.96677,57.70583]	[11.98093,57.70309]	0.309		
24	Driver0	890	890	1111.1002	1275.3006	385.3006	[11.95625,57.69846]	[11.96554,57.69775]	0.706		
25	Driver9	940	940	1523.9006	1654.3008	714.3008	[11.95087,57.69761]	[11.93586,57.69958]	1.023		
26	Driver1	945	945	1132.1004	1511.701	566.701	[11.9492,57.69627]	[11.96171,57.69581]	0.445		
27	Driver8	990	990	1107.7004	1222.1006	232.1006	[11.96885,57.69793]	[11.97702,57.69515]	0.459		
28	Driver3	1020	1020	1443.2004	1727.401	707.401	[11.97142,57.70585]	[11.98545,57.70137]	0.453		
29	Driver5	1060	1060	1714.4006	2179.8008	1119.8008	[11.96499,57.70923]	[11.95852,57.70295]	0.408		
						656.9006			0.3453333333		

Figur E.2: Euklidisk approximation resultat scenario 1

OrderID	Förare	Order beställides (s)	Order delades ut (s)	Order plockades upp (s)	Order levererades (s)	Leveranstid (s)	Från	Till	Algoritmtid (s)	Förare	Väg
0	Driver2	20	20	62.3002	525.6004	905.6004	[11.97522,57.6994]	[12.0002,57.71623]	0.705	Driver0	S10->D10->S20->S20->D20->S25->D25
1	Driver4	45	45	154.3002	523.3002	478.3002	[11.98478,57.69753]	[11.98485,57.68894]	0.209	Driver1	S15->D15->S22->S22->D22->S9->D9
2	Driver8	70	70	155.4002	346.6004	276.6004	[11.97796,57.692314]	[11.96601,57.68978]	0.134	Driver2	S0->D0
3	Driver6	120	120	465.7002	1173.7006	1053.7006	[12.01285,57.69969]	[11.97028,57.69584]	0.345	Driver3	S11->S21->S21->D11->D21->S28->D28
4	Driver5	140	140	334.2002	975.1006	835.1006	[11.95252,57.69089]	[11.97512,57.70033]	0.18	Driver4	S1->D1->S7->D7
5	Driver7	180	180	281.8002	501.2002	321.2002	[11.97718,57.69922]	[11.96531,57.6982]	0.312	Driver5	S4->S8->S18->S18->D4->D18->S27->D8->D27
6	Driver9	270	270	323.6002	579.2002	309.2002	[11.96688,57.69842]	[11.97042,57.69364]	0.303	Driver6	S3->S19->D19->D3
7	Driver4	320	320	719.8004	927.3006	607.3006	[11.98389,57.68247]	[11.99117,57.69078]	0.397	Driver7	S5->D5->S13->S14->S23->S23->D14->D23->D13->S29->D29
8	Driver5	350	350	455.8002	1347.801	997.801	[11.95133,57.6937]	[11.96665,57.69312]	0.176	Driver8	S2->D2->S26->D26->S17->D17
9	Driver1	390	390	1321.8016	1517.4018	1127.4018	[11.95687,57.69902]	[11.96189,57.69395]	0.534	Driver9	S6->D6->S12->S12->D12->S24->D24->S16->D16
10	Driver0	420	420	666.0002	983.0002	563.0002	[11.96566,57.70531]	[11.96452,57.6985]	0.242		
11	Driver3	430	430	698.1002	1389.4006	959.4006	[11.97047,57.70486]	[11.98331,57.70328]	0.634		
12	Driver9	450	450	690.2008	825.901	375.901	[11.97767,57.69401]	[11.97549,57.69819]	0.586		
13	Driver7	460	460	667.8002	1671.6016	1211.6016	[11.95769,57.70455]	[11.97145,57.70136]	0.806		
14	Driver7	520	520	888.0004	1452.9006	932.9006	[11.96515,57.70867]	[11.97559,57.7033]	0.728		
15	Driver1	530	530	684.2002	837.8002	307.8002	[11.95787,57.70554]	[11.96077,57.70101]	0.93		
16	Driver9	540	540	1469.3006	1575.3008	1035.3008	[11.97809,57.70402]	[11.98369,57.69938]	1.646		
17	Driver8	580	580	1606.7006	1825.2008	1245.2008	[11.97129,57.69651]	[11.98467,57.69867]	2.128		
18	Driver5	630	630	758.8004	1072.2002	442.2002	[11.95762,57.69494]	[11.9764,57.69993]	1.375		
19	Driver6	645	645	844.4002	1076.0004	431.0004	[11.97957,57.70005]	[11.96328,57.69942]	1.175		
20	Driver0	700	700	1072.7008	1222.001	522.001	[11.96291,57.69712]	[11.97166,57.70056]	1.317		
21	Driver3	740	740	1079.3004	1566.601	826.601	[11.96276,57.70268]	[11.97226,57.70638]	1.614		
22	Driver1	790	790	1030.6008	1148.001	358.001	[11.95122,57.69998]	[11.95013,57.69392]	1.564		
23	Driver7	850	850	1262.7004	1502.201	652.201	[11.96677,57.70583]	[11.98093,57.70309]	3.049		
24	Driver9	890	890	1156.3002	1302.7004	412.7004	[11.95625,57.69846]	[11.96554,57.69775]	1.214		
25	Driver0	940	940	1483.6016	1614.0018	674.0018	[11.95087,57.69761]	[11.93586,57.69958]	1.187		
26	Driver8	945	945	1257.0002	1458.3004	513.3004	[11.9492,57.69627]	[11.96171,57.69581]	1.611		
27	Driver5	990	990	1241.2008	1508.3014	518.3014	[11.96885,57.69793]	[11.97702,57.69515]	1.361		
28	Driver3	1020	1020	1587.6016	1842.0018	822.0018	[11.97142,57.70585]	[11.98545,57.70137]	2.775		
29	Driver7	1060	1060	1989.202	2454.6022	1394.6022	[11.96499,57.70923]	[11.95852,57.70295]	2.027		
						690.3408133			1.042133333		

