Dial-a-Ride

Jørgensen, Rene Munk; Madsen, Oli B.G.

Publication date:
2003

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Jørgensen, R. M., & Madsen, O. B. G. (2003). Dial-a-Ride.

Dial-a-Ride

by

René Munk Jørgensen

Supervisor
Professor, Dr.techn. Oli B. G. Madsen

Submittet in partial fulfillment of
the requirements of the degree of

Doctor of Philosophy

Centre for Traffic and Transport
Technical University of Denmark

June 30, 2002

ISBN 87-91137-10-1

ISSN 1600-1575

Report 2004-1

Trykt af XXXXXXXXXX

xxx

xxx

Preface

Entitled “Dial-a-Ride” this thesis concerns the use of operations research methods when solving the practical Dial-a-Ride problem. The thesis was prepared by René Munk Jørgensen during the period March 1999 to June 2002, partly at the department of Informatics and Mathmatical Modelling (IMM), and partly at the Center for Traffic and Transport (CTT), both related to the Technical University of Denmark (DTU).

The Ph.D. was supervised by Professor Oli B. G. Madsen, and it is submitted in partial fulfillment of the requirements of the degree of Doctor of Philosophy at the Technical University of Denmark. The project was financed by the Technical University of Denmark.

Acknowledgements

First and foremost I would like to thank my supervisor Oli B. G. Madsen for excellent support and encouragement throughout the entire project. I would also like to thank my former colleagues from section of Operations Research at IMM, especially Jesper Larsen for beeing positive, friendly, and helpfull during the period in which we shared office space. As for the last half of the duration of this project, I would like to thank Brian Kallehauge with whom I shared the office. Brian helped making the change from IMM to CTT painless and in many counts our discussions have kept the project on track.

A special thanks goes to Centre de Recherche sur les Transports at Universié de Montréal in Canada, where Jean-Yves Potvin, Michel Gendreau, and François Guertin always had time to help and provide me with valuable input.

As for the practical part of this project, I would like to thank Palle at Jensens Turisttrafik for taking the time to describe the practical aspects of the Dial-a-Ride problem and to provide data and scenarios. Also I would like to thank Jesper Bislev, who helped develope InfoRoute as part of his Masters thesis. Professor David Ryan visited CTT during the last part of my Ph.D., and I would like to thank him for taking the time to be so generous with red ink.

Last but not least I would like to thank my family. My wife for supporting me when things were not always performing as expected, and for her endless patience with me during the phases of the project.

Kgs. Lyngby, June 2002

René Munk Jørgensen

Summary

The Dial-a-Ride problem is a Vehicle Routing problem in its most general form. The problem was formulated in the beginning of the 1970's and since then a large number of researchers have worked on developing efficient algorithms for use in automated planning of Dial-a-Ride transportation systems.

In a Dial-a-Ride transportation system, passengers request a trip between two stops with either a desired time of departure from the pickup stop or a desired time of arrival at the destination. The time windows at the stops are then calculated by the operator based on additional parameters such as maximum excess ride time and allowed deviation from desired service time. All vehicles start and end at a depot, which can be different for each vehicle. The fleet of vehicles is heterogeneous meaning the vehicles can have a different capacity of passenger seats, wheelchairs and beds.

The problem is then an extension of the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW) with the additional constraints that a pickup stop must precede a delivery stop for a passenger, and that both a pickup stop and a delivery stop related to the same passenger must be assigned to the same vehicle. The objective is to minimize the cost of operation similar to the objective for the CVRPTW, with the extension that the level of service to the passengers is considered.

Several algorithms for solving the Dial-a-Ride problem have been proposed in the litterature. Because of the complexity of the Dial-a-Ride problem, most methods described in the litterature are based on various insertion techniques often coupled with some sort of clustering of stops, but since 1980 some experiments have been performed with dynamic programming, column generation and set partitioning. In recent years meta heuristics have also been applied to the problem. The metod used in this thesis is based on a clustering first insertion second technique developed at CRT in Canada in the mid 1980's.

The algorithm is extended to include constraints imposed by a practical Dial-a-Ride problem at a Danish transportation operator. These constraints are mainly related to a generalization of capacity to include substitution of seats to wheelchairs and beds, but also extensions to the objective to give a better view of passenger comfort are included.

To enable practical use of the algorithm, it is incorporated in the developed user interface and linked with a digitized road network. The user interface is developed as generically as possible making it reuseable in all sorts of Vehicle Routing problems. The foundation of the user interface is a database for storing customers, requests, routes etc.

The digitized road map is included in the developed software by use of MapX, which is a program developed and sold by MapInfo Corporation. MapX handles the visualization of the map layers, selections, and labels. The road network is read from MapX into a different network structure in memory to allow for a more intelligent calculation of shortest path between stops and vehicle positions.

Unfortunately data from actual Dial-a-Ride transportation systems is not readily available, so tests are based on data generated in this project. While the algorithm performs as expected, other results such as the stable performance of the different modules in the developed software and the developement of a practical setting for solving the Dial-a-Ride problem are equally important.

Resumé in Danish

Dial-a-Ride Problemet (DARP) er et Vehicle Routing Problem i dets mest generelle udformning. DARP blev først formuleret i begyndelsen af 1970'erne, og siden har der været en anseelig forskning i at udvikle effektive algoritmer, som kan assistere ved en automatisk planlægning i Dial-a-Ride transportsystemer.

I denne type af transportsystemer bestiller passagerer en tur mellem to stoppesteder, som i nogle tilfælde er fastlagte steder og i andre tilfælde er selvvalgte adresser. I forbindelse med valg af destinationer specificerer passageren ligeledes enten et ønsket tidligste afhentningstidspunkt eller seneste afleveringstidspunkt. Tidsvinduerne på de to stoppesteder beregnes herefter af operatøren ved hjælp af eksternt fastsatte parametre så som maksimal unødvendig rejsetid og tilladt afvigelse fra ønsket afhentnings- eller afleveringstidspunkt. Maksimal unødvendig rejsetid er den faktiske rejsetid fratrukket den kortest mulige rejsetid.

Den benyttede vognpark er heterogen, og det antages, at alle vogne starter og slutter på et eller flere depoter. Med heterogen vognpark menes vogne, som kan have forskellig kapacitet og indretning i forbindelse med sæder, kørestole og senge.

DARP er således en udvidelse af the Capacitated Vehicle Routing Problem med Time Windows (CVRPTW) med nogle ekstra begrænsninger. Disse begrænsninger siger, at et afhentningsstop skal besøges af en vogn før et afleveringsstop, samt at to stop relateret til samme passager skal besøges af samme vogn. Målet i planlægningen er at minimere de samlede kørselsomkostninger som i CVRPTW med den udvidelse, at der skal tages hensyn til serviceniveauet for passagererne, som maksimeres.

Den eksisterende litteratur inden for området præsenterer adskillige algoritmer, som kan løse Dial-a-Ride problemet. På grund af kompleksiteten og størrelsen af praktiske Dial-a-Ride problemer, er de fleste algoritmer baseret på forskellige indsætningsteknikker, som i mange tilfælde er suppleret af forskellige former for gruppering af stop. Siden 1980 er der dog også forsket en del med metoder inden for dynamisk programmering, søjlegenerering og "set partitioning". I de senere år er forskellige former for meta heuristikker også søgt benyttet. I dette projekt er der taget udgangspunkt i en metode udviklet af CRT i Canada. Metoden grupperer først stoppene, hvorefter de indsættes i ruter.

Metoden fra Canada er udvidet til at tage højde for forskellige faktorer fra et virkeligt problem hos en dansk transportør. Disse faktorer har primært relation til kapaciteten af vognene, hvor det er muligt at omlægge nogle sæder til kørestole eller senge men ikke altid den anden vej rundt. Herudover er målfunktionen ændret til at give et mere detaljeret billede af serviceniveauet for passagererne.

For at undersøge metodens brugbarhed i praksis, er den koblet med en grafisk brugergrænseflade, som er baseret på en meget generel database forberedt til lagring af data relateret til alle kendte typer af rutelægningsproblemer. Hele den grafiske brugergrænseflade er opbygget modulært, hvilket giver mulighed for genbrug ved løsning af andre typer af problemer. Til at visualisere data på vejkort er programmet MapX benyttet, men for at sikre en hurtig beregning af korteste veje mellem stop og biler, læses vejdata ind i computerens hukommelse i en speciel netværksstruktur. Herefter er et korteste vej modul tilknyttet grænsefladen.

Det har desværre ikke været muligt at benytte data fra eksisterende Dial-a-Ride transportsystemer, idet denne data ikke er umiddelbart tilgængelig. I stedet er der genereret et realistisk datasæt, hvorpå metoden er testet. Metoden fungerer som forventet, hvilket dog kun er en del af det samlede resultat, idet sammenkoblingen af de forskellige dele af den udviklede software i en grafisk brugergrænseflade med de udfordringer som opstår herved, er en lige så væsentlig del af projektet. Resultatet er en stabil og velfungerende softwarepakke, som dog stadig mangler en del i at være fuldt funktionsdygting i kommerciel sammenhæng.

Contents

Chapter 1

Terminology and abbreviations

The following chapter will give an introduction to the terminology used throughout this thesis. The definitions are for the most part reflections of the author and thus designed to give the reader the necessary background information prior to reading the thesis. The hope of the author is to make the thesis accessible to anyone with no prior experience with operations research.

1.1 Definitions

This section will list the terminology needed to define the Dial-a-Ride Problem. Since the methods described later in this thesis are used to solve problems in public transportation, an introduction to different types of public transportation will presented secondly.

1.1.1 Demand responsive transportation

- Transport
  Noun: Something that serves as a means of transportation.
  Verb: Move goods or people from one place to another.

- Transportation
  The act of performing a transport.

- Vehicle
  A conveyance (train, bus, taxi etc.) that transports people or goods.

- Depot
  The origin and final destination of a vehicle, or an intermediate depot for refuelling or reloading.

- Stop
  Where transport vehicles load or unload passengers or goods.

- Trip
  A collection of stops serviced by the same vehicle. The vehicle starts and ends a trip empty, but is never empty while performing the trip. A trip can also be the empty driving between two trips.

- Route
  A collection of trips.

- Scheduled route
  A route which will be serviced at a scheduled time. Scheduled routes are planned far in advance of the actual service.

- Tele bus
  A bus without scheduled routes to service. In order to be transported by a tele bus, customers must contact the operator and place a request for transportation.

- Coverage
  The frequency of which vehicles service a scheduled route coupled with the geographical distance between scheduled routes, and the size of the vehicles servicing the routes.

- System
  The term system is used to define the borders of an environment. Everything within a system has similar characteristics, and a system can enclose multiple smaller systems or be part of a larger system.

- Transportation systems
  An enclosed system with various transports of goods or people. The system can be just one transport or an interconnected series of transports between stops with a heterogeneous fleet of transport vehicles originating from various depots.

- Regular transportation systems
  All transports within the system are performed according to a predefined schedule. The systems capacity is fixed in an operational time horizon. Changes in the system are performed at the strategic level often with massive impacts on the overall economy of the system.

- Demand responsive transportation
  Transports are performed according to demand within a short time horizon. Capacity is flexible within the operational time horizon with minimal overall change in cost. Demand responsive transportation can be combined with a regular transportation system to improve the overall flexibility of the system.

- Demand responsive transportation system
  A system where the main characteristic is demand responsiveness.

- Pickup-and-Delivery transportation
  The term Pickup-and-Delivery is used when dealing with the transportation of goods. This type of transportation is characterized by the vehicle's ability to both pick up and deliver goods along the same route without visiting a depot in between. The goods do not necessarely have to be both picked up and delivered on the same route.

- Dial-a-Ride transportation
  Dial-a-Ride transportation is a generalization of Pickup-and-Delivery transportation. The objects being tranported here are people, which adds extra restrictions related to customer inconvienience. In Dial-a-Ride transportation people have to be picked up and delivered along the same route. Our objective is partly to minimize the travel time and waiting time for customers using this type of transportation, and partly to minimize the overall cost of transportation.

- Dial-a-Ride Problem
  The problem is to construct algorithms to assist in automated planning for a Dial-a-Ride transportation system.

The scale of the demand responsive transportation system is often limited to local transportation systems which are then connected to a regional transportation system. This is due to the fact that variations in demand usually decreases proportionally to the scale of the system, or one could say that estimates of demand are increasingly accurate with increasing system size.

1.1.2 Public transportation

When considering a public transportation system it is necessary to differentiate between three types of transport:

- Regular public transports
  These transports involve the moving of the main body of the population between stops. This is often implemented as a regular transportation system providing a certain degree of mobility within the community.

- Specialized public transports
  This includes all transports that can not be served by the regular transportation system because of special needs of the people being transported. Examples are:
  - transportation of senior citizens
  - transportation of permanently handicapped
  - transportation of temporarely disabled.

- Delivery of goods
  In combination with the movement of people within the public transportation system, there is also a need to transport various goods to the citizens of a community. An example could be food for senior citizens, books from the library to disabled citizens etc.

1.1.3 Level of service

A very important part of transportation systems is the level of service they provide to customers. It is very complicated to meassure the level of service, since it is a mix of many different physical and psycological factors. To simplify this, we will devide level of service into the following two groups where focus is on transportation of people:

- Strategic level of service

  - type of transportation system
  - frequency of scheduled routes
  - geographical proximity of scheduled routes
  - physical sorroundings at fixed stops

- Operational level of service
  - average excess transportation time for passengers
  - average deviation from expected time at stops
  - waiting time at stops
  - waiting time in idle vehicles

Strategic measures often involves comparatively large investments or at least major changes in the existing transportation system. Notice also, that the service parameters mentioned in the strategic group for the most part involved looking at the previously mentioned coverage. The better coverage of the chosen transportation system, the more service the system offers to its passengers.

In this thesis, we will focus mainly on the operational level of service. In a regular strictly fixed schedule transportation system, the measures on the operational level also becomes strategic in the sence that they are determined by the coverage of the system. However, in a demand responsive transportation system, it is possible to control the operational level of service parameters directly in the dynamic planning process.

1.2 Table of abbreviations

Abbrev.	Description
DAR	Dial-a-Ride
DARP	Dial-a-Ride Problem
DTD	Door-to-Door
EU	European Union
TW	Time Window
PDP	Pickup-and-Delivery Problem
MFC	Microsoft Foundation Classes
DSP	Data Storing and Processing
GUI	Graphical User Interface

Table 1.1: Table of abbreviations

Chapter 2

Introduction

As still more and more people and goods are being transported around the world, the present global development is going to emphasize demands on transportation in the future. An efficient and ever evolving transportation sector is crucial for the competitiveness of a society, just as it will be a prerequisite for growth and dynamic adaption to future demands.

One of the key elements in the necessary development of transportation systems, is an advanced IT-structure including software based systems for decision support. As the techniques for modelling and simulating transportation systems improve together with the performance of modern computers, the use of automatic or semi-automatic decision support software systems will play an important role in future transportation planning.

In the European economy the transport sector employs around 6.3 million persons, which is 4.1% of all persons employed, amounting to 4% of the gross national product (GNP). The transportation equipment industry employs around 2.6 million people. The overall cost of transportation within the EU when including private transportation is estimated to account for 17% of the total GNP of the union[1]. In Denmark transportation accounted for 13% of the gross national product in 1981 and 15% in 1994[2], so it seems natural to examine this area when looking to reduce costs.

[1]"EU Energy and Transport in figures 2001", European Commission, Directorate-General for Energy and Transport, 2001.

[2]"Trafikredegørelse 1997", Trafikministeriet, 1998.

Transportation is also one of the major contributors to the increasing problems concerning the environment. In 1999 transportation within the European community accounted for around 28% of the total CO_2 emission[1]. By improving the efficiency of transportation, the total distance travelled is usually decreased, which again has a very positive environmental side effect.

This thesis will concentrate only on public transportation, which in recent years has received increasing focus on modernisation both politically as well as in the media. Public transportation includes transportation of disabled and other specialized public transports as well as ordinary public transportation. When looking at ordinary public transportation, one can see that modernisation is especially important in sparsely populated areas.

In Denmark the yearly budget for public transportation is around 4 billion kroner (est. US$ 500 million). About half of these funds are used to run the regular public transportation system, and the other half constitutes the cost of the specialized public transports. In short there is a comparatively small set of people who use half the total budget for public transportation.

As the pressing need to cut costs conflicts with the need for better service within the sector of public transportation, operators are seeking better ways of structuring public transportation. There are many examples in Denmark, especially in rural areas, where large buses drive entire routes with the driver as the only passenger. To solve this problem, coverage is reduced by merging routes geographically and decreasing frequency. However reducing coverage does not seem to be the right way to reduce costs, since this only adds to the increasing amount of private transportation by car. Also by reducing coverage, a small and relatively fragile part of the population is held at a comparatively large disadvantage, since they do not have access to private vechicles.

Keeping the above in mind, we can establish a set of goals for the public transportation system. One is to improve the level of service by extending the existing coverage or by improving over-

[1]"EU Energy and Transport in figures 2001", European Commission, Directorate-General for Energy and Transport, 2001.

all coverage, and two is to minimize costs. These objectives are of course conflicting, but fortunately the existing system has room for improvement, meaning that results have shown that better transportation systems can be developed without increasing costs and sometimes even decreasing them.

There are several ways to improve the public transportation system, and many of them are being or have been tested in recent years. One improvement can be found in various types of outsourcing. One type of outsourcing that looks to be especially profitable involves funding on the basis of passenger-kilometer. This funding provides the private contractor with an incentive to increase passenger numbers, thus fullfilling some of the goals of the public transportation system.

This thesis will focus on the planning of demand responsive Dial-a-Ride transportation systems, where regular buses are substituted with tele buses to drive not only fixed routes but also variable routes on demand. Furthermore the flexibility of the Dial-a-Ride system can be a way to merge the specialized public transports with the regular public transports. The hope is to both reduce cost and improve overall coverage of the public transportation system.

2.1 Example of a DAR transportation system

The simplest form of DAR tranportation can be found in a taxi system. Here there is one pickup and one delivery on the same trip. There might be special cases where there is more than one delivery (e.g. when collecting a group of people with slightly different destinations), but for the most part the problem consists of allocating a vehicle to pickup a person or a group of persons at one location, and then take the shortest or fastest route to a single destination. After performing the transport, the vehicle is driven empty to the pickup point of the next request.

In taxi systems, capacity is of no concern when looking at the individual vehicles, but the overall capacity of the system (e.g. the number of taxis) can be restrictive. From the operator's point of view, the problem consists of minimizing the number of needed taxis and the distance driven with empty taxis. However this con-

flicts with the need to minimize customer inconvenience, since in taxi systems this consists only of the time a customer has to wait for a taxi to become available and arrive at the pickup location. So, the need to minimize customer inconvenience conflicts with the need to minimize the number of taxis. It becomes even more complicated when considering the fact that minimizing customer inconvenience also means that sometimes it is necessary to drive a taxi empty a comparatively long distance to serve the next request.

The above description of the taxi system serves to give an introduction to the complexity involved when seeking to solve DARP. Later in the thesis it will become clear that the taxi system is in fact one of the simplest recognized DAR transportation systems.

2.2 Overview

The following chapters are introduced below with a short description of what is discussed in each chapter.

- Chapter 3 The problem and main focus of this thesis will be stated. The chapter is divided into two sections addressing respectively the scientific goals and the practical issues.

- Chapter 4 This chapter gives a description of the practical problem at a bus operating company in Denmark and its relation to other recognized problems within the area of vehicle routing.

- Chapter 5 Some research has been done on the Dial-a-Ride Problem around the world. This chapter will give an extensive introduction to this research as well as provide the motivation for choosing the solution method used herein. The last part of this chapter will focus on the research and practical work performed in Denmark.

- Chapter 6 Using a mathematical formulation as a discussion approach, the various aspects of dial-a-ride problems are discussed in detail. This chapter will also address formulations of objectives and constraints not previously seen in the existing litterature.

- Chapter 7 This chapter gives a brief introduction to two commercially available software packages developed to assist in automated planning for a Dial-a-Ride transportation system.

- Chapter 8 To obtain distances on a real road network, a Shortest Path algorithm has been developed. This algorithm uses an alternative network structure to account for restrictions on the road network, and to obtain distances in short computational time.

- Chapter 9 This chapter discusses a project consisting of preparing a multi purpose graphical user interface for representing data and solutions regarding vehicle routing problems. This chapter will introduce the functionality and implementation aspects of the interface.

- **Chapter 10 This chapter presents a description of the enhanced mini cluster algorithm (McCluster) used to solve the Dial-a-Ride problem will be presented here. To some extent the chapter will focus on the handling of some of the constraints imposed by pratical applications of Dial-a-Ride.**

- **Chapter 11 McCluster has been run on various constructed and real datasets. Results will be discussed here with focus on aspects of future practical implementation.**

- **Chapter 12 Improving solutions obtained with McCluster can prove vital in acommodating as many customers as possible in a Dial-a-Ride system. Different intelligent approaches are suggested here.**

- **Chapter 13 Goals and focus points stated in chapter 4 will be addressed in this last chapter.**

Chapter 3

Thesis objectives

This chapter is divided into two sections concerned with respectively the scientific and the practical goals of the thesis. The sections will define the framework of this thesis.

3.1 Scientific contents

Based on realistic practical problems, the main focus of this thesis is to develop a new or improve an existing algorithm aimed at solving the dynamic Dial-a-Ride Problem. The algorithm will be designed for solving real-time problems and be efficiently implementet to enable practical use. The theoretical background of Dial-a-Ride Problems is described and existing literature examined.

The thesis will use clustering and insertion heuristics as a starting point since these heuristics have proven to perform well in practical environments with short run-times and reasonable results. When using such heuristics, the implementation will aim to incorporate as many practical constraints as possible to ensure proper modelling of real life problems.

When solving Dial-a-Ride Problems the distances between customers and depots must be known. To obtain this information, a shortest path algorithm will be implemented for use on real road digitized networks. This algorithm will show to perform adequately according to the practical use of the Dial-a-Ride algorithm, meaning that the run-time of the shortest path algorithm will be low enough to ensure a low overall run-time.