Figur E.3: Girig approximation resultat scenario 1

F

Appendix F

Appendix F innehåller den kompletta datan för de tre algoritmerna efter en körning genom scenario 2. Kolumnerna beskriver vad varje cell innehåller och varje rad representerar en order. Längst ner i leveranstid och algoritmtid kolumnerna finns genomsnittet av respektive kolumn.

OrderID	Förare	Order beställdes (s)	Order delades ut (s)	Order plockades upp (s)	Order levererades (s)	Leveranstid (s)	Från	Till	Algoritmtid (s)	Förare	Väg
0	Driver0	20	20	137.5002	2746.9012	2726.9012	[11.97522,57.6994]	[12.0002,57.71623]	0.66	Driver0	S0->S1->S2->S3->S4->D2->D3->D4->D1->D0
1	Driver0	400	400	646.1002	2342.901	1942.901	[11.97718,57.69922]	[11.97226,57.70638]	0.119		
2	Driver0	800	800	1037.8002	1857.5004	1057.5004	[11.96566,57.70531]	[11.96452,57.6985]	0.577		
3	Driver0	1200	1200	1366.9002	1952.3006	752.3006	[11.95769,57.70455]	[11.97145,57.70136]	1.656		
4	Driver0	1400	1400	1726.9002	2074.2008	674.2008	[11.95762,57.69494]	[11.9764,57.69993]	11.202		
						1430.7608			2.8428		

Figur F.1: BruteForce resultat scenario 2

OrderID	Förare	Order beställdes (s)	Order delades ut (s)	Order plockades upp (s)	Order levererades (s)	Leveranstid (s)	Från	Till	Algoritmtid (s)	Förare	Väg
0	Driver0	20	20	137.5002	2746.9012	2726.9012	[11.97522,57.6994]	[12.0002,57.71623]	0.519	Driver0	S0->S1->S2->S3->S4->D2->D3->D4->D1->D0
1	Driver0	400	400	646.1002	2342.901	1942.901	[11.97718,57.69922]	[11.97226,57.70638]	0.067		
2	Driver0	800	800	1037.8002	1857.5004	1057.5004	[11.96566,57.70531]	[11.96452,57.6985]	0.077		
3	Driver0	1200	1200	1366.9002	1952.3006	752.3006	[11.95769,57.70455]	[11.97145,57.70136]	0.081		
4	Driver0	1400	1400	1726.9002	2074.2008	674.2008	[11.95762,57.69494]	[11.9764,57.69993]	0.122		
						1430.7608			0.1732		