To provide a theoretical background to developed algorithms, a mathematical formulation of the Dial-a-Ride Problem is given in the thesis. Parts of the model will be derived directly from the existing literature, whereas other parts will be a new contribution to the area. The mathematical formulation will be thouroughly described, and strong points and limitations to the formulation will be mentioned.

A major part of the thesis will be to give an extensive overview of the existing literature concerning Dial-a-Ride Problems. This overview will focus on solution methods, results, and practical usability.

3.2 Practical contents

Besides focusing on the scientific aspects mentioned in the previous section, there are several aspects that needs to be examined in order to ensure the future practical use of the developed planning method.

First a user interface is developed to enable on screen visualization of results on real road networks. This user interface will enable users to do the following:

- Use the underlying road network to lookup addresses and perform shortest path calculations.
- Input and correct customer and request data.
- Run the algorithm developed and described in this thesis.
- Visualize real-time data such as vehicle positions.

A database for storing and manipulating requests within the user-interface will be constructed. This database is able to import and export data from various formats, just as the structure should fit all known types of Dial-a-Ride problems and the practical information needed in daily operations.

The overall implementation will be object oriented and implemented in C++ to follow the trends in modern computing and to improve readability of the source code. This will also ensure the possibility of incorporating source code in COM (common object modules), and thus make the implementation somewhat independent of Operating System.

Chapter 4

Problem formulation

Based on the definition of DARP stated in 1.1.1, this chapter will first give an example of a typical and very complex DAR transportation system, as it is seen at the Danish bus operator Jensens Turisttrafik. This will be followed by an introduction to static vs. dynamic DARP and the relation to other recognized problems within the area of Vehicle Routing. The chapter will conclude with a general and somewhat social discussion of parameter setting.

4.1 Jensens Turisttrafik

A description of the Dial-a-Ride problem handled by Jensens Turisttrafik (JT) offers a good overview of the complexity of real world transportation systems. At the same time the reasonable size of the problem at JT makes it possible to get into all aspects of practical transportation needs.

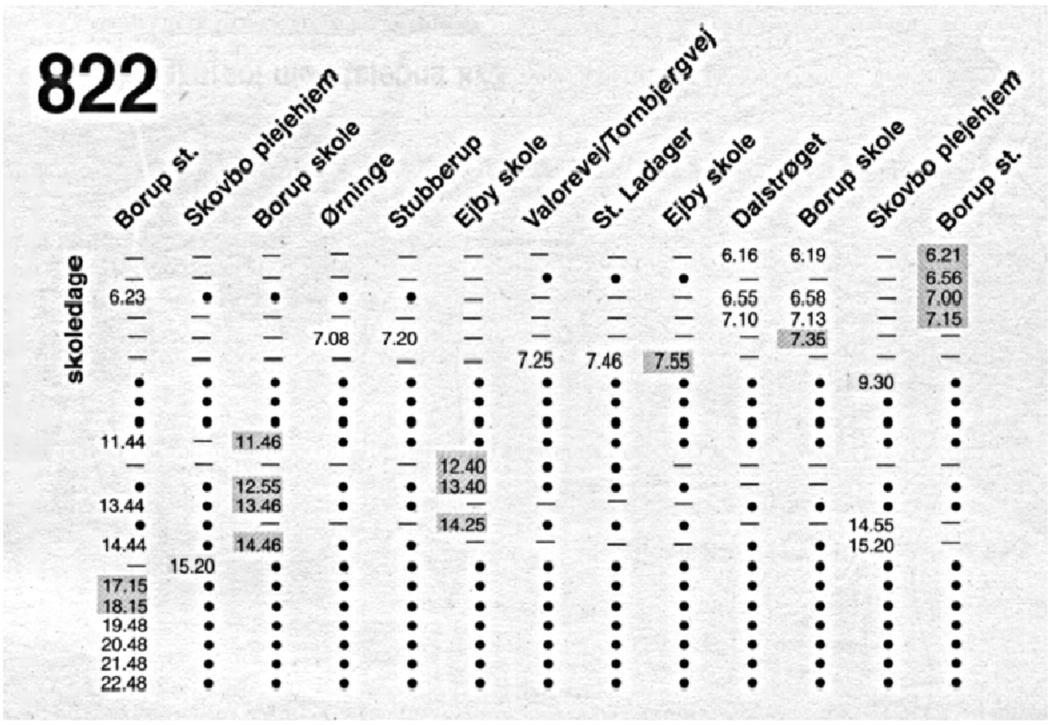
In the municipality of Skovbo in Denmark, JT operates three flexible bus routes. A typical schedule of one of these routes is found in figure 4.1, showing a mix of regular, DAR, and prioritized stops. The schedule is most easily explained by a summary of the various transportation jobs performed by the bus on a given day. The items listed are in chronological order and indicate the type of transports given priority at the time of the day.

- Morning
  - Commuters to the central railway station
  - Schoolchildren going to school

  - **Elderly going from home to activation centers**
  - **Dial-a-Ride transportation**

- **Midday**
  - **Schoolchildren going home from school**
  - **Elderly returning home**

- **Evening**
  - **Commuters from the central railway station**
  - **Dial-a-Ride transportation**



822

skoledage

Borup st.	Skovbo plejehjem	Borup skole	Ørninge	Stubberup	Ejby skole	Valorevej/Tornbjergvej	St. Ladager	Ejby skole	Dalstrøget	Borup skole	Skovbo plejehjem	Borup st.
—	—	—	—	—	—	—	—	—	6.16	6.19	—	6.21
—	—	—	—	—	—	●	●	●	—	—	—	6.56
6.23	●	●	●	●	—	—	—	—	6.55	6.58	—	7.00
—	—	—	—	—	—	—	—	—	7.10	7.13	—	7.15
—	—	—	7.08	7.20	—	—	—	—	—	7.35	—	—
—	—	—	—	—	—	7.25	7.46	7.55	—	—	—	—
●	●	●	●	●	●	●	●	●	●	●	9.30	●
●	●	●	●	●	●	●	●	●	●	●	●	●
●	●	●	●	●	●	●	●	●	●	●	●	●
11.44	—	11.46	●	●	●	●	●	●	●	●	●	●
—	—	—	—	—	12.40	●	●	—	—	—	—	—
●	●	12.55	●	●	13.40	●	●	●	—	—	●	●
13.44	●	13.46	●	●	—	—	—	—	●	●	●	●
●	●	—	—	—	14.25	●	●	●	—	—	14.55	—
14.44	●	14.46	●	●	—	—	—	—	●	●	15.20	—
—	15.20	●	●	●	●	●	●	●	●	●	●	●
17.15	●	●	●	●	●	●	●	●	●	●	●	●
18.15	●	●	●	●	●	●	●	●	●	●	●	●
19.48	●	●	●	●	●	●	●	●	●	●	●	●
20.48	●	●	●	●	●	●	●	●	●	●	●	●
21.48	●	●	●	●	●	●	●	●	●	●	●	●
22.48	●	●	●	●	●	●	●	●	●	●	●	●

— Bus does not stop **● Request a stop** **▨ Specialized transportransportation has priority**

Figure 4.1: A schedule for a bus route at Jensens Turisttrafik.

DAR transportation is performed all day as seen i figure 4.1 but when capacity is exceeded, then the group of passengers with priority is served before the DAR passengers. To simplify the planning process, JT recieves a weekly list of schoolchildren and

the elderly requirering transportation that week. Also a large part of the commuters are known in advance. All requests are listed in a spreadsheet which is printed and given to the drives at the beginning of their workday. When changes in the driving schedule occurs, the planner calls the bus driver with the information.

The total number of passenger trips (requests) on the three routes is about 300 per day, of which half is schoolchildren and the elderly. A large part of the remaining requests are known in advance. Although the DAR transportation is mainly based on regular stops for pick up and delivery, some passengers are serviced Door-to-Door, which is often the case with elderly and young schoolchildren. Two of the busses have a capacity of 20 seats of which 10 can be rearraged to hold 5 wheelchairs. The third bus has a capacity of 17 seats with room for 3 wheelchairs.

In the evening only one bus services the entire municipality, and the driver is then responsible for the planning. Instead of phoning the call center (the planner), passengers just call the driver to place a request. In the evening only a very few requests are known in advance, but the number of passengers is also very limited. When time permits, or when the driver estimates it to be necessary (for example a young child living far from a regular stop having to walk home at night) Door-to-Door service is included in the DAR transportation.

The planner at JT is familiar with the area and so with his combined years of experience and special knowledge concerning the different passengers, it is unlikely that an automated planning process can compete with the manually obtained results. However, JT has a problem when the regular planner is on vacation or ill, since it is difficult for other employees to take over the planning process. Another problem is bidding for similar transportation systems in areas where the planner has little experience, since the expected cost of transportation is uncertain.

The problem on the operational level at JT does not concern a minimization of the number of vehicles, but rather an assignment of requests that coincides with the established somewhat fixed routes. At the same time it is reasonable to conclude that an objective is to maximize availability, meaning to provide as much space for passengers in the vehicles as possible in order to maximize the accomodation of “new” requests. Maximizing availability is also the objective during the evening when a regular DAR transportation system with only one vehicle services the entire area. As the problem at JT seems to include almost all of the alterations in DAR transportation system, it is chosen as a reference for the DAR algorithm to be used in the automated planning software.

4.2 Static vs. dynamic Dial-a-Ride

As with all other Vehicle Routing Problems there can be both a static and a dynamic version of DARP. In this sense static or dynamic refers to the amount of information that is known in advance. In the case where all information is known in advance it is characterized as a static problem, but when only a part or no information is available beforehand but becomes available as operations advance, it is a dynamic problem.

Static problems are usually found when modelling on a strategic or tactical planning level. In these situations data on the operational level is often constructed on the basis of historical information, or estimated according to expectations. The model is then solved to give an overview of possibilities within different scenarios.

In principle a dynamic problem can be formulated as a discreet series of static problems, where a new problem is defined each time data changes. However solving a static model many times is not always possible or efficient, so different approaches have to be used when dealing with dynamic problems. When looking to solve realistic and detailed models, where computational time is critical, it is often possible to solve the static problem first, and then develop réoptimization algorithms to apply when data changes.

In the case of DARP it will be shown that it is not realistic to solve the static case to optimum. Many transportation systems are highly dynamic on the operational level, which makes it inefficient to solve the static problem over and over again. To explain the different aspects of DARP, a discussion based on the mathematical formulation of the static DARP will be presented in chapter 6, but as described in chapter 10 the algorithm used later will solve the static problem first, and then alter the solution when data changes.

4.3 Related problems

As mentioned in chapter 6 the mathematical formulation of DARP is based partially on that of PDP (also called the Vehicle Routing Problem with Pickup and Delivery - VRPPD). DARP is a generalization of PDP taking into account customer satisfaction by adding additional factors to the objective function. These additions can be formulated in various ways. In chapter 6 we will see how total customer travel time and waiting time in an idle vehicle can be addressed in the objective function, but other formulations of customer satisfaction, deviation from desired service time, for example, can be used.

PDP is again a generalization of VRPTW (Vehicle Routing Problem with Time Windows) where the VRPTW is the special case of the PDP with either the origins or the destinations located in a common depot. VRPTW is itself the extension of the CVRP (Capacitated VRP) where the service at each customer must happen at a fixed time interval in VRPTW. Using the same arguments as with DARP, the VRPTW is NP-hard. Even finding a feasibly solution to the problem with a fixed fleet size is itself NP-complete as mentioned in Toth & Vigo[1].

As the above mentioned problems are all generalizations of the TSP, this often leads to solution methods, in which the formulation uses the TSP as a subproblem. The TSP is a description of a transportation system where a travelling salesperson has to visit a number of customers once and only once, travelling the shortest possible distance. Very often the subproblem TSPs are small, which makes it possible to solve these problems to optimality or at least in very short computational time with a heuristic.

Although not directly related, the SPP (Shortest Path Problem) deserves some attention in this section also. With or without Time Windows or Capacity constraints, the various VRPs all depend on knowing the distances between customers and depots in order to find optimal or near optimal solutions. In static problems these distances can be calculated once and placed in a distance matrix for future lookup, but in the case of dynamic problems taking into account vehicle positions, the rapid solution to the SPP can be crucial to obtain results in short computational time. However SPP will be discussed in more detail in chapter 8.

4.4 Parameter setting

The following is a general discussion concerning the setting of parameters to control the objective when solving DARP. As these parameters have impact on objectives such as level of service and overall cost within the public transportation system, this will mostly be a social discussion. One important consideration when introducing DAR transportation systems is the existing goodwill among potential passengers.

First of all it is easy to conclude that the lower the existing service the more goodwill among the potentional passengers towards a new system. The passengers are the essential part of a public transportation system, which does not always show in the present systems. Coverage and scheduled routes are not always consistent with the actual transportation needs, a situation which, fortunately is changing. Because of the increasing cost of running public transportation, there is now a need to rationalize the sector.

As the need to cut costs conflicts with the need to increase the service in the rural areas to ensure the mobility of the population, there are a few questions that have not been answered. The most important is concerning the level of service. As it is now, the handicapped, seniors, and patients receive a much higher level of service than the rest of the population. Notice that this statement is based solely on the definition on level of service as stated in section 1.1.3.

Results from existing DAR transportation systems have shown that the disabled, seniors etc. now had to share vehicles with other passengers. This raises the question: Should the level of service for the disabled be higher than for the rest of the population? Of course it is, but only when necessary. Most disabilities do not exclude the possibility of sharing a minibus with others. It is always hard to give up privileges, but somehow the level of service should be close to the same for everybody.

The goal for public transportation is not to give the highest possible level of service to people, since that would mean giving everybody access to a taxi or a car of their choice. The goal is however to give a level of service that will ensure that everybody can travel from and to anywhere within the country in reasonable comfort within a reasonal traveltime for a reasonable price. Comparing the fares of the public transportation system with the cost of owning and driving a car, determines the necessary level of comfort and excess travel time in the public transportation system.

Now why spend time discussing the social aspects in a report concerning the operational research view on the planning proces? The reason is that it is very important to set the goals before starting an automated proces, since the outcome is mainly dependent on two parameters:

- level of service
- cost of operation.

Most of the projects mentioned later in section 5.2 state that the level of service is to be improved within the original budget. The question is then why all projects start by defining the level of service. By simulation it would be possible to actually find the best possible level of service given the maximum cost.

Chapter 5

Previous work

This chapter will aim to give a general understanding of where, when, and what work has been carried out within the topic Dial-a-Ride throughout the years. The academic interest for DARP started in the late 1960s mainly with small localized manual planning. However this chapter will concentrate on work performed from 1971 forward, since this is the beginning of the documented research and initial results. The chapter will end with a description of practical projects carried out in Denmark in recent years.

5.1 Paper survey

The DARP was first examined by Wilson et al.[57] and Zobrak[59] in 1971. Wilson et al. started the development of real-time algorithms designed specificly for DARP, and many later concepts such as sequential insertion of customers has been derived from this initial work. Zobrak's work which ran simultaneous with Wilson et al. focused mainly on the design and impelementation of DARP. In generel, developments on DARP can be divided into two classifications. The first works with the development of efficient algorithms, which is the main focus of this thesis. The second consists of work performed in designing and evaluating DAR Systems with regards to level of service, energy cost etc.

Throughout most of the 1970's almost all work was carried out within the area of simulating, implementing and evaluating DAR Systems. MIT developed the first computer controlled Dial-a-Ride system in 1972[40]. Here a heuristic insertion method was used to sequencially insert customers in routes by minimizing the inconvienience to existing customers and maximizing service to the new customer. The problem solved was one of immediate request DARP, which means that a customer calls the dispatcher with a request for immediate transportation between two destinations. The dispatcher then informs the customer of estimated pickup and delivery times.

Wilson[54], in his overview of developments until 1972, refers to the MIT method. Wilsons main focus is to evaluate the future potential role of computerized planning in Dial-a-Ride systems in contrast to the widespread manual planning. He proposes a Provisional Route Assignment Algorithm (PRAA). Wilson states that the overall objective of the DAR algorithms should be: "To provide lowest mean service times within fixed costs and guaranteed bounds on the worst service times." In 1972 however the amount of computational power available was very limited, so Wilson also mentions that the necessary detail level of the algorithms, in order for them to compete with an adequate manual dispatcher, makes it very hard to solve computationally. PRAA consists of 3 components:

- 1 - insertion of new requests into the provisional route of the best vehicle
- 2 - transfer of requests between routes to improve overall planning
- 3 - redistribution of unallocated vehicles to improve service to future demands.

Wilson mainly addresses the first of the three components where he sets the framework of a greedy insertion heuristic. Wilson concludes that the computerized systems can handle a much larger capacity than the manually planned systems but with fewer details in the planning process. He concludes that it might be efficient to establish computer aided planning systems to combine the best of the two worlds.

Unfortunately Wilson does not specify the exact nature of the DARP he is solving with PRAA. Keeping in line with the previous work at MIT, it does however seem likely that Wilson is again solving the immediate request DARP. The introduction of PRAA by Wilson is very interesting since the basic structure (greedy insertion and post optimization by swapping) of many recent developments can be traced back to PRAA. The third component, that of redistributing unallocated vehicles according to expected future demand has unfortunately not received the same attention as the insertion and swapping components.

In 1973 Slevin & Cooper[46] report experiences with a DAR System in Abingdon, England. Here a minibus was inserted to traverse three fixed schedule routes as a supplement to the existing bus system. Requests could then be met at any point along those routes. As with previously mentioned literature, the article does not specifically state the exact operational nature of the DAR transportation system addressed.

The most important conclusion in this paper is that DAR systems should only be introduced where they can greatly improve the level of service to customers. It is suggested that customers seem to be reluctant to accept a novel system if no comparatively large improvements over the old system is apparent.

The results from the Abingdon experiment are not completely relevant in 2002, because only about half the population in the area of Abingdon had access to a phone in 1970, which obviously limited the number of potential passengers for the DAR transportation. One thing that has not changed though is the fact that the walking distance to the nearest pickup point or bus stop is very influential on the number of passengers using the system.

In 1974 Hobeika[24] simulated a DAR Bus System in Indiana. Again in 1975 Fels[16] estimated the energy costs of DAR Systems compared to other transportation systems. In short the conclusion of these articles was that DAR transportation can improve the level of service to the customers without leading to higher costs or energy consumption, when implemented on relevant transportation systems. In some cases, DAR systems might even decrease the cost of transportation while also improving the level of service. One important conclusion in Hobeika[24] was that DAR transportation systems seem to work best in geographical areas with medium to low demand.

Wilson returned to the DARP in 1975[55] proposing a second generation algorithm for computer control procedures. The basic framework of the algorithm is similar to PRAA mentioned previously, but changes are made to the assignment of new customers to existing routes. First, the assignment takes into account the fact that different users will have different preferences, so in order to maximize customer satisfaction, identification of individual preferences need clarification. Secondly the assignment should consider the impact on future assignments. Wilson identifies four different groups of users based on previous experiments. These four groups each have individual criteria for perceving the level of service in the DAR transportation system. The criteria are:

- immediate service
- immediate transferring service
- advanced pickup service
- advanced delivery service.

Immediate service is a criterion for customers who want to be picked up and delivered as soon as possible without having additional deadlines associated with the trip. Immediate transferring service customers have a desire to reach the first possible departing fixed route vehicle at a certain destination. Advanced pickup covers the customers who specify a time window for pickup, and advanced delivery customers specify a time window for delivery. There are of course other parameters controlling the passengers perception of the level of service such as bus driver attitude, but these parameters are naturally beyond the control of the algorithm. After identifying these four criteria, Wilson constructs objective functions according to the dissatisfaction of the passenger groups. When a passenger is classified, the dissatisfaction is calculated accordingly, thus enabling the computer control system to compute solutions maximizing customer satisfaction in much more detail than was previously possible.

The paper by Wilson is interesting, because it includes more than one type of request for transportation in a DAR transportation system. Wilson also formulates the possibility of placing the DAR transportations system within a scheduled transportation system by including the immediate transferring service customers.

In 1978, Stein[48] proposed a possible design of a immediate request DAR System. Based on a mathematical model of a regular bus line, Stein proposes an algorithm for the static problem where a region is divided into k subregions with a fixed transfer point. For each subregion, a bus is inserted to serve that particular subregion. Each bus then traverses the subregion on a TSP tour and ends at the transfer point where passengers can transfer to other regions and so on.

For the dynamic case of DARP, Stein first examines the SV-DARP where a region is again divided into subregions. The problem is then fomulated as decisions in discreet instances of time as to which region the bus must visit next, and to which points in that region the bus must visit. This can be formulated as a queuing system where the units to be served are the feasible pickup and delivery points. When a demand arises, the corresponding pickup point enters the queue, and when the pickup point has been visited the corresponding delivery point enters the queue. The algorithm then starts by visiting the region with the largest number of feasible points and then visits all the feasible points there. This is repeated until no more feasible points are in the system.

For the MV-DARP the algorithm is a mix of the algorithm for the static DARP and the algorithm for the SV-DARP. Here, specialized buses are inserted in the subregions but allowed to traverse more than one subregion according to the algorithm for SV-DARP. The subregions then have transfer points on the boundaries of the regions where all the specialized buses meet to transfer passengers between regions.

Stein then defines a class of algorithms to be used on real DAR problems. This class includes five elements:

- The decomposition of area and specialization of buses. The area must be divided into a large number of subregions, and buses must each have a fixed set of subregions to serve.
- Tours for the buses are determined as the demand becomes known. Each time a bus enters a new region, an optimal TSP

tour is calculated for that region.

- The following rules for visiting subregions are applied. When a bus enters a subregion, it must visit all feasible points in that region before moving on to the next subregion, and all subregions should be visited in a fixed order.

- The transfer points are usually located at the boundaries of the subregions, and buses might only travel between transfer points (line-hauling) with zero subregions allocated.

- Concerning quotation times, the passengers need not know beforehand the expected pickup and delivery times, which is seen as a separate problem.

The theoretical approach suggested by Stein is based on certain assumptions, where the uniform subregions and the constant and large number of passengers seem to be the most obvious limitation. There is also the problem of limited capacity, which is not addressed in this paper. Real passengers also need to know the expected pickup and delivery times. However for useon practical problems, the proposed algorithms might give a general idea of how to design a DAR transportation system.