Figur F.2: Euklidisk approximation resultat scenario 2

OrderID	Förare	Order beställdes (s)	Order delades ut (s)	Order plockades upp (s)	Order levererades (s)	Leveranstid (s)	Från	Till	Algoritmtid (s)	Förare	Väg
0	Driver0	20	20	137.5002	2845.5026	2825.5026	[11.97522,57.6994]	[12.0002,57.71623]	0.617	Driver0	S0->S1->D1->S2->D2->S4->S3->S3->D3->D4->D0
1	Driver0	400	400	646.1002	941.0002	541.0002	[11.97718,57.69922]	[11.97226,57.70638]	0.172		
2	Driver0	800	800	1324.1004	1603.5002	803.5002	[11.96566,57.70531]	[11.96452,57.6985]	0.339		
3	Driver0	1200	1200	1989.5012	2262.5014	1062.5014	[11.95769,57.70455]	[11.97145,57.70136]	0.355		
4	Driver0	1400	1400	1743.4008	2384.402	984.402	[11.95762,57.69494]	[11.9764,57.69993]	0.489		
						1243.38128			0.3944		

Figur F.3: Girig approximation resultat scenario 2

G

Appendix G

Appendix G innehåller den kompletta datan för de tre algoritmerna efter en körning genom scenario 3. Kolumnerna beskriver vad varje cell innehåller och varje rad representerar en order. Längst ner i leveranstid och algoritmtid kolumnerna finns genomsnittet av respektive kolumn.

OrderID	Förare	Order beställdes (s)	Order delades ut (s)	Order plockades upp (s)	Order levererades (s)	Leveranstid (s)	Från	Till	Algoritmtid (s)	Förare	Väg
0	Driver2	20	20	67.3002	296.0006	276.0006	[11.97522,57.6994]	[11.96861,57.69564]	0.654	Driver0	S2->S2->S11->S12->S6->D2->D6->D11->D12
1	Driver2	25	25	103.5004	361.6002	336.6002	[11.97718,57.69922]	[11.96531,57.6982]	0.395	Driver1	S3->S9->S5->D3->D5->D9
2	Driver0	30	30	460.2002	821.9008	791.9008	[11.97388,57.70179]	[11.96452,57.6985]	2.854	Driver2	S0->S1->D0->D1->S7->D7
3	Driver1	31	31	338.1002	638.9008	607.9008	[11.97522,57.6994]	[11.98331,57.70328]	2.948	Driver3	S8->S10->S4->D8->D4->D10
4	Driver3	70	70	578.3006	715.301	645.301	[11.97809,57.70402]	[11.98369,57.69938]	1.049		
5	Driver1	250	250	454.6006	753.001	503.001	[11.97718,57.69922]	[11.98467,57.69867]	0.701		
6	Driver0	280	280	615.8006	866.301	586.301	[11.97957,57.70005]	[11.96328,57.69942]	2.93		
7	Driver2	300	300	519.2004	808.9006	508.9006	[11.97718,57.69922]	[11.97226,57.70638]	3.907		
8	Driver3	310	310	432.4002	668.8008	358.8008	[11.97243,57.70162]	[11.98545,57.70137]	4.405		
9	Driver1	320	320	408.5004	915.3012	595.3012	[11.97399,57.6989]	[11.97369,57.69387]	6.333		
10	Driver3	325	325	486.2004	751.2012	426.2012	[11.9744,57.70128]	[11.98123,57.70026]	89.26		
11	Driver0	330	330	460.2002	900.8012	570.8012	[11.97388,57.70179]	[11.96663,57.6997]	166.364		
12	Driver0	390	390	531.8004	990.0014	600.0014	[11.97522,57.70076]	[11.96484,57.7029]	115.983		
						523.6162923			30.59869231		

Figur G.1: BruteForce resultat scenario 3

OrderID	Förare	Order beställdes (s)	Order delades ut (s)	Order plockades upp (s)	Order levererades (s)	Leveranstid (s)	Från	Till	Algoritmtid (s)	Förare	Väg
0	Driver2	20	20	67.3002	296.0006	276.0006	[11.97522,57.6994]	[11.96861,57.69564]	0.695	Driver0	S2->S2->S11->S12->S6->D11->D2->D6->D12
1	Driver2	25	25	103.5004	361.6002	336.6002	[11.97718,57.69922]	[11.96531,57.6982]	0.14	Driver1	S3->S9->S5->D3->D5->D9
2	Driver0	30	30	460.2002	883.401	853.401	[11.97388,57.70179]	[11.96452,57.6985]	0.184	Driver2	S0->S1->D0->D1->S7->D7
3	Driver1	31	31	338.1002	638.9008	607.9008	[11.97522,57.6994]	[11.98331,57.70328]	0.215	Driver3	S8->S10->S4->D8->D4->D10
4	Driver3	70	70	578.3006	715.301	645.301	[11.97809,57.70402]	[11.98369,57.69938]	0.332		
5	Driver1	250	250	454.6006	753.001	503.001	[11.97718,57.69922]	[11.98467,57.69867]	0.222		
6	Driver0	280	280	615.8006	927.8012	647.8012	[11.97957,57.70005]	[11.96328,57.69942]	0.313		
7	Driver2	300	300	519.2004	808.9006	508.9006	[11.97718,57.69922]	[11.97226,57.70638]	0.252		
8	Driver3	310	310	432.4002	668.8008	358.8008	[11.97243,57.70162]	[11.98545,57.70137]	0.362		
9	Driver1	320	320	408.5004	915.3012	595.3012	[11.97399,57.6989]	[11.97369,57.69387]	0.273		
10	Driver3	325	325	486.2004	751.2012	426.2012	[11.9744,57.70128]	[11.98123,57.70026]	0.378		
11	Driver0	330	330	460.2002	814.8008	484.8008	[11.97388,57.70179]	[11.96663,57.6997]	0.445		
12	Driver0	390	390	531.8004	1010.5014	620.5014	[11.97522,57.70076]	[11.96484,57.7029]	0.528		
						528.0393692			0.3337692308		

Figur G.2: Euklidisk approximation resultat scenario 3

G. Appendix G

OrderID	Förare	Order beställdes (s)	Order delades ut (s)	Order plockades upp (s)	Order levererades (s)	Leveranstid (s)	Från	Till	Algoritmtid (s)	Förare	Väg
0	Driver2	20	20	67.3002	296.0006	276.0006	[11.97522,57.6994]	[11.96861,57.69564]	0.569	Driver0	S2->D2->S11->S12->S12->S6->S6->D11->D6->D12
1	Driver2	25	25	103.5004	361.6002	336.6002	[11.97718,57.69922]	[11.96531,57.6982]	0.167	Driver1	S3->S5->S5->D5->D3->S10->D10
2	Driver0	30	30	170.9002	389.8002	359.8002	[11.97388,57.70179]	[11.96452,57.6985]	1.049	Driver2	S0->S1->D0->D1->S7->D7
3	Driver1	31	31	322.4002	660.1012	629.1012	[11.97522,57.6994]	[11.98331,57.70328]	0.532	Driver3	S4->S4->D4->S9->S9->S8->S8->D8->D9
4	Driver3	70	70	370.3004	476.3006	406.3006	[11.97809,57.70402]	[11.98369,57.69938]	0.51		
5	Driver1	250	250	359.2004	526.7006	276.7006	[11.97718,57.69922]	[11.98467,57.69867]	0.989		
6	Driver0	280	280	677.1012	958.7018	678.7018	[11.97957,57.70005]	[11.96328,57.69942]	1.116		
7	Driver2	300	300	519.2004	808.9006	508.9006	[11.97718,57.69922]	[11.97226,57.70638]	1.634		
8	Driver3	310	310	710.3018	846.502	536.502	[11.97243,57.70162]	[11.98545,57.70137]	1.688		
9	Driver3	320	320	629.1014	1033.6026	713.6026	[11.97399,57.6989]	[11.97369,57.69387]	2.908		
10	Driver1	325	325	851.8016	969.1018	644.1018	[11.9744,57.70128]	[11.98123,57.70026]	1.719		
11	Driver0	330	330	521.5004	876.1014	546.1014	[11.97388,57.70179]	[11.96663,57.6997]	2.515		
12	Driver0	390	390	593.1008	1041.4024	651.4024	[11.97522,57.70076]	[11.96484,57.7029]	2.379		
						504.9089231			1.367307692		

Figur G.3: Girig approximation resultat scenario 3

H

Appendix H

Appendix H innehåller den kompletta datan för de tre algoritmerna efter en körning genom scenario 4. Kolumnerna beskriver vad varje cell innehåller och varje rad representerar en order. Längst ner i leveranstid och algoritmtid kolumnerna finns genomsnittet av respektive kolumn.