Stein's results however justified additional research within the design of algorithms for solving DARP. In 1980 Psaraftis[36] presented an exact dynamic programming algorithm using backward recursion for solving the Single-Vehicle, Many to Many, Immediate Request DARP. This dynamic programming solution require $O(n^2 3^n)$ time where n is the number of customers which put a strong limit on the number of customers that can be handled by the method. A more serious limitation of Psaraftis' approach was the omission of any consideration of time windows.

Psaraftis continued his work in another paper in 1983[38]. In this paper a different version of the dynamic programming algorithm is described. The main difference is the use of forward recursion as opposed to backward recursion. Also in this new version of the algorithm, Psaraftis examines the case where customers specify time windows at both pickup and delivery points. The upper bounds of the time windows are considered hard. For example, if a vehicle arrives later than the upper bound, the route is not feasible. The same is true for the lower bounds, where a vehicle will stay idle uppon arrival before servicing the customer when the lower bound of the time window is reached.

The introduction of time windows in the dynamic programming algorithm forced Psaraftis to develop a forward recursion scheme. Naturally, backward recursion can not keep track of time onward from a specified time. The objective of the algorithm is to minimize the time needed to service all customers in the transportation system, and this may not be generalizable to minimizing customer inconvenience etc. The fact that time windows are specified by the customer at both pickup and delivery points may prove to be a problem when solving practical instances, since this severely lessens the planning and solution space.

Also in 1983 Psaraftis[37] published a paper concerning a heuristic for solving the single vehicle many-to-many euclidian DARP. This heuristic is based on the minimum spanning tree of the nodes (pickup and delivery points) of the problem. First a TSP tour through all nodes is constructed using the minimum spanning tree of the nodes and without destinguishing pickup point nodes from delivery point nodes. In the next step a pickup node is chosen randomly, and the TSP tour is then traversed clockwise until all nodes have been visited and included in the DAR tour. While traversing the TSP tour, any node that has been previously visited or any delivery point node where the corresponding pickup point node has not yet been visited is not yet included in the DAR tour. Psaraftis then suggests various improvement methods to be used on the result obtained by the heuristic. One possible improvement method could be regular local interchanges or a repetition of the DAR tour construction step but moving counterclockwise. Lastly the DAR tour construction step could be repeated with different starting pickup nodes. After performing some or all of these improvement steps several times, the best solution found is chosen. This heuristic has a complexity of $O(n^2)$ and results have shown the algorithm very effective at solving this type of DARP, which again is the immediate request DARP without capacity constraints.

As with the backward recursion dynamic programming scheme, a serious limitation is the computational time. In practical cases it is not possible to use a dynamic programming method in an on-line scheduling environment. Another limitation is the lack of constraint on capacity, and the disability to minimize customer inconvenience.

In the beginning of the 1980's, work with DARP began at Centre de Recherche sur les Transports (CRT) and Groupe d'Études et de Recherche en Analyse des décisions (GERAD) in Canada, which resulted in a working paper by Roy et al.[41] in 1983 concerning the Multi-Vehicle DARP. This article addresses transportation of handicapped, and uses a heuristic to divide the customers into clusters after which routes are generated within the clusters by insertion. The heuristic takes into account both the advance request and the demand responsive part of the problem, and the objective is to maximize vehicle productivity for a fixed level of service. The Roy et al. working paper[41] is one of the first examples of the class of algorithms designated as "cluster first route second". Here the requests are characterized by an origin, a destination, and a desired service time corresponding to respectively a desired departure or arrival time depending on the type of request. The request may consist of an individual person or a group of persons to be transported. Also specified for each request is the quality of service that is to be provided. Besides the requests, the heuristic takes into account driver duties which are known in advance and handling times at the stops. The quality of service is then specified by the operator using two parameters. The maximum allowed deviation from the desired pickup or delivery time and the maximum excess ride time of the passenger in the vehicle. Excess ride time is defined as the difference between the actual ride time and the minimal ride time necessary to service the request. Time windows can now be calculated according to the description in section 6.3 and figure 6.1 (1) on page 77.

To construct the initial clusters, a neighbor concept is introduced. First a maximum time between stops related to two different requests is given, and then the following conditions are set up to define two requests as neighbors. If the difference between the latest departure time of one stop and the earliest arrival time at another stop is less than the specified maximum time, the stops might be considered neighbors. However this is only enough in the case where the stops are of different types (e.g. one is origin and the other is destination). If the stops are of the same type, another condition is required to ensure the requests are going in the same direction. This condition is constructed by looking at the other stops in the two requests where an ellipse is defined around the stops.

After having constructed the clusters according to the neighbor concept, the authors introduce a measure of affinity between a request and a route. First, the feasibility of inserting the request into the route is examined. Infeasibility results in a very large measure of affinity. For all feasible solutions, the measure of affinity is calculated as the detour of the route by inserting the request. The affinity values are calculated for all requests in all routes and inserted into a matrix. The heuristic then starts by initializing one or several routes with neighboring requests. The requests are then read in chronological order and insertion is attempted. If no feasible insertions are possible, a new route is constructed with the request and all neighboring requests are now inserted if feasible.

To limit the problem size, the authors use a time horizon for planning. Since the requests placed in the afternoon usually do not influence the morning requests, the requests to be considered are within a given time horizon. As the day moves forward, the horizon is extended and new requests are included. When all requests within the horizon are included in the solution, a post optimization teqnique is applied to minimize cutomer inconvenience. Here the requests are evaluated according the the actual deviation from desired service time, and swapping of requests between routes are tried to improve these values. The heuristic was implemented and tested on several data sets from Montreal. For larger problems results show a great improvement over the existing manual planning with respect to vehicle and driver utilization.

The success of this work resulted in further development and tests which are described in a series of techical reports from 1985[43][42][44] addressing respectively test problems, implementation, and results.

Also in 1983 Daganzo[8] published a very interesting paper, in which three different types of transportation namely Fixed Route Transit (FRT), Checkpoint Dial-a-Ride Transit (CPDART), and Door-to-Door Dial-a-Ride Transit (DDDART) were compared mathematically. Although the problems were simplified for comparison purposes, the results seem to be much in line with later experimental results. Daganzo looked at the cost of operating the different transportation systems under different demand pressures, and found that in the case of a large demand pressure, the FRT outperformed the other two types both in cost and in complication level of operations. However as demand pressure decreased a CPDART transportation system outperformed the FRT but by no more than 5% . When the CPDART outperformed the FRT, the DDDART was a serious competitor, and would often outperfom CPDART. The paper concluded that only in a very limited demand pressure interval was the CPDART superior but probably not superior enough to account for the extra cost of dispatching. The FRT should be used in areas with high demand pressure, and as demand pressure decreases, a regular DDDART should be used.

Belisle et al.[4] presented a conference paper in 1984 concerning the impact of using different transportation systems to service transport of the handicapped. They simulated three different scenarios. Two scenarios where minibuses where solely used and one scenario with mainly taxis as the means of transportation. In one of the minibus scenarios they used a call-back system, thus making the planning much more flexible since requests did not need to be planned right away. For the other minibus and the taxi scenario there was no call-back procedure, and expected time of pickup etc. was given to the customer upon making the request. In this simulation, service to all requests were guaranteed with the exception of requests phoned in on the day of service. Booking of requests and cancellations would have to be given at least one hour prior to departure. All desired service times are guaranteed, eg. no pickups can occur before and no deliveries after the desired time, and the pickup time is given as a 15 minute time interval in which the vehicle is guaranteed to arrive. The data used in the simulation was taken from the Wheel-Trans Service in Toronto on a regular work day. In total there was 1096 requests in the dataset, where 410 requests were regular (eg. known in advance) and 686 requests were occasional (eg. know up to 3 days in advance). The cancellation rate used was about 17% .

The algorithm used to solve the above problem was the clustering first parallel insertion technique developed by Roy et al.[41] (see earlier this chapter) and to improve results a dynamic programming method developed by Desrosiers et al.[9] was used. The result was that route operating cost was higher for the minibus scenario without call-back than with call-back, but this does not take into account the cost of performing the call-back procedure. The transportation of hadicapped is more like a taxi service than a bus service, but the choice of operations is not so much a question of level of service, but more a question of organisation.

Desrosiers et al.[10] in 1984 developed an algorithm based on clustering first and column generation second for the Multi-Vehicle many-to-many VRP. Although not mentioned in the title of the paper, the algorithm is in fact developed for solving the DARP or PDP. Again the problem consists of requests characterized by a pickup stop and a delivery stop, time windows at stops, pickup and delivery times, and a homogeneous vehicle fleet with capacities for wheelchairs and ambulatory passengers. There can be multiple depots, and the objective is to first minimize the number of work pieces (which can be seen as independent routes), and secondly to minimize the total distance travelled by the vehicles.

The algorithm takes into account capacity constraints and depot constraints, and the same vehicle must visit both the pickup and delivery stop for a request. Starting with a feasible solution, a single vehicle algorithm[9] is used to optimize each of the initial routes according to driven distance. The routes are then divided into miniclusters of customers after which integer linear programming with column generation is used to reallocate the constructed clusters between vehicles. At last the miniclusters are again opened, and the single vehicle algorithm once again optimizes the individual routes. The single vehicle problems are solved by dynamic programming, and this method will be addressed later in this chapter. Since the single vehicle problem consists of the TSP with additional time window, capacity, and precedence constraints it can be solved by dynamic programming more efficiently than the regular TSP. This is due to the fact that the solution space is severely limited be the additional constraints. The authors state that the dynamic programming approach has shown to have an execution time that is a linear function of the number of stops.

Desrosiers et al. used data in which a route usually serves between 5 and 20 customers, the number of stops are between 10 and 40. These customers are divided into miniclusters each time the vehicle becomes empty. This means that a minicluster is characterized by being a group of customers that are in the same vehicle at some point in time. On average fewer than four customers are in a minicluster. The minicluster now forms an artificial customer with time windows according to the customers with earliest pickup time and latest delivery time and a duration time. All constraints are naturally satisfied within the minicluster.

To solve the routing problem the authors create a network structure in which the nodes are the miniclusters, and the arcs are the unproductive run of a vehicle as it drives empty from one minicluster to another. A mathematical formulation of an integer programming problem is then presented with two types of variables. First the regular binary flow variables followed by a continuous time variable associated with the visit starting time for the miniclusters. Instead of solving the problem in terms of the flow variables, the problem is formulated and solved in terms of binary variables on the use of feasible paths in the graph. Since enumeration of all feasible paths is not possible, a column generation scheme using Dantzig-Wolfe decomposition is used. The columns are generated by solving shortest path problems with scheduling constraints.

The above mentioned method was tested on problems from 3 Canadian cities. The number of requests in the 6 test problems varied from 86 to 880. The number of stops is only roughly twice the number of request since additional stops have been added to simulate multiple depots, just as some stops are at the same location and therefore considered one. The size of the miniclusters were on average 4 customers, but some miniclusters consisted of more than 10 customers. Good solutions were found in comparatively short computational time (measured in minutes for the largest problem). The work resulted in a paper published in 1988[11].

Some of the papers above have used the dynamic programming method developed by Desrosiers et al.[9]. This algorithm uses a forward scheme to solve the SV-DARP with the objective of minimizing the total distance. To ensure comparatively fast computational time, criteria for the elimination of infeasible states have been developed. The problem is essentially a constrained TSP where the non-linear time constraints require increasing times at each stop, thus eliminating the possibility of sub-tours. Since the objective is to minimize the distance travelled, the customer inconvenience is only indirectly included in the formulation by the construction of the time windows.

The dynamic programming algorithms works as follows: The vehicle starts at the depot and at each iteration there are several states to choose from. In the first iteration the states are made of routes visiting a single stop representing an origin. In each of the following iterations, the states are constructed from the previous states including one additional stop among all the stops. The last iteration requires the vehicle to move back to the depot. At each iteration the number of possible states is reduced by applying the capacity, time window, and precedence constraints, thus improving performance of the method dramatically. Results on test datasets with up to 40 customers which corresponds to the TSP with 80 nodes show a computational time measured in seconds, showing the effectiveness of the state elimination scheme.

In 1986 Jaw et al.[28] published a paper on which a lot of the later work is based. The paper describe a heuristic for the Multi-Vehicle DARP based on sequential insertion of customers on vehicles and thereby deciding a time schedule for pickups and deliveries for each vehicle. The objective function minimizes the cost of transportation balanced with the minimization of inconvenience to customers. The algorithm is referred to as ADARTW short for Multi-Vehicle many-to-many Advanced request Dial-a-Ride problem with Time Windows. Only the static DARP is considered. The problem considered here is the same as in Roy et al.[41] where customer inconvenience is measured in terms of deviation from desired pickup or delivery times and excess ride time. Time windows and excess ride times are individual parameters for each customer. This enables some negotiating of these parameters with the customers. The fleet of vehicles is heterogeneous, and dwell times (handling times at stops) are considered. One additional constraint that had not been considered before is that a vehicle is not allowed to be idle when carrying passengers.

ADARTW starts by sorting the customers in a specific manner. In this paper the authors chose to sort the customers according to their earliest pickup time, but different strategies can be used to obtain different solutions. The algorithm then processes each customer in the list in sequence, and assigns the customers to the vehicles until the list is exhausted. For all feasible insertions of a customer to a vehicle the additional cost measured in operating costs and customer inconvenience the insertion with the lowest cost is performed. If a customer cannot be inserted into an existing route (vehicle), a new vehicle is initialized or the customer is rejected.

The authors also mention the possibility of considering more than one customer at a time. However this do not seem to improve results substantially, and computational time increases to some extent. The feasibility check when inserting customers into schedule blocks (blocks of active vehicle time between idle time) first checks if the insertion violates the time window constraints of the existing stops in the block without creating idle time in the middle of the block. If time window feasibility is found, the next step is to consider the excess ride time for both the inserted customer and the customers already inserted in the schedule block. The insertion is performed one stop at a time, eg. the feasibility of inserting the pickup point leads to the insertion of the delivery point. After having found all feasible insertions of the new customer into the schedule blocks, the incremental cost of insertion for each feasible insertion is calculated. This incremental cost is calculated of three parts:

- The inconvenience of the customer to be inserted is calculated as the sum of the inconveniences caused by respectively the deviation from desired service time and the excess ride time.

- The inconvenience for all customers in the block where the new customer is to be inserted is then calculated as for the customer to be inserted. Note that it is an incremental cost (eg. the difference in inconvenience of the customers in the block before the insertion and after the insertion).

- The incremental cost of the operator is then calculated as a weighted function of the additional active vehicle time (the expansion of the schedule block), the change in vehicle slack time (future flexibility of the block), and an indicator of system workload.

ADARTW has been run by Jaw et al. a large number of times on both simulated and real data, and the conclusion of the authors are that solutions found by ADARTW both in case of simulated data as well as real data are as good or better than manual planning in all respects. Computational time for real problems with 2617 customers could be solved on a VAX 11/750 in minutes.

In 1986 Psaraftis[39] published a paper comparing ADARTW with the Gouping/Clustering/Routing (CGR) algorithm developed at MIT in Boston in 1982, and described in a working paper by Jaw, Odini, Psaraftis, and Wilson. The CGR algorithm models customer satisfaction by the difference between actual pickup/delivery time and the desired pickup/delivery time. Also in CGR the vehicle fleet size is given beforehand, whereas in ADARTW the fleet size can be a variable. Both algorithms consider the advance request problem, thus treating the static problem. CGR operates with time groups of for example 30 minute intervals (eg. when the customer specifies a pickup time, an attempt will be made of picking up the customer in the corresponding time group). This also meens that the customer service guarantees are soft in CGR while they are hard in ADARTW.

The algorithm CGR works in three steps:

- The customers are grouped into time groups based on either their desired pickup or delivery time. The customers are then futher divided within a time group into clusters. This is done by looking at each time group and finding the customers that are furthest apart in distance. These customers are then designated seed customers, and there are as many seed customers as there are available vehicles.

- The vehicles are assigned to the seed customers in a least cost manner, after which the rest of the customers in the time group are assigned to the clusters formed by the vehicles and the seed customers.

- Routes within the clusters are constructed by applying a single vehicle routing algorithm on the clusters while checking capacity of the vehicle assigned to the cluster. The objective here is to minimize driven distance.

To compare CGR with ADARTW, a real and a simulated dataset was used. The real dataset covered 16 hours of operation with 2617 customers. 2397 customers were immediate request customers and the last 220 were advance request customers. Since both CGR and ADARTW only considers advance request customers, the data for the immediate request customers were converted so the actual pickup time became the desired pickup time. The fleet used in the dataset was not heterogeneous which also is a presumption used by both agorithms, but consisted of one vehicle with capacity 33, 4 vehicles with capacity 9 and 23 vehicles with capacity 17. The algorithms were then tested with a capacity of 17 for every vehicle, and ADARTW was initialized with an active fleet of 10 vehicles with the possibility of adding more vehicles as needed.

Because of the conversions, it is not possible to realistically compare the results from the algorithms with the actual solution at the dataset provider. Results of the comparison of the two algorithms on the real dataset shows a significant better performance of the ADARTW, which uses around 19% fewer vehicles and has a deviation from desired service time that on the average is less than one third of that produced by CGR. In addition the ADARTW naturally serviced all customers within the designated time windows which was not the case with CGR. The passengers in the CGR solution had to spend on average almost 50% more time in the vehicles. However the solution time for CGR was almost half that of ADARTW. When comparing the algorithms on simulated data, the difference between the solutions is not nearly as significant, and CGR actually outperforms ADARTW in some objectives while still using less computational time.

In Japan in 1990 Kubo & Kasugai[31] worked on several heuristics based on insertion algorithms, and a number of different insertion methods are described. In this article some of the previous algorithms are examined and compared with newly developed approximative algorithms. The authors present four methods for solving DARP without time windows or capacity constraints. The objective is solely to minimize the total driving distance of a single vehicle DARP.

The first method is based on sequential insertion, and after concluding that the methods inserting either a pickup point or a delivery point at each iteration provide bad results, they formulate a pairing insertion method, in which both pickup and delivery points of a customer is inserted at once. The insertion is straight forward greedy insertion where the next customer to be considered can be either the one farthest from the existing route or closest to the existing route.

The second method is based on randomized nearest neighbor principle. Here the pickup or delivery points are inserted individually, and the authors start by adding the depot and a pickup node. The next point to be added to the route is decided by using a probability function, where respectively one of the delivery points or the pickup points are visited with a certain probability.

The third method is derived from the minimum spanning tree heuristic developed by Psaraftis (described earlier in this chapter), where instead of using the minimum spanning tree to devide customers into regions, a space filling curve is used.

The fourth method is a local search procedure to be used as a post optimization procedure. It is essentially the Or-Opt method known from TSP, but modified to remove a pair (pickup and delivery) of points from the tour at once.

The results obtained by the authors from running on 10 constructed problems with 10 to 100 points (plus the depot) shows that the insertion method seems to outperform all other methods in terms of objective value. The local search procedures do not perform well under any circumstances, and they also require much more computational time. The various alterations of the Opt method works very well both with a bad and a good feasible starting solution.

In 1991 Desrosier et al.[12] wrote a report concerning an improvement in the construction of miniclusters. As the authors state, this part of solving the DARP (when using a minicluster algorithm) is essential, since the quality of service is closely linked with the resulting clusters. To construct the miniclusters in an intelligent way, the first thing to consider is an intelligent determination of neighboring requests, and the authors present a method taking into account both spacial and temporal proximities, directionality, economy of distance, and feasibility.

The temporal proximity between two requests is given by the cost (C) in distance of servicing the requests immediately after one another. If not possible to do so, c is arbitrarily large. The savings obtained by clustering two requests is measured as the sum of the cost of servicing each request subtracting X. By considering the closeness in time windows for the stops associated with two requests, the temporal proximity is checked. If it is possible to service two requests without leaving the vehicle empty in between, they are close enough in time to be considered neighbors.

The spacial proximity is calculated from an ellipse where the foci are the stops of a request. All requests with one of their stops within the eclipse can be considered neighbors with respect to the spacial proximity. The directionality is defined as an angle between two requests. This angle must be less than a specified maximum angle. To construct the miniclusters, the requests are sorted decreasingly according to their total direct duration of service, since these requests offer most flexibility when trying to incorporate new customers in the clusters. To calculate the costs of inserting customers into clusters, the previously mentioned dynamic programming method is used. After performing some tests with this new strategy of creating miniclusters, the authors conclude that the numerical performance is good and very fast.

Also in 1991, Ioachim et al.[26] presents a mini clustering algorithm using column generation similar to the previously described paper by Desrosier et al.[10]. The master problem is here formulated as a linear relaxation of a set partitioning problem, and the subproblem is again formulated as a constrained shortest path problem. To obtain integer solutions, a branch and bound strategy corresponding to branching on time variables is used. At each node of the branch and bound tree, the columns (miniclusters) are generated when needed, and the reduced cost of a minicluster is calculated as the sum of the reduced cost of the arcs within the minicluster. To solve the subproblem, a forward dynamic programming shortest path algorithm is used. As seen in Desrosiers et al.[9], the number of labels used in the dynamic programming algorithm can be reduced significantly by eliminating infeasible paths in the constrained shortest path problem. The dynamic programming algorithm is at initialization given the path of length 3 (eg. source

-> i -> $n+i$ -> sink) from the starting point, since this path need only be computed once.

The algorithm was tested on 20 problems originating again from the previously mentioned Wheel-Trans service in Toronto. The datasets had between 50 and 250 requests, and the optimal solution was found on 17 of the test problems using this heuristic. The solutions were compared to the parallel insertion heuristic described by Desrosier et al.[12], and results show, that the insertion algorithm was outperformed by the set partitioning formulation by 10% with regards to the travel time. However the computational time for the insertion heuristic on the 250 customer problem was less than 15 seconds, whereas the set partitioning formulation used 5133 seconds. The work is also presented in a paper from 1995[27] where test on an actual problem with 2545 requests is presented. The result was a 5.9% improvement in total travelling time over the heuristic. In order to solve the large problem with the set partitioning method, the problem was divided into seven smaller problems. These problems were then solved in about 5 hours of computational time.