OrderID	Förare	Order beställdes (s)	Order delades ut (s)	Order plockades upp (s)	Order levererades (s)	Leveranstid (s)	Från	Till	Algoritmtid (s)	Förare	Väg
0	Driver2	20	20	62.3002	525.6004	505.6004	[11.97522,57.6994]	[12.0002,57.71623]	0.78	Driver0	S9->D9->S20->D20->S25->D25
1	Driver4	45	45	154.3002	523.3002	478.3002	[11.98478,57.69753]	[11.98485,57.68894]	0.606	Driver1	S10->S14->S24->D24->D10->D14
2	Driver8	70	70	155.4002	346.6004	276.6004	[11.97796,57.692314]	[11.96601,57.68978]	0.359	Driver2	S0->D0->S16->D16->S27->D27
3	Driver6	120	120	465.7002	1185.0004	1065.0004	[12.01285,57.69969]	[11.97028,57.69584]	0.47	Driver3	S11->S23->D23->D11
4	Driver5	140	140	334.2002	1170.3006	1030.3006	[11.95252,57.69089]	[11.97512,57.70033]	0.586	Driver4	S1->D1->S7->D7
5	Driver7	180	180	281.8002	501.2002	321.2002	[11.97718,57.69922]	[11.96531,57.6982]	0.653	Driver5	S4->S8->D8->S18->D4->D18
6	Driver9	270	270	323.6002	561.8002	291.8002	[11.96688,57.69842]	[11.97042,57.69364]	1.167	Driver6	S3->S19->D19->D3->S29->D29
7	Driver4	320	320	719.8004	927.3006	607.3006	[11.98389,57.68247]	[11.99117,57.69078]	1.451	Driver7	S5->D5->S13->S15->D15->S21->D13->D21
8	Driver5	350	350	455.8002	759.2002	409.2002	[11.95133,57.6937]	[11.96665,57.69312]	1.868	Driver8	S2->D2->S26->S22->D22->D26
9	Driver0	390	390	730.9002	926.5004	536.5004	[11.95687,57.69902]	[11.96189,57.69395]	3.177	Driver9	S6->D6->S12->S17->D12->S28->D28->D17
10	Driver1	420	420	500.7002	1420.6006	1000.6006	[11.96566,57.70531]	[11.96452,57.6985]	2.147		
11	Driver3	430	430	698.1002	1370.6006	940.6006	[11.97047,57.70486]	[11.98331,57.70328]	1.98		
12	Driver9	450	450	688.9002	936.0006	486.0006	[11.97767,57.69401]	[11.97549,57.69819]	2.044		
13	Driver7	460	460	677.8002	1127.5006	667.5006	[11.95769,57.70455]	[11.97145,57.70136]	2.426		
14	Driver1	520	520	708.4002	1570.6008	1050.6008	[11.96515,57.70867]	[11.97559,57.7033]	2.079		
15	Driver7	530	530	695.6004	804.3002	274.3002	[11.95787,57.70554]	[11.96077,57.70101]	3.231		
16	Driver2	540	540	981.9002	1088.9002	548.9002	[11.97809,57.70402]	[11.98369,57.69938]	4.973		
17	Driver9	580	580	821.3004	1587.8006	1007.8006	[11.97129,57.69651]	[11.98467,57.69867]	5.298		
18	Driver5	630	630	954.0004	1263.9008	633.9008	[11.95762,57.69494]	[11.9764,57.69993]	8.054		
19	Driver6	645	645	844.4002	1087.3002	442.3002	[11.97957,57.70005]	[11.96328,57.69942]	12.935		
20	Driver0	700	700	1014.8002	1164.1004	464.1004	[11.96291,57.69712]	[11.97166,57.70056]	8.521		
21	Driver7	740	740	889.6004	1296.3008	556.3008	[11.96276,57.70268]	[11.97226,57.70638]	7.584		
22	Driver8	790	790	1387.3004	1504.7006	714.7006	[11.95122,57.69998]	[11.95013,57.69392]	6.708		
23	Driver3	850	850	1101.2002	1335.3004	485.3004	[11.96677,57.70583]	[11.98093,57.70309]	3.06		
24	Driver1	890	890	1227.8002	1374.2004	484.2004	[11.95625,57.69846]	[11.96554,57.69775]	3.028		
25	Driver0	940	940	1425.7006	1556.1008	616.1008	[11.95087,57.69761]	[11.93586,57.69958]	5.833		
26	Driver8	945	945	1307.6002	1693.9008	748.9008	[11.9492,57.69627]	[11.96171,57.69581]	9.582		
27	Driver2	990	990	1270.9004	1385.3006	395.3006	[11.96885,57.69793]	[11.97702,57.69515]	11.124		
28	Driver9	1020	1020	1270.7002	1525.1004	505.1004	[11.97142,57.70585]	[11.98545,57.70137]	9.328		
29	Driver6	1060	1060	1615.2006	2080.6008	1020.6008	[11.96499,57.70923]	[11.95852,57.70295]	11.684		
						618.8304933			4.424533333		