In Denmark a Masters thesis concerning transportation of patients in a dynamic DAR environment with Falck (a Danish operator) was finished in 1991 by Christensen and Jensen[7]. The thesis describes the development of two strategies for solving the problem. Both strategies are based on ADARTW mentioned previously, which is expanded to include some of the problem specific parameters stated by Falck. The problem here is to minimize the total drive time for all vehicles while also minimizing the discomfort of the passengers. The discomfort is measured by deviation from expected departure respectively arrival times of the vehicles at the stops, as well as the excess ride time of the passengers.

Falck used a limited number of vehicles “selling” rejected passenger trips to taxi companies. The first strategi called “PDARtaxa” uses the greedy insertion structure of ADARTW to insert as many passengers into the vehicles as possible. All passengers not inserted are then sold to taxi companies. The second strategi constructs as many vehicle trips as necessary in order to accommodate all passengers. The vehicle trips are then assigned to the vehicles. If the capacity offered by the vehicles is insufficient, the cost of selling a vehicle trip to a taxi company is calculated for each trip. The vehicle trips are then assigned to vehicles and taxi companies according to the calculated costs.

Results from using the strategies on a real problem with 370 requests in one of the regions operated by Falck showed a 20% saving in cost. Also important was the average deviation from desired service time, which was improved from 50 minutes to 13 minutes. The CPU-time used for solving the static problem was less than 2 seconds.
Although it is not really related to algorithms within the field of Operations Research, an interesting article was published by Williamson in 1992[53] concerning the development of an advanced user interface to assist the planners of Dial-a-Ride problems in daily scheduling. The software package developed was designed to help dispatchers and call centers in sorting and storing data to be displayed in a helpful manner on a computer screen, thus giving dispatchers a greater overview of the possible solutions. The author mentions that a computer controlled planning procedure used in conjunction with the interface software might give a great improvement in the planning process.

At CRT in Canada, work concerning the DARP continued, and a new heuristic was proposed by Potvin and Rousseau in 1992[35] as an extension to the work presented by Jaw et al.[28]. Here a constraint-directed search is applied to the problem, and the computational results are compared to those of ADARTW. Once again schedule blocks are used, thus making a solution not feasible if a customer waits in an idle vehicle.

The proposed constraint-directed search algorithm can be formulated in three phases:

- clustering (optional)
- beam search
- post optimization.

The clustering phase is optional and consists of identifying groups of customers that fit naturally together. This is accomplished as previously described in the paper by Roy et al.[41]. After clustering the customers, they are then sequenced by searching for a feasible sequence within each cluster, by constructing a route and iteratively inserting the customers of a cluster into the route. If customers can not be directly inserted into their corresponding route, they are discarded from the initial construction. Instead of just inserting one customer at a time into the constructed routes, the authors propose a beam search procedure that helps take into consideration the customers that have not yet been inserted. The not yet inserted customers are sorted according to their earliest pickup time, and the customers with earliest pickup time within a given time horizon are then examined. At each of these states, an additional option is included, namely the construction of an entirely new route. A set of the customers giving the best possible solutions in the beam search procedure are then inserted into the routes, and the rest of the customers are discarded until the next iteration. The number of customers to be assign to routes at each iteration is called the beam search width, thus the ADARTW has a beam search width of one. During the beam search phase, solution quality is evaluated by summing up the values of weighted utility functions concerning cost of operation and customer inconvenience measured in deviation from desired service time and excess ride time. Cost of operation is given as number of vehicles to be used, total vehicle time, utilization of vehicles, and distribution of customers among vehicles. After having inserted all customers into the routes, a post optimization phase is started. Here two swap operations and an OPTIMUM procedure are used.

SWAP-1 aims to eliminate the smaller routes by selecting all the customers in the smallest route and considering them for insertion by the beam search procedure into all other existing routes. If all customers in the smallest route can be inserted into other routes, SWAP-1 then considers the next smallest route. SWAP-1 stops when feasible possibilities of moving customers are exhausted. SWAP-2 is then applied to each two customers in turn. Here the pickup and destination points of two customers are simply exchanged between routes while maintaining feasibility. SWAP-2 stops when there are no more pairs of customers where an exchange can improve the solution. Within each schedule block OPTIMUM is now applied. This step consists of finding the optimum sequence of the stops within the schedule block. In order to avoid a large increase in computational time, only blocks with three customers or fewer are considered.

The constraint-directed search procedure was tested on data with 90 customers in 9 hours of operation, (ie. 10 customers per hour). The results were compared to those of ADARTW, where the beam search procedure generated a better solution with respect to both service quality and cost of operation. As expected, increasing the beam width generate even better results, although with a considerate increase in cumputational time. ADARTW was about twice as fast (66 seconds) as the beam search procedure (125 seconds) with respect to computational time.

A Masters thesis by M. K. Mikkelsen from Denmark was presented in 1994[33] comparing results of a miniclustering algorithm with results from an extension of ADARTW called REBUS (see below). The miniclustering algorithm was based on the algorithm mentioned in Ioachim et al.[26] previously in this chapter, where column generation is used in the construction of routes resulting in a set partitioning problem. The miniclustering algorithm was extended by Mikkelsen to include additional parameters to control passenger discomfort and to enable a more direct comparison with the results from REBUS.

Mikkelsen concluded that the results obtained by the minicluster algorithm were superior when stops were clustered, and in 12 of the 15 test problems, the miniclustring algorithm outperformed REBUS with regards to solution quality. However the CPU time used by the miniclustering algorithm was substantially above that of REBUS, which could limit the practical use of miniclustering.
In 1995 Madsen et al.[32] presented an insertion algorithm based on ADARTW developed for use by the Copenhagen Fire-Fighting Service (performing ambulance service, transportation of disabled and other specialized transports besides fire-fighting). The algorithm REBUS was designed to handle in excess of 50.000 requests per year, and implemented in a dynamic environment intended for online scheduling. Response time of inserting an immediate request is less than 1 second, permitting interaction between system, callcenter and customer. The DARP treated here differs somewhat from previously considered problems because of the multi dimensional capacity involved. REBUS is also designed for the dynamical case of DARP.

To enable the operater to control the solution with respect to various objectives, the authors introduce several operator controlled parameters. To control the insertion procedure, requests are sorted according to difficulty of insertion. Difficult requests are inserted first. The difficulty is measured as a sum of difficulties related to time windows, excess ride time, and capacity requirements. A narrow time window is harder to satisfy than a wide time window, large excess ride time is easier to comply with than short excess ride time, and small capacity requirement is easier to handle than large capacity requirements. When dealing with capacity, note that the algorithm operates with multiple capacity, eg. regular seat, child seat, space for wheelchair, and space for beds. The different possible insertions of requests in a preliminary plan are ranked according to parameters concerning driving time, waiting time, deviation from desired service time, and capacity utilization. The insertion algorithm REBUS now processes the requests to be inserted according to their difficulty by generating all feasible insertions into the existing routes, one for each vehicle. The feasible insertion generating the smallest change in the objective function is then performed. If no feasible insertions exist, the request can be assigned to a fictitious vehicle eg. a taxi.

Although REBUS is based on ADARTW, it differs to some extent in the way the requests are sorted. As stated above, the requests are sorted according to cost. REBUS also makes it possible to minimize the waiting times for vehicles, just as there is an upper limit on the total ride time of each customer. REBUS also allows for vehicles which can have different capacities that exclude one another. There are also some differences in the insertion procedure, where REBUS reduces the number of feasible insertions by applying capacity, excess ride time, and time windows constraints at this stage.

REBUS was implemented in the C++ programming language and tested on a HP-735/9000 computer with the UNIX operating system. The size of the dataset was 24 vehicles and 300 customers taken from real data provided by the Copenhagen Fire-Fighting Service, and the problem was solved in less than 10 seconds. The dynamical addition of a new customer to the existing routes took less than 1 second once the original problem was solved.

In 1995 Healy and Moll[22] published a paper concerning an extension to the traditional local improvement procedure. This extended local search was then applied to the DARP. The heuristic is based on a concept called “sacrificing” by the authors and show considerable improved results when compared to the solution quality of the traditional local search heuristic. In general the extension is formed by adding an extra goal to the search procedure. When a local optimum is reached with respect to the primary goal, a sacrificing step improving the secondary goal is allowed even though the solution quality is temporarily worse.

Based on neighborhoods generated by applying a swap procedure to the current solution, the extension of the local search heuristic can be described as follows. Whenever a local optimum is found, the seach heuristic enters a sacrifice phase, where the size of the neighborhoods to the local optimum are evaluated. By using 2-opt or 3-opt swap methods on the requests in routes, the number of possible swaps can be quickly estimated, and the largest neighborhood can be identified. Although moving to a solution with a large neighborhood results in worse solutions with respect to the objective function, the step is allowed, and a search starts for a local optimum in this new solution. There are various ways of terminating the sacrifice phase, but the authors here chose to sum the length of the segments sacrificed in each 2-swap. When this total length exceeded twice the number of customers, the sacrificing step was halted.

The extended local search procedure was compared to the traditional local search on 10 dataset constructed with 10 to 100 customers. All distances were Euclidian. On these datasets the sacrificing step was performed with the 2-opt swap procedure, and compared to a local search heuristic using 2-opt swapping and 3-opt swapping. Because of the much larger neighborhood generated by the 3-opt method, this also generated by far the best result. However the computational time for running 3-opt is far larger than for the 2-opt because of the resulting much larger search space. The authors show solutions of their sacrificing extension with respect to total driving distance that are in between the solutions obtained by regular 3-opt and 2-opt. This is accomplished in about 15 times the computational time of 2-opt, but still a factor of 20 faster than the regular 3-opt. Although not explicitly indicated in the paper, the extended local search seems to be very similar to a Tabu Search method.

Robert Dial introduced a new design in fully automated planning for the DARP in 1995[14]. He suggests that each vehicle in the fleet operates with its own computer communicating solely with the computers in the other vehicles, thus enabling the fleet to act like a swarm of ants. The transportation system resulting from this structure is called ADART (Autonomous Dial-a-Ride Transit), and all functions in the system are maintained by computers with no human interference. The billing for using the transportation system is performed electronically, and upon subscription to the transportation system, the customer registers a personal list of addresses to be used when placing a request in the system. When a request is received, it is sent to a vehicle, that can either include it into its own schedule or send it off to another vehicle. This all happens without the drivers' knowledge, and their only job now is to follow instructions from the onboard computer. Whenever a vehicle computer has free cycles it works on optimizing the current schedule. When a request is received by the computer, it calculates its marginal cost of servicing the request. The information about the request and the initial vehicle's marginal cost is broadcasted to the other vehicles, who then calculate their own marginal cost. Any vehicle with a better marginal cost responds to the broadcast, and the request is send to the vehicle best suited to service the request. There is of course need for some centralized functions in order for the transportation system to work. These functions are for instance billing information, personal address list storing etc.

In 1996 Ben-Akiva et al.[5] published a survey concerning the impact on demand by introducing a Dial-a-Ride transportation system. Different service quality parameters were tested according to the expected effect on demand. Results showed that one of the most important parameters is the travel time and time window of pickup. For every 5 minutes of extra excess ride time, the demand lowers with 10-15% , and increasing the time window of expected pickup also lowers the demand substantially.

Considering the advance request DARP, Toth and Vigo published a paper in 1997[51] where a set of real-life instances from Bologne involving about 300 requests were solved using a parallel insertion heuristic called TV. The problem here consists of a mixed set of vehicles, where the objective is to minimize the use of taxis with regard to both number of taxis and distance/time driven by the taxis. This is to be accomplished within a specified service quality. The heuristic proposed here is based on the relaxation of the desired service time by the introduction of a piecewise linear user inconvenience penalty in the objective function.

TV starts by initializing a set of routes in which the size of the set is given by an estimate of the minimum number of vehicles needed to serve a fraction of the requests. The estimate is calculated by considering only the capacity requirements of the problem discarding all other constraints. At each iteration of the insertion of requests into routes, a min-cost Rectangular Assignment Problem is solved on a matrix of requests and routes. Each element in the matrix describes the marginal cost of the best insertion of the request into the route. If no feasible insertion can be found, the marginal cost is set to a very large number, and where a request cannot be inserted into any route, a new route is initialized based on the available vehicle best suited to serve the request. To further improve the solutions obtained by TV, the authors introduce a Tabu Thresholding algorithm (TT). TT works in a way very similar to the algorithm described in Healy and Moll[22], but instead of searching the largest neighborhood, a candidate list is maintained in TT. Where the extended local search used a sacrifice phase, the TT algorithm uses a mixed phase (performing respectively "sacrificing" steps and improvement steps) that is repeated for a certain number of iterations, whereafter an improve phase takes over. The two phases are alternated until the best known solution has not been improved for a certain number of iterations.

The neighbourhood of a given solution can be obtained by performing three different steps:

- Remove a request from a route, and try to insert it into a different route.
- Perform a swap of two requests belonging to two different routes.
- Perform a trip double insertion. Here a request is removed from a route and inserted into a nother route. Then a request from a third route is inserted into the first route.

The above mentioned TV and TT were applied on the datasets from Bologna, and tests were run on an IBM Personal Computer 486/66. The solution obtained by TV in less than 30 seconds of computational time satisfy all operational constraints, and offered a considerably better level of service than the hand made solutions. The hand made solutions were slightly infeasible, but resulted in trip durations almost as good as those of TV. TV was able to reduce the cost of operation by more than 36% by reducing the number of taxis used from 245 to 108. By running the TT improvement heuristic the number of taxis used was reduced even further to only 42 taxis while still producing a solution that is better than the hand made solution with regards to all parameters.

In 1998 Charikar and Raghavachari[6] talked at a conference about the finite capacity DARP. They introduce a non-trivial approximation algorithm for the capacitated DARP without time windows. Also they consider the preemptive DARP, which allow delivering the load at intermediate locations, thus pickup up the load later for final delivery. Although the the problem is referred to as Dial-a-Ride, it is perhaps better classified as the Pickup-and-Delivery problem, but it still deserves a little space in this chapter.

As meta heuristics are being used to solve still more problems, Baugh et al. introduced a solution to DARP using Simulated Annealing in 1998[30]. The authors use Simulated Annealing since they find it appears to be the meta heuristic best suited, since the technique is easily adapted to problems with a well defined neighborhood structure and it seems to find near-global optimums in a reasonable amount of computational time. The authors also state that Simulated Annealing is preferable because of the ease of integrating other meta heuristics such as Tabu Seach into the procedure.

To solve the problem using Simulated Annealing, the authors use a Cluster-First Route-Second strategy where the clustering is performed by simulated Annealing and the routing within the clusters are done by a space-time nearest neighbor heuristic. The authors argue that the most crucial part of DARP is the assigment of customers to vehicles and not the routing part which usually only involved a comparatively small set of customers. The clustering technique is initialized by randomly assigning customers to clusters, and then two different operations are used to alter the clusters. The first alteration is obtained by randomly swapping customers from different clusters, thus leaving the total number of clusters, and the size of the clusters the same. The second alteration is obtained by swapping randomly selected customers to randomly selected clusters, thus making it possible to annihilate clusters, generate new clusters, and change the size of the clusters.

The strategies for altering the clusters are desirable since they are simple. Also they give an opportunity to generate a cluster from another cluster which is important when using Simulated Annealing. The exchange operation allows for a smoother solution space, whereas the swapping allows for dynamical updating of the clusters. At each iteration of the Simulated Annealing, the routing heuristic is invoked on the clusters and the number of vehicles needed to service the routes is reassessed. The objective function used to evaluate the resulting solution penalizes the total distance traveled by all the vehicles, the total disutility of the customers, and the number of vehicles used. If the new set of clusters result in an improvement, it is accepted. Otherwise it is accepted with a probability calculated with the parameters change in cost, and temperature.

To further improve the results of the Simulated Annealing scheme, a Tabu List is introduced. This list keeps in memory the accepted transitions at each iteration so as to give the transitions a certain amount of time to show their effectiveness. The resulting algorithm has been tested on a data set provided by the Winston Salem Transit Authority with more than 300 customers a day. A significant number of the customers travel only very short distances. Results show that the algorithm presented here improves the manual solution obtained in all respects. Travel time is reduced by 25% which is also the case for duration of individual trips and number of vehicles. However the computational effort required to obtain these solutions was not provided by the authors. In general it would seem better to use a Tabu Search Heuristic than a heuristic based on Simulated Annealing since Tabu Search rely on special characteristics of a problem. In the case of DARP, the expected solution structure is so well known it can be used to develop intelligent neighborhood search procedures to improve solution quality in a Tabu Search environment.

5.1.1 Chronological overview

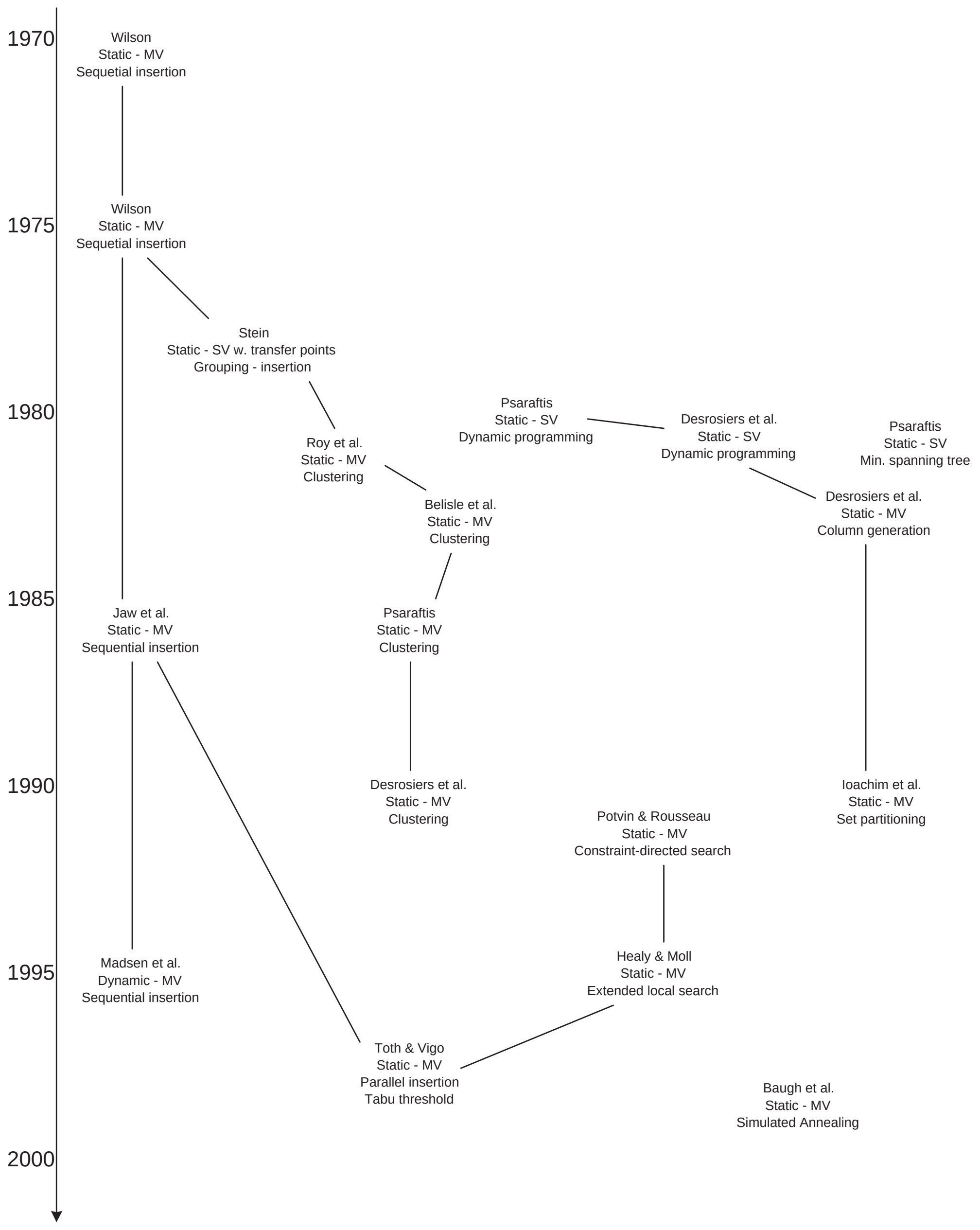


Figure 5.1: Chronological overview of some of the publications on the Dial-a-Ride Problem over the years.

5.2 Dial-a-Ride in Denmark

The development within the area of alternative public transportation methods in Denmark is not quite as extensive as in other countries. However after a few years' delay, we started examining the possibilities of introducing new transportation systems by analysing the need for and gains made by such an introduction. While Canada, for instance, established a Dial-a-Ride bus system in the mid 1980's (see section 5.1), we did not start this development until the beginning of the 1990s.

5.2.1 Standard of reference

The following sections will describe various initiatives in Denmark within the area of demand responsive public transportation. In order to be able to compare these initiatives, this section will list parameters concerning the objectives and transportation systems chosen here to evaluate and compare the different projects. The actual comparison is done later in section 5.2.5.

There are two main objectives when transforming public transportation systems. The first priority is often to raise the level of service without increasing costs, but in many cases raising the level of service in some areas lowers the level of service in other areas. The following list of objectives should give an overall idea of the concept.

- Decrease total costs
  - by decreasing level of service in some or all areas.
  - while keeping current level of service in all areas.
  - while raising level of service in some or all areas.
- Increase level of service
  - in some or all areas with no increase in costs.
  - in some or all areas with little increase in costs.
  - in some areas by lowering service in other areas.

The areas mentioned above are:

- Traditional public transportation in more populated areas.

- Traditional public transportation in less populated areas.
- Transportation of senior citizens.
- Transportation of handicapped.
- Transportation of temporarely disabled.
- Delivery of goods and food for senior citizens.

To reach the above mentioned objectives, different transportation systems can be used. The solution methods used in Denmark up until now are listed below. Note that these transportation solutions can either be added to an existing system or replace some/all of the existing system.