Figur H.1: BruteForce resultat scenario 4

H. Appendix H

OrderID	Förare	Order beställdes (s)	Order delades ut (s)	Order plockades upp (s)	Order levererades (s)	Leveranstid (s)	Från	Till	Algoritmtid (s)	Förare	Väg
0	Driver2	20	20	62.3002	525.6004	505.6004	[11.97522,57.6994]	[12.0002,57.71623]	0.834	Driver0	S10->S20->D10->D20->S22->D22
1	Driver4	45	45	154.3002	523.3002	478.3002	[11.98478,57.69753]	[11.98485,57.68894]	0.145	Driver1	S15->D15->S26->S24->S9->D24->D26->D9
2	Driver8	70	70	155.4002	346.6004	276.6004	[11.97796,57.692314]	[11.96601,57.68978]	0.148	Driver2	S0->D0
3	Driver6	120	120	465.7002	1173.7006	1053.7006	[12.01285,57.69969]	[11.97028,57.69584]	0.515	Driver3	S11->S21->D11->D21
4	Driver5	140	140	334.2002	1193.3002	1053.3002	[11.95252,57.69089]	[11.97512,57.70033]	0.124	Driver4	S1->D1->S7->D7
5	Driver7	180	180	281.8002	501.2002	321.2002	[11.97718,57.69922]	[11.96531,57.6982]	0.218	Driver5	S4->S8->D8->S18->D4->D18->S29->D29
6	Driver9	270	270	323.6002	579.2002	309.2002	[11.96688,57.69842]	[11.97042,57.69364]	0.203	Driver6	S3->S19->D19->D3
7	Driver4	320	320	719.8004	927.3006	607.3006	[11.98389,57.68247]	[11.99117,57.69078]	0.35	Driver7	S5->D5->S13->S14->S23->D13->D14->D23
8	Driver5	350	350	455.8002	759.2002	409.2002	[11.95133,57.6937]	[11.96665,57.69312]	0.203	Driver8	S2->D2->S17->S27->S28->D28->D17->D27
9	Driver1	390	390	1205.0006	1473.7012	1083.7012	[11.95687,57.69902]	[11.96189,57.69395]	0.182	Driver9	S6->D6->S12->D12->S16->D16->S25->D25
10	Driver0	420	420	666.0002	1080.5004	660.5004	[11.96566,57.70531]	[11.96452,57.6985]	0.165		
11	Driver3	430	430	698.1002	1386.3004	956.3004	[11.97047,57.70486]	[11.98331,57.70328]	0.218		
12	Driver9	450	450	690.2004	825.9006	375.9006	[11.97767,57.69401]	[11.97549,57.69819]	0.234		
13	Driver7	460	460	667.8002	1399.3006	939.3006	[11.95769,57.70455]	[11.97145,57.70136]	0.263		
14	Driver7	520	520	888.0002	1454.4008	934.4008	[11.96515,57.70867]	[11.97559,57.7033]	0.272		
15	Driver1	530	530	684.2002	792.5004	262.5004	[11.95787,57.70554]	[11.96077,57.70101]	0.322		
16	Driver9	540	540	1031.0002	1137.0004	597.0004	[11.97809,57.70402]	[11.98369,57.69938]	0.483		
17	Driver8	580	580	1034.9002	1717.301	1137.301	[11.97129,57.69651]	[11.98467,57.69867]	0.47		
18	Driver5	630	630	954.0004	1286.9004	656.9004	[11.95762,57.69494]	[11.9764,57.69993]	0.537		
19	Driver6	645	645	844.4002	1076.0004	431.0004	[11.97957,57.70005]	[11.96328,57.69942]	0.466		
20	Driver0	700	700	1006.6002	1212.5006	512.5006	[11.96291,57.69712]	[11.97166,57.70056]	0.374		
21	Driver3	740	740	1076.2002	1563.5006	823.5006	[11.96276,57.70268]	[11.97226,57.70638]	0.325		
22	Driver0	790	790	1481.9008	1599.301	809.301	[11.95122,57.69998]	[11.95013,57.69392]	0.416		
23	Driver7	850	850	1202.1004	1503.701	653.701	[11.96677,57.70583]	[11.98093,57.70309]	0.372		
24	Driver1	890	890	1188.5004	1335.1008	445.1008	[11.95625,57.69846]	[11.96554,57.69775]	0.37		
25	Driver9	940	940	1523.9006	1654.3008	714.3008	[11.95087,57.69761]	[11.93586,57.69958]	0.41		
26	Driver1	945	945	1034.8002	1426.901	481.901	[11.9492,57.69627]	[11.96171,57.69581]	0.369		
27	Driver8	990	990	1114.3004	1820.4012	830.4012	[11.96885,57.69793]	[11.97702,57.69515]	0.407		
28	Driver8	1020	1020	1400.2006	1654.6008	634.6008	[11.97142,57.70585]	[11.98545,57.70137]	0.362		
29	Driver5	1060	1060	1714.4006	2179.8008	1119.8008	[11.96499,57.70923]	[11.95852,57.70295]	0.49		
						669.14394			0.341566667		