- More extensive scheduling of existing public transportation.
- Long term (typically 5 years) contracts with independent operators and
  - centralized planning.
  - decentralized planning.
- Dial-a-Ride systems with
  - pure door-to-door transportation.
  - regular stops.
  - a combination of door-to-door and regular stops.

5.2.2 Initial projects

HUR is by far the largest single planner of public transportation in Denmark. Thus they were also the first to introduce new concepts in the coordination of specialized public transports. In the late 1980's HUR started a project to centralize the planning of a demand responsive transportation system, hoping thereby for a massive saving on the overall costs in the area.

In the beginning of the 1990's HUR was operational, and over the years they have obtained a significant decrease in costs caused mainly by two initiatives. First, a central exchange was established to handle all requests from customers, which decreased the administrative costs of the system. Secondly, the gathering of all request in one place made it possible to coordinate the transportation in a much more efficient way because of the better general overview. Thus HUR was able to invite tenders for parts of the total transportation need, which again resulted in a decrease of the cost per time a passenger is in the system.

The results today of the reorganization within HUR are 47 % fewer driven kilometers total and a decrease in costs per trip of 69 %. These numbers from before the reorganization used in this comparison are of course adjusted to reflect the current situation. All in all the HUR model has been very successful in reaching the expected goals.

In 1995 COWI (Danish Consulting company), in cooperation with Grenaa Kommune (Danish municipality) and the Danish State Department of Traffic, started analysing performance by simulation of a system named "Kaldebus". Kaldebus consists of regular bus stops beeing serviced by minibuses, but instead of servicing the bus stops at regular intervals, the passengers call the buses from the bus stops to order transportation. All in all the system is similar to a taxi based transportation system, where pickup locations and destinations are defined by the regular bus stops.

The Kaldebus project consisted of three phases of which the first phase was a description of the new transportation system as written in the previous paragraph. Phase two was the completion of a demand estimation, in which the attitude of the possible passengers toward the project was evaluated. Phase two was completed in 1996 with the result that with the right service parameters (Max. waiting time at a busstop 10 min., max. driving time between stops 25-30 min., and rate of fare 10 dkr.) there would be an increase in the number of passengers of about 30 % amounting to about 200 passengers per day. In phase two COWI also sets the framework for the practical testing of Kaldebus. However phase three consisting of the practical testing was never carried through for somewhat blurred economical reasons.

In 1991 BAT, a trafic operator on a Danish island, introduced an arrangement for handicapped that allowed them up to 160 trips by taxi a year. This arrangement became such a success that the rapidly increasing demand and rising costs led to a discussion of the possibility of introducing a public door to door trasportation system. This system would then be available to all passengers within the BAT operating area, with the purpose of economical and environmental benefits by operating big scale.

The discussion resulted in a combined public transportation system with both regular bus based transportation and a taxi based door-to-door (Dial-a-Ride) system called HandyBAT. HandyBAT was partly financed by the Danish State Department of Trafic and established as a joint work of handicap organisations, users, taxi operators, universities etc. The scale of HandyBAT was in 1995 about 350 users and the serviceparameters were 40 minutes at a maximum waiting time at pickup location, maximum 30 minutes early arrival at destination, maximum driving time 200% of shortest driving time or 35 minutes more than shortest driving time, and at least 20 min. longer than shortest driving time is acceptable. HandyBAT was the first operational computerized Dial-a-Ride transportation system in Denmark.

5.2.3 Knowledge center for public transportation in rural areas (CPTRA)

In 1999 the Minister of Trafic in Denmark launched a center for improving the public tranportation in low populated areas (CPTRA). The center was given 90 million dkr. to support projects within those areas in the following four years. The founding has been and is to be used on costs rising as a direct consequense of the adaption to new transportation systems, things such as hardware and software for using GPS in the route planning process in a dynamic transportation environment. CPTRA is also to become a knowledge bank for the various operators within this area, and the starting point consists of a report made by the Department of Trafic[52] in 1999.

Since the report made by the Department of Trafic is very important for the development of alternative public transportation systems in Denmark in near future, the following will aim to give a rough idea of some of the conclusions. As a start it is shown that the number of potential passengers in the rural areas is around 200.000 people. These people are all part of families, who do not have direct access to a car. Based on case studies with a varying degree of adaption to demand resposive transportation systems from regular systems, the conclusions in [52] follow.

As this is being written the new administration in Denmark has decided to close CPTRA. How this will affect already established initiatives is not yet known. However as decribed in section 5.2.4 CPTRA has already managed to help launching several initiatives, just as the transportation sector in Denmark now has a larger awareness towards the possible improvements found by restructuring part of the public transportation system.

In many cases it is possible to give a more extensive service within an unchanged budget by introducing Dial-a-Ride buses, though often it is nessesary to add regular school buses etc. to meet peek demand. Also it is somewhat difficult to integrate different types of transportation e.g. handicapped tranportation and regular bus transportation, since the passengers see theDial-a-Ride system as time demanding and not very flexible. To adapt it succesfully it is important to have extensive support among operators and the public service administration.

The following is a description of 10 smaller projects of introducing various new transportation systems in different parts of Denmark. Some of these are still operational and some were terminated after a limited period of time. A more thorough description can be found as a suplemnent to the report made by the Department of Trafic[52]. The small projects are very similar in structure, but since they show a usefull pattern they are all included here.

In Ravensborg a Totalbus (a bus performing all public transportation needs eg. transportation of people and goods such as food) was introduced in 1992. The bus had a regular schedule in the morning and afternoon rushhours transporting mainly school children to and from school. In between rush hour Totalbus performed Dial-a-Ride bus tranportation between busstops with possible deviations. Demand for transportation was given by telefon directly to the driver. In total 6 Totalbus lines were established; there is a

general consensus[1] among the population that it is now more convenient to live in the area. Also it has been succesful to have the buses deliver food and books for the elderly, but the delivery of medicine and other goods have not been used extensively. Overall the Totalbus project had a 25 % increase in the number of passengers over 3 years and still the total costs of running this improved system showed savings after 3 years.

In 1993 a Borgerbus (Danish for Citizens bus) was introduced in the area of Fasterholt-Kølkær. In 1995 Borgerbus, which was made a permanent part of the public tranportation system in the area, consists of a regular Dial-a-Ride bus system to supplement the existing system. About 10 % of the potential passengers used Borgerbus which has raised the level of service while being just about economically neutral.

On the same island HandyBAT, experimented with a better coordination of the local activities with the public transportation system. There was an overall increase in the number of passengers, but it is unsure if it was in any way connected with the coordination. Overall almost no passengers chose to use the buses on the special premisses.

In the area around the danish city of Brønderslev two regular bus routes where substituted with a Dial-a-Ride minibus, and two other routes were adapted to the new situation in order to handle school children. On weeknights and weekends a taxi was used as the Dial-a-Ride bus. The minibus followed a regular route with only exceptional deviations. Results shows that even though demand decreased by 90 passengers per month, 84 % of the passengers were happy with the new system. Costs decreased with the demand.

In Lejre the existing public transportation system consisted of two local bus routes and a closed school bus system. In the evenings a Dial-a-Ride bus was the only service, also it only covered the southern part of the area. This was changed to one regular bus route, 3 open (for tranportation of all possible passengers) school bus routes, and an all-day Dial-a-Ride bus covering the entire area.

[1]75 % agrees

In the beginning the Dial-a-Ride bus was operating independently of any regular lines, but since passengers found it inconvenient to have to call for the bus every time, the system was changed to a regular route based bus. However the bus still carried out door-to-door transportation, and it was still necessary to call if there was a need to go to destinations far from the regular route. Results indicated an increase in demand by 20-30 %, with overall passenger satisfaction. The opening of the school buses to accept other passengers did not provide any flexibility. There was neither increase nor decrease in finances.

The next project was carried out in the area around Nysted. Here the situation was somewhat different from the previous projects, since the public transportation system consisted of open school bus routes. To obtain a higher level of service for non school children, the school bus routes were closed to other passengers, and a Dial-a-Ride bus (Landsbybussen) was inserted. Landsbybussen had the primary goal of transporting the senior citizens to and from a regional center, with 3-4 daily departures, secondly it functioned as a regular open dial-a-ride bus. This lead to an increase in demand by 340 % (to 120 passengers per week) of satisfied passengers; however there was a budget increase of about 3 %.

One of the most successful projects was in Præstø. Here the regular bus routes were again transformed into door-to-door Dial-a-Ride buses. Except for small problems concerning the peak hours for school children, the passengers were very content with the new transportation system and wanted it extended. Also unique for this project was the apparent succes of integrating the various specialized transportation tasks[2] in the system. The demand was trippled to 50.000 passengers per year and at the same time the costs were decreased by 17 %. One of the primary reasons for the satisfaction among the passengers was the direct contact[3] between the operator/driver of the buses and the passengers.

In Kjellerup the situation was reversed when an attempt was made to increase the coverage of the public transportation system. A small bus was inserted to transverse the existing routes on peak

[2]Handicapped, seniors etc.

[3]Demand was telephoned directly to the driver

hours and assume the existing routes in low demand hours. The plan was to include the delivery of library books, food for elders, and other goods, but this was never implemented because of structural barriers[4]. Results showed no change in demand in spite of extensive marketing.

Inspired by the project in Præstø, a county consisting of a small island, transformed their seven local routes to two buses transporting school children to and from school and driving as door-to-door Dial-a-Ride buses in between. Eventhough the costs went up moderately, the demand did not increase. However as in Præstø, the passengers wanted the system extended.

5.2.4 Projects partly funded by CPTRA

Although the ten projects mentioned in the previous section 5.2.3 are the basis of the knowledge obtained by CPTRA, additional adaptions in the rural area of Denmark can be found. Keeping the above in mind, which follows is a more extensive description of recent developments partly founded by the new center. There is not yet a complete documentation of these developments, but a description of the centers actions so far can be found in the centers yearly report[17]. However the center hosted a small seminar on the improvement of public transportation in rural areas on which the following is based.

So far the center has supplied various areas with a total funding of twenty million dkr. Thirteen million has been divided between two large projects, the rest is divided among nine smaller projects like the ones already described. The 2 large projects are almost identical and operational as this is written. The main driving force in the first large project is Nordjyllands Trafikselskab (Traffic operator of northern Jutland) from now on reffered to as NT. NT has also inspired and participated in the second large project lead by Vestsjællands Trafikselskab (VT - Traffic operator of Western Sealand).

Before introducing the projects, the following list defines "specialized transportation", since it is treated independently of regular public transportation in the projects. Specialized tranportation

[4]Not mentioned in the report

consists of:

- Transportation of handicapped
- Transportation to/from medical care
- Other municipal transportation

Putting the regular scheduled public transportation aside for the moment, NT has developed a system for coordinating the specialized transportation within their comparatively large operations area. This system is based on a software module called Planet (more detailed describtion in chapter 7), which assigns passengers to vehicles. The goal of NT is to reduce the costs by 20-25 % in comparison to the standard transportation systems where each passenger is serviced by a regular taxi at the cost of the regular fare paid by the government. This reduction is anticipated as a direct consequense of the ability to fill up the vehicles and just as important as a consequense of using the taxis and minibuses based on contracts.

As illustrated in figure 5.2 the Planet system consists of a server performing the assignment of passengers to vehicles and a number of clients placed at a call center and at the various institutions such as hospitals, municipal offices etc. The server is designed with a large over capacity to ensure instant response time for the clients. Also the plan is to use Planet for regular Dial-a-Ride buses open to all in the future as a suplement or maybe even a replacement of the existing transportation system.
Planet is based on the following cycle:

- The transportation is offered to the various smaller operators eg. taxi companies, minibus companies.
- The operators asks a price per driven hour and a price per hour waiting.
- All prices are used as input in Planet eg. no company is refused.
- Planet assigns passengers to vehicles based on the prices of the vehicles and geographical positions 24 hours a day.
- The transportation is once again offered to the operators.

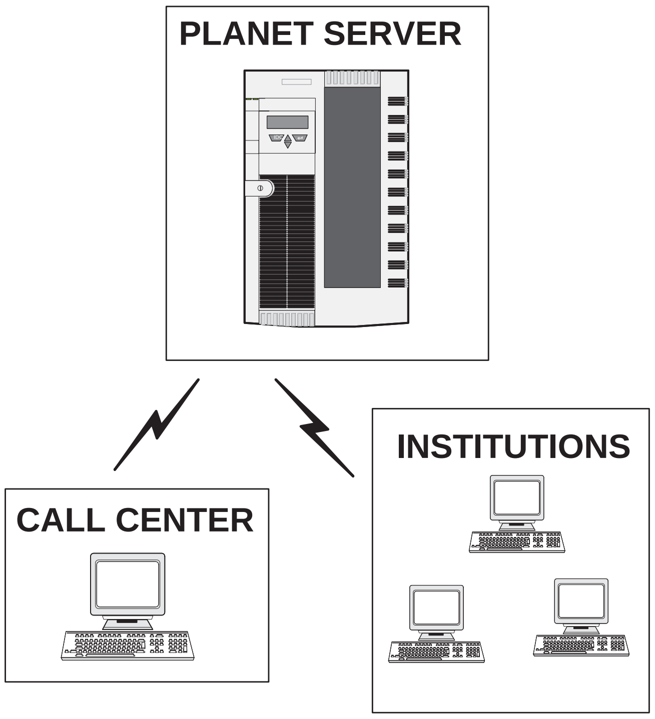


Figure 5.2: The representation of the total Planet system.

As this is written NT cooperates with around 100 operators with a total of about 500 vehicles, and by coordinating the transportation the total number of vehicles used has decreased. The least inexpensive vehicles are of course given most of the demand so by having more demand on fewer vehicles, the operators have the possibility of making a greater profit per vehicle, lowering the prices.

The use of Planet has resulted in reduced costs for passengers, who also gets a response on their order imediately on placing their demand. Also important information is automatically transferred to the destination (this is mainly for handicapped), and regular demand is taken care of automatically. Transportation demanded by hospitals, senior centers is also simplified resulting in less administration and more control over costs. It is now possible to make a larger profit per vehicle since they no longer have to keep track of the billing; Planet does this automatically.

There are of course also disadvantages by using a system such as Planet, and the experience at NT is that the passengers are not happy about having excess driving time or risking unnecessary waiting time, and most feel it is inconvenient to have to share the vehicle with others. Hospitals and other institutions miss the closeness with the patients just as they feel they depend too much on technology. For the operators it is harder to estimate prices and foresee outcome just as it brings more competition.

The results so far are savings on the transportation of patients of about 6,5 %, handicapped transportation 26 %, and other medically related transportation 25 %. However there has been increased administrative costs of a little more than half the saved amount. All in all total savings of 11 %. The experience gained by NT during the implementation of Planet points to various problems and traps. First of all the hardware must be designed so there is no down time. Also the personnel must be educated properly to make sure no knowledge is lost, and because the automated process does not handle everything, there is still the need for the human factor. Good will among the organizations, institutions and operators using the transportation system is also very important, and last, the implementation of the system is very expensive.

Based on all of the above, the following section will comment on the projects made so far as well as list a number of additional areas of investigation in the future.

5.2.5 Comments and comparison

What can be concluded from the projects mentioned in this chapter? Why is the adaption to a more flexible transportation system not always succesful? What still needs to be done to improve the public transportation in the less populated rural areas? Proposals to some of the answers will follow and hopefully help make the often costly but necessary transition to a higher level of service for citizens in these areas less problematic in the future.

Table 5.1 shows a comparison of the projects with regard to the parameters mentioned in section 5.2.1. The table displays the change in overall costs of the local transportation system after the adaption to a demand responsive model which is then compared with the change in the level of service of the traditional and specialized

Project	*Cost*	*Service*		*Goods*	*DAR*		*Reaction*
		Trad.	*Spec.*		*DtD*	*Stops*	
Ravnsborg	Down	Up	***	Yes	Yes	Yes	Better
Fasterholt-Kølkær	—	Up	Up	***	Yes	No	Better
Gudhjem	Up	Up	***	***	No	Yes	—
Brønderslev	Down	Down	***	***	No	Yes	—
Lejre	—	Up	***	***	Yes	Yes	Better
Nysted	Up	Up	Up	***	Yes	No	Better
Præstø	Down	Up	Up	***	Yes	No	Better
Kjellerup	***	***	Down	No	No	Yes	Worse
Møn	Up	Up	—	***	Yes	No	Better
Ringe	Up	Up	Down	***	Yes	Yes	—

Table 5.1: Evaluation of Danish projects

transports, also weather transportation of goods is included. The two Dial-a-Ride columns indicate the structure of the new system with true Door-to-Door transportation, regular stops, or a combination of both. The last column gives an indication of the passenger reaction after the adaption. "*" indicate that the feature was not included or information not available, and "—" is a symbol of no change from before.**

The table shows that in almost all instances it was possible to raise the level of service in order to achieve an increase both in number of passengers but also in passenger satisfaction. In about half these cases, the cost of operation was actually lowered. In general all projects where the adaption to DAR transportation was total (e.g. the regular transportation system was swapped to a pure DAR transportation system, the level of service always increased from the passengers point of view).

Keeping in mind the discussion of the service parameters from section 4.4, it is extremely valuable to be able to evaluate the cost connected to adjustning these parameters. For instance, how much worse would the driven routes be by decreasing the time interval between a request is given to when it is carried out, or how many more vehicles will be needed with a demand increase of 10 %? It would also be valuable to know where to place a vehicle waiting for the next request based on the probablilities of getting requests in

the various areas. A knowledge generating simulation environment would of course also provide a useful tool for automatic planning once the parameters are locked.

The projects tested various different combinations of transportation systems showing that what worked in one area did not work in another. This might be according to the density of the population e.g. low density calls for total door-to-door transportation, higher density calls for a combination of regular stops and door-to-door transportation and so on. Most areas also have a comparatively large municipal center or a railway station which takes care of the passage to and from the area. Again looking at the current demand should tell something about which transportation system has the highest probability of succes.

There is one focus point missing in the entire discussion of an improved public transportation system. The work leading to the establisment of the center of knowledge not only focused on the improvement of service but also on the amount of energy spent on transportation. It is reasonable to state that another goal of the adaption of public transportation is to decrease the total distance driven and the size of the buses used, which can be done by using the capacity of the buses efficiently. It might, however, be a good idea to include the planning of the driven routes in the assignment of passengers to operators. There are still many areas in Denmark where the entire transportation system, however small, is operated by one single operator. Maybe a route planning and visualization tool could save distance in those frameworks.

All projects agree that it must be convenient to submit a request to the operator, and that the personal contact between the operator and the passengers is vital. All of the above supports the need for a tool that helps planning the transportation without taking over the entire process. Most often automated systems do not plan nearly as well as can an experienced person with a lot of local knowledge. However it is always helpful to use tools that help structuralize and visualize the planning process, especially when the experienced person is not around. Somehow that local knowledge must be conserved, since it seems that flexible transportation systems based on personal contact and local knowledge are most likely to succeed.

There are basically two possibilities for the traffic units. Either they contract with smaller operators and perform the planning before relaying orders to the operators, or they outsource the system in large chunks to larger operators, who then plan the fulfillment of the orders themselves.

Chapter 6

The mathematical model

This chapter starts with definition of the notation used to describe the DARP in a mathematical model. It is a very complicated model, which can only be solved to optimality for very small instances. Thus we will not try to solve the model as a part of this thesis but instead the mathematical formulation will be used to help discuss the various aspects of the DARP. The formulation will also assist in later implementation of heuristics in which the objectives and constraints can be used as a guidance.

The formulated mathematical model is very similar to the one presented in chapter 9[1] concerning PDP, with some extensions taken from Baugh et al.[30]. Only a very few papers contain a mathematical model of the DARP. However in order to fully discuss the aspects of practical DARP, some time will be spend developing further extensions to the mathematical model.

6.1 Notation and basic model

First let us introduce the notation used to formulate the mathematical model. We have a set of n requests consisting of a pickup point i and a delivery point $n+i$ together with a demand d_i giving the number of passengers to be transported from i to $n+i$. The complete set of pickup points is denoted $P = \{1, ..., n\}$ and delivery points $D = \{n+1, ..., 2n\}$. Let $N = P \cup D$. For each vehicle $k \in K$ we have two nodes: An origin depot $o(k)$ and a destination depot $d(k)$. Let $A = N \cup \{o(k), d(k)\} \forall k \in K$. Since not all vehicles are necessarily used, let $V \subset K$ denote the set of vehicles used to obtain a solution, and v be the number of vehicles used. Each vehicle has a capacity C^k The problem consists of transporting d_i passengers from i to $n+i$, so the change in load at node i and $n+i$ is represented respectively by $l_i = d_i$ and $l_{n+i} = -d_i$. L_i^k is the load of vehicle k after serving node i. Each node i (both pickup and delivery) has to be served within a time window $[a_i, b_i]$. Let T_i^k be the starting time for service at node i on vehicle k. The driving time between nodes i and j is denoted $t_{i,j}$, and the service time at node i is s_i. We now introduce the binary decision variable $x_{i,j}^k$ with value 1 if vehicle k services node i and then drives staight to and services node j, otherwise $x_{i,j}^k$ has value 0.

When constructing the basic model, we first need to state some general assumptions concerning the DARP. Note that these assumptions derived from the problems seen in existing literature, from which we try to capture the most common characteristics.

- The objective is to minimize the total cost of transportation and to maximize the level of service provided to passengers.
- Vehicles start and end at a depot (not necessarily the same depot).
- Passengers must be picked up and delivered by the same vehicle.
- Time windows for pickup and delivery of passengers are given and must not be violated (hard time windows).

Keeping these items in mind, we can now construct a basic mathematical model starting with the objective function. Since we will consider both the cost of operation and the level of service, it seems clear that we have a multicriteria problem. This is in itself an extensive area of science which we will only address shortly here by considering two main procedures.

One procedure for solving a multicriteria problem is to prioritize the objectives and then solve the problem according to the objective with highest priority first and lowest priority last. This solution, however, does not seem reasonable in the case of DAR, since the desired level of service is dependent on the type of transportation (see section 1.1.3 and 4.4). In the case of a public transportation system, it is desireable to attract as many passengers

as possible while still controlling the cost of operation which also points towards a procedure for solving the multicriteria problem where the influence of the objectives can be controlled in great detail. Thus we will use the procedure where every objective in the multicriteria objective function is multiplied by a variable. This variable can then be set by the operator according to the desired effect of the objectives on the final solution.

We now consider what objectives to include in the objective function. To represent the cost of operation we will look at the number of vehicles used together with the total driving time. The actual cost is often calculated on the basis of bus-hours which of course also includes the service for passengers. However since the service time is fixed, it makes sense just to minimize the driving time of the vehicles. As shown in section 1.1.3 it can be very difficult to determine a proper objective with regards to level of service. It is, however, well known that the number of passengers in the public transportation system is closely related to the total transportation time for each customer which often is by far the most important factor to passengers. Using total time of transportation for each passenger, we can now formulate the multicriteria objective function:

$$min \ \ \alpha \sum_{k\in V} \sum_{(i,j)\in A} t_{i,j} x^k_{i,j} + \beta v + \gamma \sum_{k\in V} \sum_{i\in P} (T^k_{n+i} - s_i - T^k_i) \tag{6.1}$$

To decide the influence of each objective in the objective function we have introduced the multipliers α, β, and γ. Now we need to ensure that the number of vehicles leaving the depots is equal to the number of vehicles returning to the depots. This is done by adding the following two constraints:

$$\sum_{k\in V} \sum_{j\in P\cup d(k)} x^k_{o(k),j} = v \tag{6.2}$$

$$\sum_{k\in V} \sum_{i\in D\cup o(k)} x^k_{i,d(k)} = v \tag{6.3}$$

To be sure each customer is picked up and delivered only once and that both operations are carried out by the same vehicle, we add

the following constraints to the model:

$$\sum_{k\in V}\sum_{j\in A} x_{i,j}^k = 1 \quad , \quad \forall i \in N \tag{6.4}$$

$$\sum_{j\in N} x_{i,j}^k - \sum_{j\in N} x_{j,n+i}^k = 0 \quad , \quad \forall k \in V, i \in P \tag{6.5}$$

In order to obtain a feasible solution, we introduce a compatibility constraint which constraint states that in order for a vehicle to service nodes i and j in sequence, the time of arrival at node j must be larger than the time the vehicle leaves node i plus the driving time between nodes i and j:

$$x_{i,j}^k(T_i^k + s_i + t_{i,j} - T_j^k) \leq 0 \quad , \quad \forall k \in V, (i,j) \in A \tag{6.6}$$

As stated earlier in this section, we assume the time windows at nodes to be fixed and hard. This is modelled by a constraint requiring the start of service of vehicle k at node i to be within the specified time window:

$$a_i \leq T_i^k \leq b_i \quad , \quad \forall k \in V, i \in A \tag{6.7}$$

Not only does the pickup and delivery nodes of each customer have to be served by the same vehicle but also the pickup node must be visited by the vehicle before the delivery node, resulting in the precedence constraint:

$$T_i^k + s_i + t_{i,n+i} \leq T_{i+n}^k \quad , \quad \forall k \in V, i \in P \tag{6.8}$$

The number of passengers loaded at a pickup node must match the number of passengers unloaded at the corresponding delivery node. This is ensured by the following constraint, which states that when vehicle k has a load L_i^k after visiting node i, it must have a load $L_i^k + l_j$ after visiting node j:

$$x_{i,j}^k(L_i^k + l_j - L_j^k) = 0 \quad , \quad \forall k \in V, (i,j) \in A \tag{6.9}$$

To ensure the capacity of a vehicle is not exceeded at any point in time and that a vehicle after visiting a pickup node i has at least a load corresponding to the number of passengers picked up at i, we introduce the following capacity constraint:

$$l_i \leq L_i^k \leq C^k \quad , \quad \forall k \in V, i \in P \tag{6.10}$$

Every vehicle must start at a depot empty and return to a depot empty:

$$L^k_{o(k)} = L^k_{d(k)} = 0 \quad , \quad \forall k \in V \tag{6.11}$$

As stated earlier the decision variable $x^k_{i,j}$ is binary:

$$x^k_{i,j} \in \{0,1\} \quad , \quad \forall k \in V, (i,j) \in A \tag{6.12}$$

To further discuss the various aspects of DARP, the following is divided into subsections concerning respectively the objective function, the time window constraints, and the capacity constraints. These extensions are contributions solely of the author and based on experiences with practical problems. Before introducing the extensions to the mathematical formulation of the DARP, we summarize the basic formulation to be used in comparison with the extensions. Note also that T^k_i is a variable, thus making some of the constraints inequalities.

$$min \;\; \alpha \sum_{k\in V} \sum_{(i,j)\in A} t_{i,j} x^k_{i,j} + \beta v + \gamma \sum_{k \in V} \sum_{i \in P} (T^k_{n+i} - s_i - T^k_i) \tag{6.1}$$

subject to

$$\sum_{k\in V} \sum_{j \in P \cup d(k)} x^k_{o(k),j} = v \tag{6.2}$$

$$\sum_{k\in V} \sum_{i \in D \cup o(k)} x^k_{i,d(k)} = v \tag{6.3}$$

$$\sum_{k\in V} \sum_{j\in A} x^k_{i,j} = 1 \quad , \quad \forall i \in N \tag{6.4}$$

$$\sum_{j\in N} x^k_{i,j} - \sum_{j \in N} x^k_{j,n+i} = 0 \quad , \quad \forall k \in V, i \in P \tag{6.5}$$

$$x^k_{i,j}(T^k_i + s_i + t_{i,j} - T^k_j) \leq 0 \quad , \quad \forall k \in V, (i,j) \in A \tag{6.6}$$

$$a_i \leq T^k_i \leq b_i \quad , \quad \forall k \in V, i \in A \tag{6.7}$$

$$T^k_i + s_i + t_{i,n+i} \leq T^k_{i+n} \quad , \quad \forall k \in V, i \in P \tag{6.8}$$

$$x^k_{i,j}(L^k_i + l_j - L^k_j) = 0 \quad , \quad \forall k \in V, (i,j) \in A \tag{6.9}$$

$$l_i \leq L^k_i \leq C^k \quad , \quad \forall k \in V, i \in P \tag{6.10}$$

$$L^k_{o(k)} = L^k_{d(k)} = 0 \quad , \quad \forall k \in V \tag{6.11}$$

$$x^k_{i,j} \in \{0,1\} \quad , \quad \forall k \in V, (i,j) \in A \tag{6.12}$$

6.2 The objective function

This subsection will concentrate first on the problem of setting the multipliers α, β, and γ, since this is dependent on the structure of the problem as well as the problem size. Furthermore, we will discus possible additions to the objective function, to improve customer satisfaction with solutions produced by the mathematical model.

Possible savings when substituting a regular transportation system with a DAR transportation system can be divided into two catagories: Sequencing, and parallelization. Sequencing consists of scheduling neighboring customers to be transported one after the other thus minimizing empty driving. In this case neighboring customers are customers where the first customer destination is close to the second customer pickup point. Parallelization, on the other hand, is the ability to plan for having more than one group of customers in the vehicle at a time.

It is important to distinguish between sequencing and parallelization since they represent two different strategies, which are not always compatible. For instance, when transporting the severely disabled, it might be prudent to have only one customer in the vehicle at a time and then try to serialize customers. However when performing regular public transportation in a DAR transportation system, the number of passengers riding together in a vehicle should be as high as possible.

When problem size is relatively small when measured in number of customers, but the area covered by the DAR transportation system is relatively large, it seems logical to focus a great deal on minimizing the distance travelled. Often historical information can be used to calculate probabilities of where the origin of the next customer is going to be. When the vehicle is idle, it can be placed somewhere near the most probable next customer. In such DAR transportation systems, there is usually just a single vehicle (SV-DARP) or a fixed number of them, thus making the need to minimize number of vehicles unnecessary. Again there will often be only one customer in the vehicle at a time, meaning that the vehicle will perform close to a taxi service, which makes the need to minimize customer inconvenience unnecessary. In short there will be savings mostly because of sequencing.

As problem size increases, saving on parallelization also increases, making the objective of minimizing distance driven less important, and the objective of minimizing number of vehicles more important. In systems with a fixed number of vehicles, the second objective is equivalent to a maximization of accomodation, which means a maximization of the systems ability to accomodate new customers. When ensuring maximum accomodation, we look at minimizing the travel time for each customer in the system. Depending on how the time windows are determined (see 6.3), the travel time will always be within the allowed limit, or the 3rd part of the objective function will be changed to penalize the violation of travel time limit.

Many practical applications of DARP have shown that it might be prudent to add an extra term to the objective function, namely the minimization of a customer's time spent in a waiting vehicle which can be acomplished by introducing two new decision variables into the objective function: T^k_{0i} is the time a vehicle k arrives at node i, and T^k_{1i} is the time the vehicle k leaves node i. Now we have the following objective function:

$$\begin{aligned} min \;\; & \alpha \sum_{k \in V} \sum_{(i,j) \in A} t_{i,j} x^k_{i,j} + \beta v + \gamma \sum_{k \in V} \sum_{i \in P} (T^k_{n+i} - s_i - T^k_i) \\ & + \delta \sum_{k \in V} \sum_{i \in N} (T^k_{1i} - T^k_{0i} - s_i) \end{aligned} \tag{6.13}$$

This would also add the following two restrictions to the formulation to ensure that the vehicle arrives before starting service at node i, and leaves after servicing node i not later than the time window permits:

$$T^k_{0i} \leq T^k_i \quad , \quad \forall k \in V, \forall i \in N \tag{6.14}$$

$$T^k_{1i} \leq b_i + s_i \quad , \quad \forall k \in V, \forall i \in N \tag{6.15}$$

By introducing this to the formulation of DARP, we also minimize the time spend waiting at a customer location before a customer is served, and the time spend waiting in the vehicle after the customer is served until the vehicle moves on to the next service node.

The above formulation applies the same weight to waiting time for an empty and a full vehicle meaning that the formulation also tries to minimize vehicle idle time which might not be preferable, since it could result in a vehicle picking up a customer a little early and then waiting with the customer in the vehicle before servicing the next customer. To avoid this problem, the waiting time could be multiplied by the load of the vehicle to generate the objective function:

$$\begin{aligned} min\ \ \alpha \sum_{k \in V} \sum_{(i,j) \in A} t_{i,j} x^k_{i,j} + \beta v + \gamma \sum_{k \in V} \sum_{i \in P} (T^k_{n+i} - s_i - T^k_i) \\ + \delta \sum_{k \in V} \sum_{i \in N} ((T^k_i - T^k_{oi}) L^k_{i-1} - (T^k_{1i} - T^k_i - s_i) L^k_i) \end{aligned} \tag{6.16}$$

A positive effect of 6.16 is that a vehicle will wait untill the last possible moment before servicing a customer, since it is more "expensive" to wait after servicing the customer.

6.3 Time windows

The mathematical formulation of DARP described in section 6.1 is based on the assumption of hard time windows in which case, the total travel time is taken into account when determining the time windows for each request. There are two possible scenarios in which the customer specifies either an earliest time of pick up or a latest time of delivery. Then the operator specifies maximum excess travel time and maximum time window interval at destinations.

If excess travel time E is considered, the formulated mathematical model is not adequate. Until now excess travel time has only been formulated in the objective function, where total travel time for each customer is minimized. However by adding the following constraint 6.17 to the model, a limit can be set on excess travel time.

$$\sum_{i \in P} (T^k_{i+n} - t_{i,n+i} - s_i - T^k_i) \leq E \quad , \quad \forall k \in V \tag{6.17}$$

E is most often defined as a fraction of direct travel time. For example if E is set to 25% , a direct travel time of 1 hour would permit an excess travel time of 15 minutes. There are cases in

which a fixed excess travel time is set common to all customers. An example of this could be $E = 20minutes$ for all customers. Using E, the time window interval is set as the following shows. If earliest time of pickup is specified, the time interval at pickup is set at the pickup location to be a_i =the specified time, $b_i = a_i+$a fixed time interval. Likewise when latest time of arrival is specified, then b_{n+i} =the specified time and $a_{n+i} = b_{n+i}-$a fixed time interval. Now the time window in the opposite end of the trip (e.g. where nothing is specified by the customer) can be set according to two methods:

1 **If $\{a_i, b_i\}$ is set according to the specifications by the customer, then $a_{n+i} = a_i+t_{i,n+i}$ and $b_{n+i} = b_i+t_{i,n+i}+E$. The same principle is use for the situation where the latest arrival time i specified by the customer.**

2 **The time window interval length is fixed, so the time window where nothing is specified will need to be decision variables in the model.**

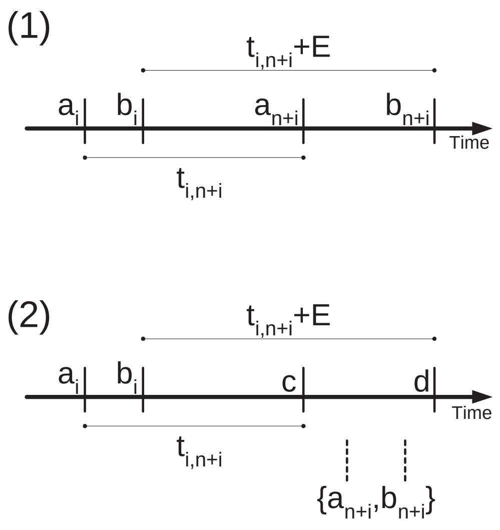


Figure 6.1: Setting the time windows according to the two principles.

The principle of the two methods are illustrated in figure 6.1. Considering (1) it is obvious that this is a very simple way of calculating the time windows. However it seems unreasonable that a time window interval increases with the excess traveltime of the customer. When earliest time of pickup is specified, this method is adequate,

but when latest time of arrival is specified, the time window for pickup can become unreasonably large. This model for calculating the time windows can still be used in systems in which the customers receive a notification before actual pickup.

Using the method where the unspecified time window becomes a decision variable (2) is much more problematic. First of all, the entire DARP must be solved in order to set the time window, whereas (1) only needs to make sure that a feasible solution can be found after adding the customer. Secondly the solution space can be severely reduced by setting time windows too tightly there by reducing the flexibility of the model to include more customers after fixing a time window in a previous solution. In the static case of DARP, where all requests are known in advance, the second method can be used followed by a notification to customers of when they can expect to be picked up.

6.4 Capacity

In the formulation used until now, each vehicle has a fixed capacity. However this is rarely the case in real life situations, where, for instance, a minibus can have room for a number of sitting customers, wheelchairs etc. Often different types of capacity can be substituted, so a wheelchair might be equivalent to two sitting customers, but the capacity for wheelchars is not equal to half the capacity for sitting customers. The simplest case of this exists with a fixed capacity in units (e.g. seats), and all other capacities can be expressed in those units. An example: A minibus has twenty seats. Ten of those seats can be substituted for 5 wheelchairs, or they can be used as regular seats. This simple case can be extended to $Q = \{1, ..., q\}$ different capacities. Let $q = 0$ represent the standard unit type (eg. seats), with total unit capacity of $C^k(0)$ for vehicle k. We now introduce this into the mathematical formulation as follows:

Let $l_i(q)$ be the change in load of type q at node i, and $L_i^k(q)$ be the load of type q in vehicle k after servicing node i. We now introduce a capacity substitution variable $p = \{p_0, ..., p_q\}$ where $p_0 = 1$, and p_q is the number of units substituted when loading one unit of type q. In the mathematical formulation we now replace 6.9, 6.10, and 6.11 with:

$$x_{i,j}^k(L_i^k(q) + l_j(q) - L_j^k(q)) = 0 \quad , \quad \forall k \in V, (i,j) \in A, q \in Q \tag{6.18}$$

$$l_i(q) \leq L_i^k(q) \leq C^k(q) \quad , \quad \forall k \in V, i \in P, q \in Q \tag{6.19}$$

$$L_{o(k)}^k(q) = L_{d(k)}^k(q) = 0 \quad , \quad \forall k \in V, q \in Q \tag{6.20}$$

$$\sum_{q \in Q} L_i^k(q) p_q \leq C^k(0) \quad , \quad \forall k \in V \tag{6.21}$$

Constraint 6.18 is similar to 6.9 with the addition that the type of capacity used when visiting stop i must match not only the quantity but also the type of capacity vacated at the corresponding stop j. The load of each type q on vehicle k after visiting a stop is controlled by 6.19, stating that this load must be less than or equal to the capacity of type q of vehicle k servicing the stop. It also states that the load of type q after leaving a stop must be less than or equal to the quantity loaded at the stop for the vehicle k. Constraint 6.20 insures that the vehicles start and end with an empty load of all types. The only new addition to the mathematical formulation is 6.21 where the various capacities are multiplied by the substitution number and summerized to ensure that the total capacity of the vehicle is not exceeded.

The above formulated extension of the capacity restrictions should in most cases cover practical problems, but there might be special cases where it is possible to substitute different capacities that cannot be converted to regular seats. In such cases, the mathematical formulation will be almost unreadable, since a second index would be needed on the substitution parameter, and a variable number of substitutions would be needed to ensure feasibility. However if this is of any interest, there will be an algorithmic formulation of this in chapter 10.

6.5 The extended mathematical formulation

To summarize, the extensions to the basic formulation of the DARP given in this chapter are repeated to give a full overview of the

resulting mathematical formulation.

$$min \;\; \alpha \sum_{k \in V} \sum_{(i,j) \in A} t_{i,j} x_{i,j}^k + \beta v + \gamma \sum_{k \in V} \sum_{i \in P} (T_{n+i}^k - s_i - T_i^k) \tag{6.16}$$
$$+\delta \sum_{k \in V} \sum_{i \in N} ((T_i^k - T_{oi}^k) L_{i-1}^k - (T_{1i}^k - T_i^k - s_i) L_i^k)$$

subject to

$$\sum_{k \in V} \sum_{j \in P \cup d(k)} x_{o(k),j}^k = v \tag{6.2}$$

$$\sum_{k \in V} \sum_{i \in D \cup o(k)} x_{i,d(k)}^k = v \tag{6.3}$$

$$\sum_{k \in V} \sum_{j \in A} x_{i,j}^k = 1 \quad , \quad \forall i \in N \tag{6.4}$$

$$\sum_{j \in N} x_{i,j}^k - \sum_{j \in N} x_{j,n+i}^k = 0 \quad , \quad \forall k \in V, i \in P \tag{6.5}$$

$$x_{i,j}^k (T_i^k + s_i + t_{i,j} - T_j^k) \leq 0 \quad , \quad \forall k \in V, (i,j) \in A \tag{6.6}$$

$$a_i \leq T_i^k \leq b_i \quad , \quad \forall k \in V, i \in A \tag{6.7}$$

$$T_i^k + s_i + t_{i,n+i} \leq T_{i+n}^k \quad , \quad \forall k \in V, i \in P \tag{6.8}$$

$$\sum_{i \in P} (T_{i+n}^k - t_{i,n+i} - s_i - T_i^k) \leq E \quad , \quad \forall k \in V \tag{6.17}$$

$$x_{i,j}^k (L_i^k(q) + l_j(q) - L_j^k(q)) = 0 \quad , \quad \forall k \in V, (i,j) \in A, q \in Q \tag{6.18}$$

$$l_i(q) \leq L_i^k(q) \leq C^k(q) \quad , \quad \forall k \in V, i \in P, q \in Q \tag{6.19}$$

$$L_{o(k)}^k(q) = L_{d(k)}^k(q) = 0 \quad , \quad \forall k \in V, q \in Q \tag{6.20}$$

$$\sum_{q \in Q} L_i^k(q) p_q \leq C^k(0) \quad , \quad \forall k \in V \tag{6.21}$$

$$x_{i,j}^k \in \{0,1\} \quad , \quad \forall k \in V, (i,j) \in A \tag{6.12}$$

Chapter 7

Existing software

Two commercially available software packages for solving dynamic DAR problems are briefly introduced in this chapter. The descriptions are based on information obtained at conferences from the companies and users of the products. This information is then paired with marketing material from the companies.

7.1 MobiRouter

The MobiRouter system is not a public transportation planning system as such, but rather a DRTS (Demand-responsive transport services) planning tool. The designers of MobiRouter view their system as perfectly able to compete with public transportation, both time- and costwise.

MobiRouter consists of a software package, which contains optimization software for planning the transportation. Communication between dispatcher and vehicles takes place using the regular cellular phone network. Each time an order is to be executed, the system describes the precise route the driver should use. This obviously creates the need for each driver to have an on-board computer, although examples in MobiRoutes information material seems to suggest that a regular cellular phone could suffice. This does not, however, seem to be the rule. When an order has been planned, the customer is then adviced by the dispatcher.

Although MobiRouter do not claim to be operating in real-time, they do say that a few minutes of advance warning is enough for them, which would suggest near real-time capabilities. At the same time, they state that their map-based route design works in real-time. Nothing else is mentioned concerning the possibility of manually manipulating routes.

MobiRouter especially focuses on three groups: The elderly, the disabled and school children. These groups, and their obvious special transportation needs are mentioned several times by MobiRouter as their main concern. They target municipalities as their main customers, since municipalities would have the overall responsibility for the welfare of the three groups.

Apart from the customer being able to phone in an order, the MobiRouter system is able to receive orders by both SMS and WAP.

7.2 Transmation

The Transmation system can be used both for the planning of DRTS and public transportation. Again, the software is the main focus. Transmation has both transportation planners and software designers as part of their design team. The system runs using the windows environment.

The main window of the product consists of orders not yet completed. In this window, orders can be planned both automatically and manually. Whole trips can be deleted or divided into several trips, as well as erased or added to other trips. All information from a customer should be received through one phone call. The details about pick-up etc. should be resolved at the same time.

The automation part of the system utilizes fuzzy logic as its main component. Any final decisions about a trip, however, is left to the dispatchers, allowing for both automatic and semiautomatic transportation planning. Various other functions, such as storing past transport data about a certain customer for later use are also included in the system. This has been done on the urging af various dispatchers cooperating with the design team. Another feature is the systems ability to coordinate with other public transportation, e.g. information concerning the public transportation network can be called up on demand by the system.