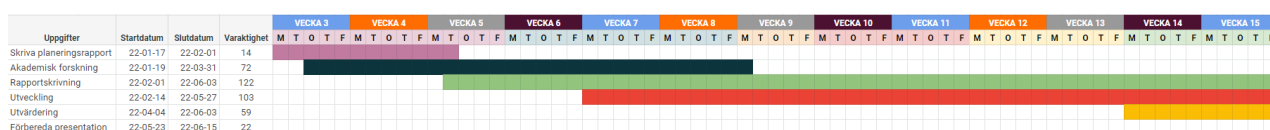
Figur H.2: Euklidisk approximation resultat scenario 4

OrderID	Förare	Order beställdes (s)	Order delades ut (s)	Order plockades upp (s)	Order levererades (s)	Leveranstid (s)	Från	Till	Algoritmtid (s)	Förare	Väg
0	Driver2	20	20	62.3002	525.6004	505.6004	[11.97522,57.6994]	[12.0002,57.71623]	1.325	Driver0	S10->D10->S20->S20->D20->S25->D25
1	Driver4	45	45	154.3002	523.3002	478.3002	[11.98478,57.69753]	[11.98485,57.68894]	0.35	Driver1	S15->D15->S22->S22->D22->S9->D9
2	Driver8	70	70	155.4002	346.6004	276.6004	[11.97796,57.692314]	[11.96601,57.68978]	0.176	Driver2	S0->D0
3	Driver6	120	120	465.7002	1185.0006	1065.0006	[12.01285,57.69969]	[11.97028,57.69584]	0.504	Driver3	S11->S21->S21->D11->D21->S28->D28
4	Driver5	140	140	334.2002	992.3002	852.3002	[11.95252,57.69089]	[11.97512,57.70033]	0.238	Driver4	S1->D1->S7->D7
5	Driver7	180	180	281.8002	501.2002	321.2002	[11.97718,57.69922]	[11.96531,57.6982]	0.566	Driver5	S4->S8->S18->S18->D4->D18->D8->S26->D26
6	Driver9	270	270	323.6002	579.2002	309.2002	[11.96688,57.69842]	[11.97042,57.69364]	0.422	Driver6	S3->S19->D19->D3->S29->D29
7	Driver4	320	320	719.8004	927.3006	607.3006	[11.98389,57.68247]	[11.99117,57.69078]	0.54	Driver7	S5->D5->S13->S14->S23->S23->D14->D23->D13
8	Driver5	350	350	455.8002	1311.501	961.501	[11.95133,57.6937]	[11.96665,57.69312]	0.223	Driver8	S2->D2->S17->S27->D27->D17
9	Driver1	390	390	1321.8016	1517.4018	1127.4018	[11.95687,57.69902]	[11.96189,57.69395]	0.801	Driver9	S6->D6->S12->S12->D12->S24->D24->S16->D16
10	Driver0	420	420	666.0002	983.0002	563.0002	[11.96566,57.70531]	[11.96452,57.6985]	0.383		
11	Driver3	430	430	698.1002	1389.4006	959.4006	[11.97047,57.70486]	[11.98331,57.70328]	0.435		
12	Driver9	450	450	690.2008	825.901	375.901	[11.97767,57.69401]	[11.97549,57.69819]	0.714		
13	Driver7	460	460	667.8002	1611.002	1151.002	[11.95769,57.70455]	[11.97145,57.70136]	0.827		
14	Driver7	520	520	888.0004	1392.301	872.301	[11.96515,57.70867]	[11.97559,57.7033]	0.737		