Several diagnostic tools are also included in the system. Graphs, tables etc of past transportation plans can be displayed when needed to provide statistics and further optimize the planning process.

7.3 Summary

Although there seems to be some differences between the two systems, they seem remarkably alike. Both deal with automatic planning. Both state that their customers receive fast service. MobiRouter has not mentioned anything about diagnostic tools, but from any software system statistics can always be obtained.

Setting all claims of cost-saving, environmental advantages etc. aside, what could probably set their systems wide apart in terms of performance are their build-in algorithms. MobiRouter have mentioned nothing of their methods. Transmation have mentioned using fuzzy logic, but not explained in-depth. From the material offered by both, it is impossible to draw any conclusions about their effectiveness, nor is an analysis of used algorithms within the scope of this paper.

Chapter 8

Shortest Path Problem - MLThreshX2

The Shortest Path Problem (SPP) is one of the most fundamental problems in Operations Research, in which it is used in numerous network applications. SPP in its simplest form seeks to find the shortest path between two points in a network while travelling only on arcs within that network.

The following will focus on calculating shortest paths on real-road networks, which is an application of SPP that is gaining an increasing amount of interest in todays research. The developement of ever more detailed digitized roadmaps in conjunction with the use of the Global Positioning System (GPS) demands fast algorithms.

As the level of detail on digitized roadmaps increase, the demand for more acurate and realistic solutions in SPP calculations becomes obvious. It is therefore necessary to look at new ways of performing SPP calculations in which the complexity of adding restrictions such as low bridge, no right turn, etc. to the road network has minimal impact on the performance of the algorithm.

8.1 Overview

There will be two main parts of this chapter: Part one focuses mainly on the algorithm to be used; Part two focuses on the design of the road network.

First there will be a short definition of the general SPP followed by a short introduction to the algorithm L-Thresh-X2 and the modifications to this algorithm that are implemented in ML-Thresh-X2. Then the design of the network will be adressed, and results of tests on various networks will be presented.

8.2 Definition and notation

Let us introduce a graph $G(N, E)$ where N is a set of nodes, and E is a set of arcs. The length of the arc between node i and j is denoted $d(i, j)$. The distances can be physical distance, time used to travel the arc, Euclidian distance etc. s is the given starting node. $d(s, i)$ is the upper bound on the shortest distance from s to i, $d^*(s, i)$ is the shortest distance from s to i, and $p(i)$ is the predecessor to node i. Now the problem consists of finding the shortest path between s and all other nodes in the graph, or:

$$\boxed{\text{Find } d^*(s,i) \text{ from node } s \text{ to all nodes } i \in N - s}$$

where the criteria of optimality is:

$$\boxed{\begin{array}{c} d^*(s,i) \text{ for all nodes } i \in N - s \\ \text{is found if and only if:} \\ d^*(s,j) \leq d^*(s,i) + d(i,j) \text{ for all arcs } (i,j) \in E \end{array}}$$

8.3 Choosing the right algorithm

When choosing an algorithm to use on real-road networks, the basic structure of the network needs to be considered. In Glover et al.[21] where the most widely used types of graphs are described it seems clear that a road-network is best described by either a random network or a transit grid network. Zhan & Noon[58] found that on real-road networks covering different states in the US, the basic threshold algorithm was amongst the 5 best performing algorithms. Also in Mondou et al.[34] the threshold algorithm showed promising results by being the overall fastest algorithm on computer generated grid and random networks.

Road-networks are always sparse networks (generally 2-4 arcs per node) but not homogeneously sparse (like most randomly generated networks). The fact that density usually rises with the level of population. This leads to the assumption that the algorithm needs to perform fairly well on graphs with a density within a certain interval.

The algorithm L-Thresh-X2 is described in Glover et al.[21], where it is shown that it is overall the best performing threshold algorithm. Andersen & Grejs[3] compared L-Thresh-X2 with Dijkstra's heap implementation described in Johnson[29] on a road network over the city of Aarhus in Denmark. They found that the L-Thresh-X2 performed significantly better than the Dijkstra with heap implementation.

Keeping all of the above in mind, the L-Thresh-X2 algorithm seems to be the most efficient algorithm to use on real-road networks with the European structure that is characterized by more or less random placement of roads.

8.4 L-Thresh-X2

L-Thresh-X2 is a modification of the original threshold algorithm Thresh-S developed by Glover, Klingman & Phillips in 1985[20]. The modifications are thoroughly described in Glover & Klingmann[21] thus they will not be addressed here.

L-Thresh-X2 consists of the following four steps:

Step 0. *Initialization:*
Initialize the predecessor $p(i) = 0$ and the distance label $d(s, i) = \infty$ for each node $i \in N$. Also create and initialize three mutually exclusive and collectively exhaustive sets of nodes called NOW, NOW', and $NEXT$. The sets are all initially empty. Set iteration count $k = 0$ and threshold value $t = 0$.

Step 1. *Select an Element of NOW:*
Elements in NOW and NOW' are selected in LIFO fashion (Last In First Out) as follows. If $NOW = \emptyset$ then goto Step 3, else get the node u that was last placed in NOW.

Step 2. *Scan selected node:*
Scan node u by examining each node $v \in \{v_1, ..., v_n\}$ in the

forward star E_u of u where $E_u = v : (u, v) \in E$ as follows. If $d(s, u)+d(u, v) < d(s, v)$ then $\{d(s, v) = d(s, u)+d(u, v)$ and $p(v) = u$ and if $(d(s, v) \leq t$ and $v \notin NOW$ and $v \notin NOW')$ then $\{$insert v at the end of NOW' and if $v \in NEXT$ then $NEXT = NEXT - \{v\}\}$ else if $v \notin NEXT$ then insert v in $NEXT\}$.

Step 3. *Repartition Scan Eligible Nodes:*
Start the next iteration or stop by performing the following actions: If $NOW' = \emptyset$ then $\{$if $NEXT = \emptyset$ then STOP else $\{k = k+1$ and recompute threshold value and select all nodes i from $NEXT$ where $d(s, i) \leq t$ and insert the selected nodes into NOW and goto Step 1$\}\}$ else $\{NOW = NOW'$ and $NOW' = \emptyset$ and goto Step 1$\}$.

Recompute Threshold Value. :
Calculate the new threshold value as:

$$t_k = MIN_k + TINC$$

where

$$MIN_k = \begin{cases} min\{d(s,i)|i \in NEXT\} & \textbf{when} \quad k = \{0,1\} \\ t_{k-1} + 1 & \textbf{when} \quad k = \{2,3,...\} \end{cases}$$

$$TINC = \begin{cases} [P * L_{MAX}] & \textbf{when} \quad DENSE \leq 7 \\ [P * L_{MAX}/(DENSE/7)] & \textbf{when} \quad DENSE > 7 \end{cases}$$

$$DENSE = min(35, [|E|/|N|])$$

$$L_{MAX} = \begin{cases} max\{d(i,j)|(i,j) \in E\} & \textbf{random and grid problems} \\ max\{d(i,j)|\textbf{grid arcs}\} & \textbf{transit grid problems} \end{cases}$$

$$P = \begin{cases} 0.25 & \textbf{for random problems} \\ 1.5 & \textbf{for grid problems} \\ 1.0 & \textbf{for transit grid problems} \end{cases}$$

The values for P are empirically found, and for the reasons described in section 8.3 we use the values for a random network when calculating the threshold value.

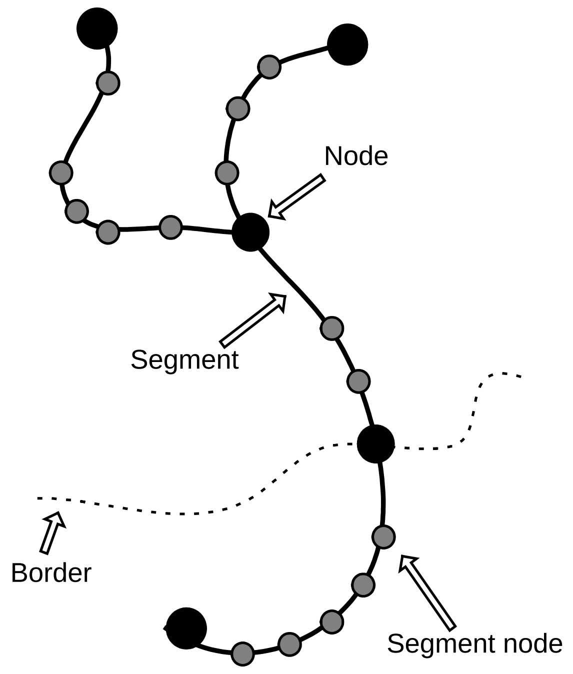


Figure 8.1: The representation of a real road.

8.5 Road network design

When digitized a road has the structure shown in figure 8.1. There is a node at every point where either two roads join together or where the attributes of the road change. A segmentnode is included to show the geographic positioning of the road, and the piece of road between two segmentpoints is called a segment. On figure 8.1 a border is indicated. This border could for instance represent a change in zip code which is a change in the road attributes which again results in a node, at which the road crosses the border. A link is defined as a part of the road existing between two adjacent nodes, so the road shown in figure 8.1 is represented by four links.

The traditional way of representing this structure is to construct a regular graph, which shows travel on the arcs between the different nodes and then updates various parameters at the nodes. The

problem with this representation arises when various restrictions are added to the network. For example with the "no right turn" restriction which one traditionally had to take into account by remembering the node one came from. By having to check where one came from every time he determines the forward star, the process of exploring the network slows down.

When exploring the possibilities of incorporating restrictions on the real road network into the sollution, it seems logical to use an approach that splits nodes affected by the restrictions into a number of artificial nodes. For instance in case of a "T"-crossing with a "no right turn", the node where the roads meet could be split up into 2 artificial nodes as seen in figure 8.2 (nodes 2 and 3). However it is in this representation still possible to travel nodes 2, 3a, 3b, and 6 and thereby make a right turn.

To avoid the problems mentioned above let us introduce a new way of representing the road network. Instead of concentrating on

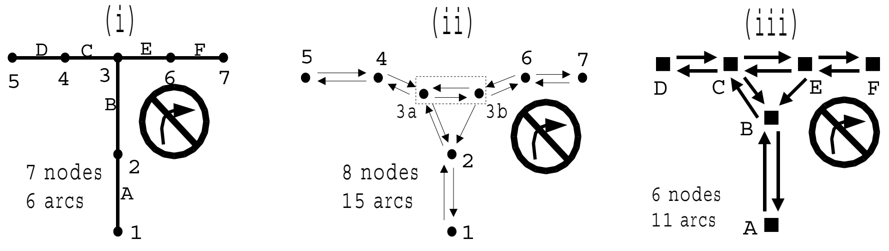


Figure 8.2: Handling "no right turn" restrictions. (i) is the regular network, where the letters A through F refers to (iii). (ii) is the split node structure, and (iii) is the new representation

nodes as the primary part of the network, we now direct the focus on the links between the nodes, and we convert these links to new nodes on which we store all relevant information, after which the old nodes are deleted. This gives us a new graph where the nodes represent the links (roads) and the arcs are the possible connections to other links. This is illustrated in figure 8.2 (i) and (iii), where A constitutes the path between nodes 1 and 2, B is the path between nodes 2 and 3 etc.

In order for the distances to be consistent with the original road network after the modification, the distances are now represented as follows. Since the new nodes are the old links, the old starting node s is now converted to a set of new starting nodes L, which consists of the set of old links beginning at the old node s. For each $l \in L$ we set the distance $d(s,l) = |l|$ where $|l|$ is the distance of the old link l. In short we travel the distances of the old links by visiting the new nodes.

Since the density[1] of a road network varies between two and four on the average, the number of nodes in the new graph is two to four times higher than the traditional representation. One could expect that the CPU time would increase when solving the shortest path problem on this new networkstructure. However this is not the case, since the number of possible shortest paths has not increased by the conversion.

In case of a "no right turn" restriction we can just omit the arc between the two new nodes as seen in figure 8.2 and thus decreases the size of the network which leads to a significant observation that the size of the network decreases with the introduction of restrictions on the road network and thereby speeds up the calculations instead of slowing them down. It is easily seen that this is also the case when adding other restrictions such as "one way".

8.6 ML-Thresh-X2

After having changed the structure of the network as mentioned in the previous section, it is necessary to modify the original algorithm. Also the recursive formulation of the algorithm turns out to cause problems when running on large networks. This is easily solved by reformulating the algorithm to be iterative, which is, of course, a transformation not depending on the underlying network design. Also the initialization step is omitted and the algoritms now uses dynamic initialization.

To avoid having to initialize the network before each calculation, the nodes are initialized dynamically by updating a label on the

[1]Number of links compared to number of nodes in the original graph

node describing in which iteration the node was last used. So if the node was not yet used in this iteration, it is initialized and otherwise nothing happens.

To convert the algorithm to be iterative the variable *state* is introduced. Then a superroutine running the various steps based on the *state* variable is constructed replacing Step 0. Step 1 is then modified to do nothing but control the *state* variable. In this way the *goto* statements are avoided, and the algorithm finishes each step before moving on. The procedure is described in more detail below.

The modification of the original algorithm to run on the new network structure consists mainly of changes to the initial new nodes to be examined and the updates on the distance labels. Since the starting node for the algorithm is an old type node, the modified algorithm uses a list of new nodes (arcs connected to the old node) as the definition of the starting point. In order to insure that all possible paths out of the starting point are explored. The new nodes in the list already have a distance different from 0 to the starting point, which is the length of the old arc constituing the new node.

Keeping the above modifications in mind, ML-Thresh-X2 now consists of the following four steps:

Step 0. *Superroutine:*
If the algorithm is run for the first time then create and initialize three mutually exclusive and colloctively exhaustive sets of nodes called NOW, NOW', and $NEXT$. The sets are all initially empty. Set iteration count $k = 0$ and threshold value $t = 0$. Otherwise set increase iterationcount by 1. In every case set threshold value and state variable to 0. Run the following loop: While $stop = false$ {run Step 1; if $state = 2$ then run Step 2 else run Step 3}.

Step 1. *Investigate NOW:*
If $NOW = \emptyset$ then $state = 3$ else $state = 2$.

Step 2. *Scan successors:*
Select u from NOW. Do the original step, but remember to initialize nodes used in previous calculation.

Step 3. *Move nodes or stop:*

Start the next iteration or stop by performing the following actions: If $NOW' = \emptyset$ then {if $NEXT = \emptyset$ then $stop = true$ else {$k = k + 1$ and recompute threshold value and select all nodes i from $NEXT$ where $d(i) \leq t$ and insert the selected nodes into NOW}} else {$NOW = NOW'$ and $NOW' = \emptyset$}.

There is no change in the way the threshold value is calculated.

It is important to remember that the new network structure changes the handling of distances and starting points as previously described in this section.

8.7 Implementation

To get optimal performance from the implementation of the algorithm, it is important to consider how the movement of elements between the sets is implemented. Also the algorithm needs to lookup in which set the element is placed in practically every step, which can be very time consuming if it is performed by a regular search procedure. The last important factors are to insert and erase elements from the sets. The implementation of this can have a tremendous impact on the speed of the algorithm.

The algorithm is implemented in Visual C++ using the containers found in the Standard Template Libarie (STL) for storing the sets. Using an object oriented approach insures that the algorithm is encapsulated in a way that makes it relatively simple to switch algorithms when new ones are developed without having to change the surrounding administrative source code.

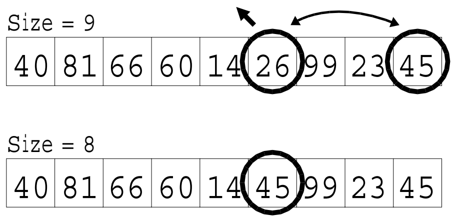


Figure 8.3: Erasing elements from $NEXT$.

When inserting or erasing elements to and from arrays, vectors etc. it leads to a memory allocation or deallocation which is time consuming. For this reason the set $NEXT$, which is implemented as a vector, is given a variable that contains the size of the set (not the vector). Whenever there is a need to erase an element (object) from the vector, an operation as shown in figure 8.3 is performed. The element to be erased is read or copied from the vector, and then the last element in the set is copied to its place, and the size of the set is decreased by 1. In this way it is not necessary to deallocate memory, just as it is not necessary to allocate space next time an element is inserted.

Another way of bringing down the time spent on memory handling is the allocation (reservation) of memory in the first initialization phase. Knowing the approximate size of the road network, it is possible to decide on the memory to be allocated from start, which is sensible since time spend on allocation of memory increases much less than linear with the amount of memory to be allocated.

As each element is implemented as an object, the element keeps track of its position in the various vectors itself, thus avoiding a search procedure whenever an element has to be found. When knowing the id of the element (or by using a pointer to the element), the placement and position in the vectors are also known.

8.8 Results

Test of the algorithm were performed on several real road networks from Scandinavia on a Compaq Deskpro Pentium II 300 Mhz MMX with 6 GB Ultra DMA harddisk and 288 MB RAM running Windows NT 4.0 workstation. The results are shown in table 8.1.

There are no "one way" or "no turn" restrictions on the networks, since this information was not available at the time of the tests. Also it will be noted that the number of nodes and arcs referred in the table are based on the original network structure.

The run times for ML-Thresh-X2 constitutes a summary of 100 runs on the networks. These 100 runs are based on randomly se-

Network name	Characteristics			CPU-times (sec.)		
	# nodes	# arcs	Density	Best	Avg.	Worst
Randers	3,983	9,942	2.50	0.01	0.02	0.04
Aarhus	14,392	34,822	2.42	0.08	0.09	0.11
Kbh97	46,820	116,564	2.49	0.30	0.32	0.34
Kms-D200-96	51,405	144,042	2.80	0.39	0.41	0.42
Kms-vejg-96	52,447	149,314	2.85	0.42	0.44	0.48
KmsAarhus95	63,171	171,834	2.72	0.48	0.51	0.55
Oslo98	74,970	172,814	2.31	0.39	0.45	0.52
KmsKbh97	86,273	227,522	2.64	0.65	0.68	0.72
Sverige	97,212	228,286	2.35	0.61	0.67	0.89

Table 8.1: Results from tests on various Scandinavian networks. All solution times are in seconds.

lected starting nodes, from which the entire shortest path tree is calculated which means that after one run the shortest path from the starting node to all other nodes in the network is known and can be returned. The runtimes in table 8.1 are shown as seconds to calculate the intire shortest path tree from one node in the network to all other nodes.

Note that the CPU-time spend on reading data and constructing the network in memory is not included in the results above, since this task is only performed at startup. Also in the above runs, no resulting trace (path) from the calculated shortest path tree is returned. In figure 8.4 the average CPU-times for ML-Thresh-X2 is shown as a function of the number of nodes in the original network. It is seen that the CPU-time is approximately linear the number of nodes. This is especially the case with the Danish[2] networks, whereas deviations occur when running the algorithm on networks such as "Oslo98" and "Sverige". A reason for these deviations could be that the density of the networks changes when in a different areas. An example is the "KmsAarhus95" which has fewer nodes and links than "Oslo98", but with a density that is 18% higher.

Since the algorithm is transformed to deal with the links, not caring about the nodes, it might be interesting to look at the CPU-time as a function of the number of links shown in figure 8.5. In this

[2] All networks except for "Oslo98" and "Sverige"

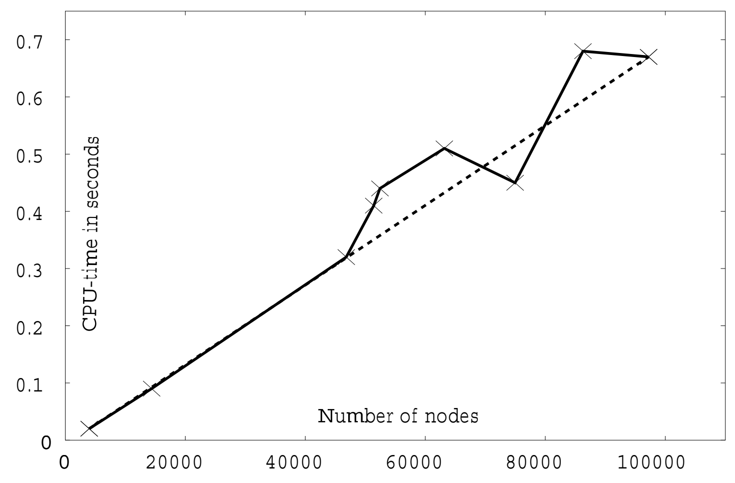


Figure 8.4: CPU-time as function of the number of nodes.

case it seems even more obvious that the CPU-time is linear in the network size. In practical application however, it should not matter whether the CPU-time is depending on number of nodes or links, since these two numbers on these types of networks are proportional.

Based on the results shown above, it is not safe to conclude that CPU-time is linear with respect to network size, since none of the tests were carried out on really large networks. The total road network of Denmark for instance has 500,000 nodes and approximately 1,000,000 arcs. Experiments on the total road network of Denmark have been carried out elsewhere and based on the results obtained there, the CPU-time can still be assumed linear with respect to network size.

8.9 Future work

As the need for fast calculation of shortest paths is vital to the incorporation of real road networks in Vehicle Routing Problems, the following will address some possibilities not yet explored for speeding up the performance.

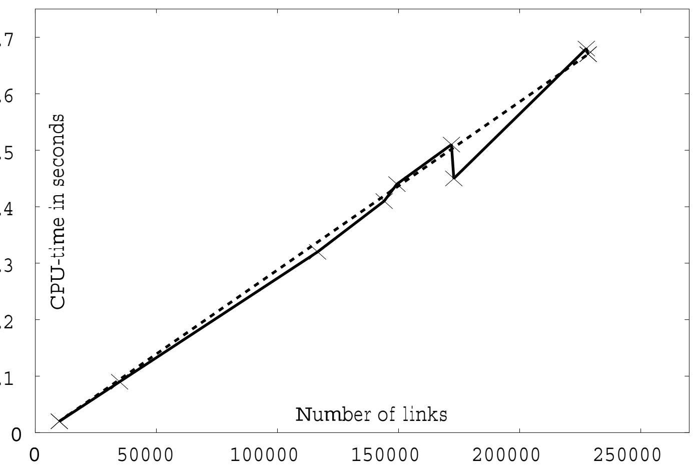


Figure 8.5: CPU-time as function of the number of links.