15	Driver1	530	530	684.2002	837.8002	307.8002	[11.95787,57.70554]	[11.96077,57.70101]	1.788		
16	Driver9	540	540	1469.3006	1575.3008	1035.3008	[11.97809,57.70402]	[11.98369,57.69938]	2.5		
17	Driver8	580	580	1028.3002	1413.9008	833.9008	[11.97129,57.69651]	[11.98467,57.69867]	2.535		
18	Driver5	630	630	758.8004	1085.9006	455.9006	[11.95762,57.69494]	[11.9764,57.69993]	2.787		
19	Driver6	645	645	844.4002	1087.3002	442.3002	[11.97957,57.70005]	[11.96328,57.69942]	1.67		
20	Driver0	700	700	1072.7008	1222.001	522.001	[11.96291,57.69712]	[11.97166,57.70056]	1.847		
21	Driver3	740	740	1079.3004	1566.601	826.601	[11.96276,57.70268]	[11.97226,57.70638]	2.266		
22	Driver1	790	790	1030.6008	1148.001	358.001	[11.95122,57.69998]	[11.95013,57.69392]	1.781		
23	Driver7	850	850	1202.1008	1441.6014	591.6014	[11.96677,57.70583]	[11.98093,57.70309]	1.888		
24	Driver9	890	890	1156.3002	1302.7004	412.7004	[11.95625,57.69846]	[11.96554,57.69775]	2.928		
25	Driver0	940	940	1483.6016	1614.0018	674.0018	[11.95087,57.69761]	[11.93586,57.69958]	1.961		
26	Driver5	945	945	1631.8014	1833.1016	888.1016	[11.9492,57.69627]	[11.96171,57.69581]	1.892		
27	Driver8	990	990	1107.7004	1222.1006	232.1006	[11.96885,57.69793]	[11.97702,57.69515]	1.938		
28	Driver3	1020	1020	1587.6016	1842.0018	822.0018	[11.97142,57.70585]	[11.98545,57.70137]	2.476		
29	Driver6	1060	1060	1615.201	2080.6012	1020.6012	[11.96499,57.70923]	[11.95852,57.70295]	2.646		
						661.6308267			1.371466667		

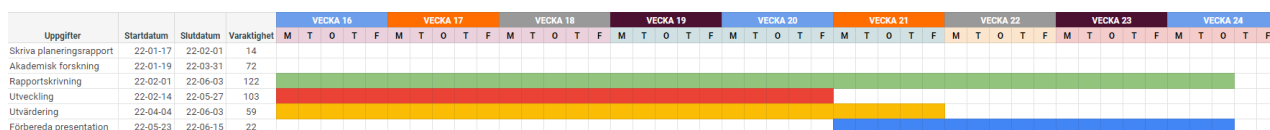
Figur H.3: Girig approximation resultat scenario 4

Appendix I

Appendix I innehåller 2 figurer av GANTT-schemat som projektet utgick ifrån. Figur I.1 innehåller de tretton första veckorna från vecka 3 till vecka 15. Figure I.2 innehåller de nio sista veckorna från vecka 16 till vecka 24.



Figur I.1: GANTT-schema för projektplan V.3 till V.15



Figur I.2: GANTT-schema för projektplan V.16 till V.24

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se



CHALMERS