The most obvious and logical way of improving the performance of the algorithm is to introduce of a stop criterion. Since many applications using shortest path calculations include customers and depots located very close on the network, it seems a waste to calculate the full shortest path tree in every instance. It will, however, be inefficient to check if the required path is found every time a new link is to be examined, which will also require the use of a label setting algorithm. Instead when using ML-Thresh-X2, a check could be performed each time a new threshold value is to be calculated, since it is easily seen that all shortest paths within the radius of the previous threshold value will be the optimal shortest paths.

The placing of stop criteria in the manner described above can also be of interest when looking at the calculation of the threshold value. If an upper bound for the distance is known, it would seem logical not to let the threshold value exceed this limit.

Another way of speeding up the process of calculating the shortest path is network reduction. One way of doing this, is to omit the nodes dividing a road because of change in attributes. In these instances it is then possible to join two links thereby reducing network size. Also removing links that are outside the circumferential

circle of all customers, depots etc., might reduce the CPU-time spent in the last iteration of the calculation.

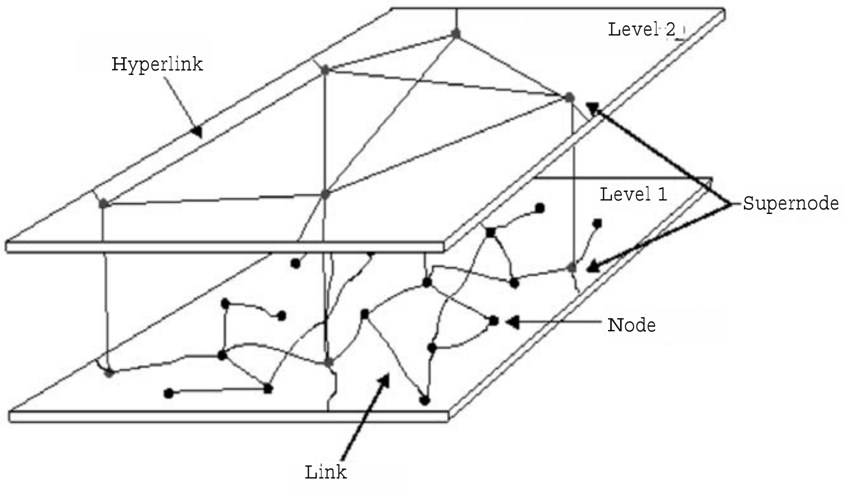


Figure 8.6: Modifying the network by introducing levels.

When using very large road networks as for instance a map of all Europe, it will probably not be efficient to perform calculations on the entire road network. A better idea might be to introduce various network levels. Figure 8.6 illustrates two levels of the same road network, where level 1 is the regular digitized network. The idea is that the shortest path will be calculated locally around destinations and globally on a higher level.

Of course it can not be guaranteed that the optimal path is found when some of the calculations are performed on a higher level, but over large distances the inaccuracy might not be significant. A large portion of the work needed for implementing a level strategy would be examining the areas and distances required if such a solution is valid.

To construct the various levels, it is necessary to explore the network to find the supernodes, as these will decide the link structure of the upper levels. In case of a problem with many regular and well known destinations, an upper level could be constructed with the placement of these customers as supernodes. However if desti-

nations are changing rapidly in time, it might be better to gather data from solutions to seek out the most visited nodes in the network, and then construct an upper level based on that knowledge.

Chapter 9

InfoRoute

InfoRoute is the administrative environment in which the DAR algorithm is to function. It consists of a database and a GUI coupled with a road network module and the shortes path algorithm. The GUI is developed with Danish toolbars and menus, but a description in English will complement the screenshots.

9.1 Database design

To begin construction of the InfoRoute program, a database must be designed. Since this database will lay the foundation for all of the work to come later, as well as ensuring the usefulness of the end product, a lot of work will have to be put into the design. The purpose of the design process is to create a standard, which itself can be used for virtually all sorts of transportation planning. With this in mind, a series of demands are to be made for the design of the database.

First and foremost, the number of variables must be kept at a minimum, while still ensuring the usefulness of the database. This means, that without altering the database, it should be possible to use it for anything from the transportation of people to transporting goods, from oil delivery to container transport etc. It is obviously a difficult task to get all of the necessary data crammed into a handfull of parameters. On the other hand, using more parameters than is strictly needed will make the database to complex, which again stand a good chance of compromising the demand for the database to be of general use as a tool for transportation planning.

Another demand, this one also derived from the need for simplicity, is the demand that data inserted into the database should not, or should only in rare cases, be present at more than one place. This is both an advantage because it keeps the size of the database at a minimum, but in addition to this, a more compact database (as a general rule) makes the processing of inserted data faster.

Yet another consequence of the demand for a general tool is the fact, that all table- and parameternames are in english, despite that the product is to be used, at least during testing, by a danish firm (Jensens turistfart). Thus, the names in the database and the names used in the GUI are not necessarily identical.

As may be guessed from the fact mentioned above, the final demand is that the database can be completely separated from the GUI and a new GUI can be designed and implemented, but still be able to use the old database. This ensures the longevity of the database, ie. a user will not have to reinsert new customer data just because he wishes to modernise his GUI. By having this last demand, virtually anyone should be able to place a new GUI on top of the "old" database.

Since there are so many different transportation algorithms that works in many different ways, there is not one single way to represent their recquired input. However, no developer needs to use all of the tables described below, as this would violate the demand for a general tool. No problems will arise from leaving some of the tables unused.

The input for the program/database are arranged in the tables described in the following sections.

9.1.1 Customer

Customer is used for storing information about single customers, thus making the entire table into a customer filing system. Customer information is identified by a customer number, id. In addition to this, the table contains general information about the customer, such as name, address, telephone number etc. These pieces of information are indexed to obtain fast access to every customer, no matter what information is used to search for him. The table furthermore contains the coordinates for the customers residence. In the case of the InfoRoute program, these will not have to be entered by the user, but will be calculated automatically and inserted into the table.

Following the coordinate slot, the e-mail address of the customer can be entered. Finally, there is a field for special notes that one way or the other do not fit into the other slots. This might be special information about restricted access to the customers house or any other information that might be important to know for the driver or the recipient of the transportation request.

This customer database not only simplifies the process of transporting the customer to or from his residence by allowing the user to simply transfer the information to the request database (described later). It is also vital for companies that need to bill their clients after transportation of either the client himself or goods, that the client has ordered. In the tasks covered by this project (telebusses), payment falls instantly, so this aspect of the costumer database is not strictly necessary. However, in order to ensure the general usefulness of the database as a tool for transportation planning, it was included anyway.

9.1.2 Request

Request is used for the storing of information concerning each request. It comprises a unique id once more, this time to identify the request, as well as a reference to the customer id. This prevents the need for entering any customer information again. It also contains the time, that the order was received. This can later on be used as a tool to simulate the processing of an order to determine whether or not this process can be improved upon.

Among other things, this will make it possible to examine how the much of an effect a long or short warning has on the final solution. The data can also be used to simulate the behaviour of a customer in the system. Another possibility is, if the transport company wishes to do so, it can charge customers different amounts for the same transportation, based on how much warning they were given, as express orders tend to be more costly, since optimum solutions are difficult to obtain when planning is rushed. Finally, this data can be used to evaluate the behaviour and effectiveness of the underlying algorithms.

Apart from this, Request contains the deadline for the completion of the order, and its start- and destination address. These consists of a single text field, as the address in this case is the combination of both the roadname and number. This is to allow addresses to be located in the address database.

The addition of letters to the road numbers are not registered in the address database. They are therefore placed in a separate parameter. The sole reason for their inclusion is the necessity for a driver to have access to the correct address. The ZIP codes of the start- and destination addresses are also to find in the Request table. They are indexed to allow a user to look up all destinations within a given Zip code area.

InfoRoute will insert the coordinates of the start and destination adresses after they are entered. They are used by the underlying algorithm to find a shortest path.

Requests are also given a priority, represented by a positive integer. The lower this value, the higher the priority of carrying out the request. Priorities only become useful when the supplier is unable to satisfy all requests for transportation and has to select which orders can be delayed or cancelled. It is obviously not a situation that should arise to often. One of the tools that can be used to avoid the situation is the possibility for the supplier to accept or turn down an order at the point of receiving it by using a fast algorithm that will seek to approximate the final result. Still, this is no guaranty of the situation never occuring. Furthermore, as mentioned before, the database should be useful for later use by other programs. Thats the reason for the inclusion of priority in the design.

The order type determines which sort of transportation is requested within the scope of possibilities, that the supplier offers. For example, a petroleum company will offer various kinds of petrochemical products to be delivered to the customer, while for a company specialising in transporting people this parameter might represent whether or not special measures need to to taken to accomodate the transportation of disabled people etc.

Finally the amount or number of goods or people to be transported is registered. Request, like Customer, also contains room for special notes, that for some reason or another will not fit into the other categories.

9.1.3 RepeatRequest and RepeatRequestType

RepeatRequest is rather special, as it contains all of the same variables as Request plus an additional one called repeattype. RepeatRequest is to be used for the automated generation of orders based on deals, that the customer and suppliers arrange. In the transportation industry, it is often the case that orders are generated in this way, rather than the customer having to place each and every order manually. The methods used for generating orders are many and varied. There exists the plain interval order, where a certain amount of a product is delivered at fixed intervals. But rather more complicated systems can also be found, for example the degree day system used by the oil industry, in which each chronological day is given a certain degree day value, based on the temperature on that day. A customers use of oil is then calculated from the total amount of degree days since he last received a shipment of oil. The oil company then uses the system to determine when it is necessary to send a truck to refill the customers tank. In other parts of the transportation industry other, more or less elaborate systems, are used for generating ordrers.

For the reasons mentioned above, the database contains the possibility of defining various ways of automatic generation of orders. These methods are represented in RepeatRequestType by an integer. It is then up to the user (supplier) to define the different methods and numbering them. The end result is, that the program should look for the method of generation, generate a new order and place it in Request, just as if the customer himself had placed the order manually. In InfoRoute, however, no such generators will be implemented at this time. They must be developed specifically for each user, according to his needs.

9.1.4 TypeRH and TypeRV

When a transportation task is carried out, not all of the time spend is used for driving. At both the beginning and the end of the task, a certain amount of time used for handling the goods or passengers must be added to get the total transportation time. The table TypeRH is used to store these extra amounts of time.

TypeRV describes the vehicle type with respect to which sorts of goods and/or passengers the vehicle can transport. In addition to this, there is a vehicle number which refers directly to the table Vehicle. TypeRV also contains the maximum capacity of a vehicle. There is, however, no key for the exchange of different sorts of goods. Instead, this key can be found in the Table PayloadSubstitution.

9.1.5 Vehicle and Depot

Vehicle is used to register vehicles in a fleet. Each vehicle is given an id-number. To each number is attached a description of the vehicle, a vehicle type and the home depot of the vehicle. There is also a number to indicate whether or not a vehicle is subject to restrictions regarding the types of tasks it can undertake. For example, most trucks are not able to pass under low bridges or negotiate dirt roads etc. This can be registered by this number.

Depot contains information about the placement of a depot with both an address and coordinates. Depot also conatins a note field. This information is attached to a depot number, the same one that is used in Vehicle to determine the vehicles home depot. Information about depots is included in order to calculate what vehicles are to undertake what tasks, and to calculate the total time consumption of a plan, as the time to drive to and from the depot must be included.

9.1.6 ConversionTable

ConversionTable determines whether or not a type of goods is interchangeable with another. This may affect both the planning and execution of a working day. The ratio is used to either determine the remaining capacity of a vehicle or to find out if a product can replace another product and, if this is the case, what the cost will be. In the case of transporting people, it is quite relevant to use this table in connection with the transportation of disabled people. One person in a wheelchair takes up the same amount of space as two walking passengers, giving a two to one conversion ratio in this case. It must be added, though, that this in no way means that the program will (or should) prioritise healthy passengers above disabled ones (healthy passengers pay the same fare as the disabled). The problem is handled by simply giving the transportation of disabled people a higher priority.

Another use for this table is when a supplier transports different, but downwards interchangeable products, meaning that the most expensive product will satisfy most or all customers, the slightly cheaper product fewer people etc. right down to the cheapest product, which will only be accepted by those, who specifically ordered it (to save money). When products are downwards interchangeable, it can sometimes be profitable to sell an expensive product as if it was a cheaper one, since the loss in sales profit can happen be less than the increased cost of transportation to go home and get the cheaper product. Of course, this sort of downwriting only occurs if there are sudden changes in the middle of a working day.

Just as the input for an algorithm depends completely on the structure of the algorith, so does the output. In the case of this database, it has been decided to make the output tables extremely generel. In practical terms, this means that there is no single table from which a complete result or solution to a problem can be read. Rather, it has been sought to ensure a developer or programmer complete freedom to arrange a result as he wishes by means of his own application. This also means, that he will be able to best present the data to reflect the behaviour and results of his specific algorithm.

The following three sections describes table containing all the data necessary to represent of a solution.

9.1.7 SolRequest

SolRequest displays how the execution of each individual order is carried out by linking the order id with the planned start and finishing time of the order and the addition of the time spend loading and unloading. This partial solution is then, as most other tables, given a unique id number. The solution is of course affected by the fact, that during the time between loading and unloading of an order, other orders may be partially or completely executed.

This table allows a developer to pick out and present the various orders, arranging them as he prefers them to present his solution.

9.1.8 SolRoutes

SolRoutes is an overview, which displays the total amount of time spent on a working day, identifiable by a number once more. This information can be transferred directly to the application and used for simulating the application/algorithms behaviour, thus allowing the developer to make improvements on the system, should he wish to do so.

9.1.9 SolVehicle

Finally, SolVehicle ties vehicles to tours using the id of the vehicle and tour, respectively. Together with the two other output tables mentioned above, this allows for a complete solution to be constructed.

As mentioned earlier, this solution must be arranges by the developer using his own application. Following this, he can choose to let his application generate plans for each tour, a complete tourplan, or any other arrangement that the developer wishes. The only important thing to remember here is, that despite the apparent lack of a coherent plan in the tbales, all necessary information for a solution is present.

As has just been described, the structure of the database is easily grasped, it is simple and not very large. Still, this does in no way impair the usefulnes of the database, as the overall demands for simplicity and generel use makes the database easy to use later on.

9.1.10 Overview

Figure 9.1 Shows the tables of the database and displays their internal relations. It is a good indicator as to the simplicity of its design.

The figure contains the abbreviations PK for Primary Key, FK for Foreign Key and I for Indexed. A primary key is used to identify a table. A foreign key is the primary key of one table, referred to in another table. Indexed means, that the content of a table is arranged in a search tree, thus making a specifik part easy to find.

9.2 User interface

InfoRoute is designed for use within the Windows environment. This makes it a graphically oriented application. This, in turn, makes working with InfoRoute a pleasant experience, as most functions can be understood intuitively. The build-in Shortest Path algorithm makes finding the best route between to addresses easy. Furthermore, the locations of addresses can be found and both addresses and routes can be marked on a map. It is possible to select and deselect layers on the map, including the road network (although this arguably makes the map somewhat dull to look at), municipal borders, marked addresses and marked routes.

The following section will make the user proficient in the use of InfoRoute by explaining how to open the program, create customers and orders, mainpulate map layers and exploit all the other possibilities, that InfoRoute offers.

9.2.1 Starting InfoRoute and Using the Toolbars

Upon opening InfoRoute (By finding the file "InfoRoute.exe" in the folder, to which you copied InfoRoute), the first screen looks as seen in figure 9.2.

The central part of the program window is occupied by a map. Right after opening InfoRoute, the map displayed will be the Skovbo_utm32 (a map covering Skovbo municipal district, generated by data using the UTM32 standard). The map can be closed at

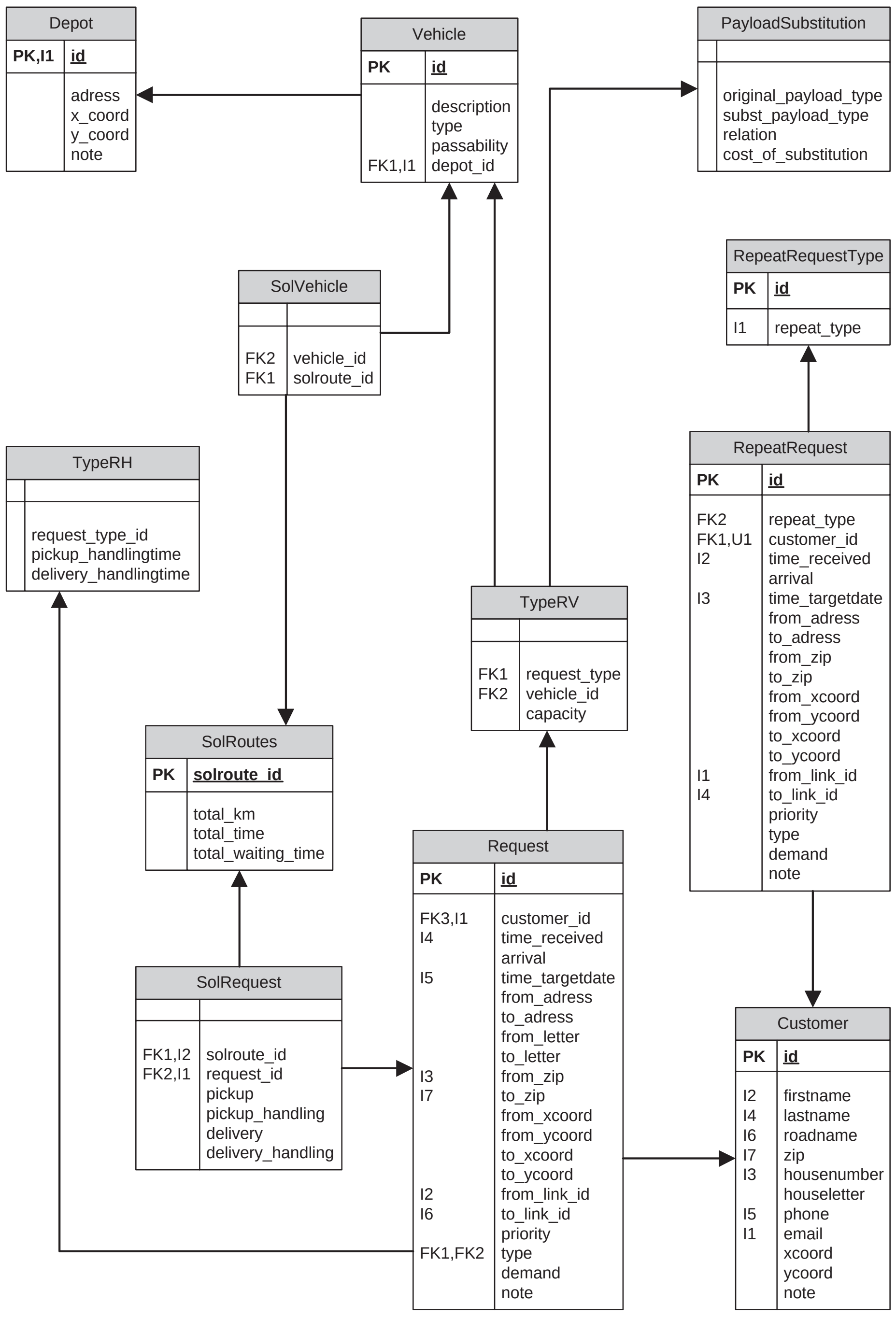


Figure 9.1: An overview of the internal relations of the database.

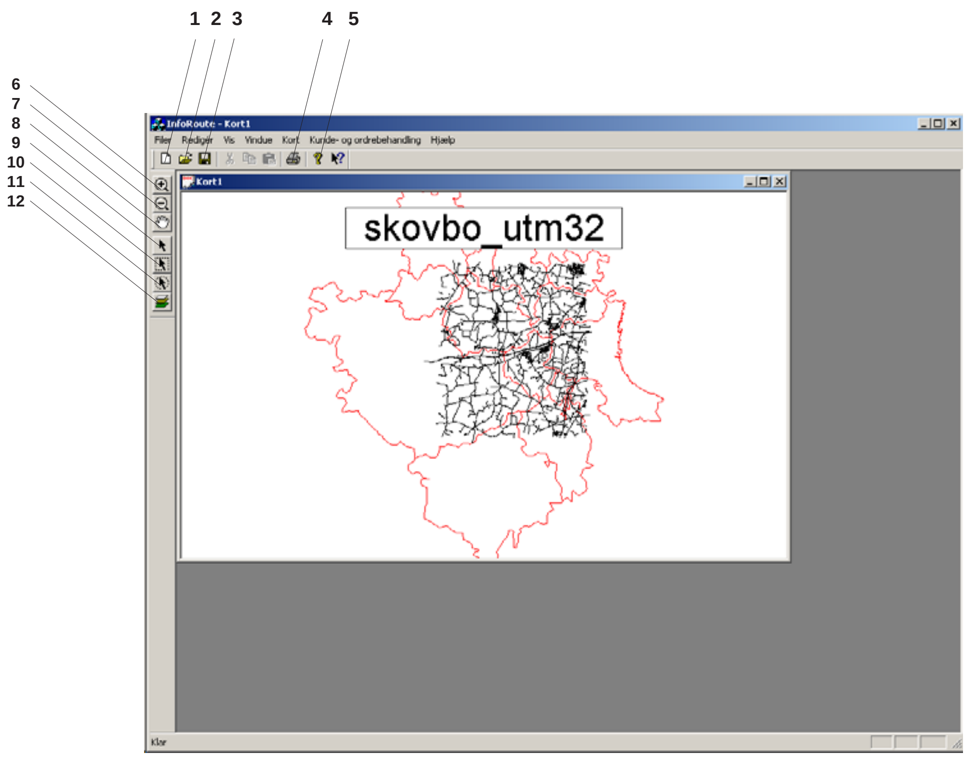

1 2 3
4 5
6
7
8
9
10
11
12
InfoRoute - Kort1
Filer Rediger Vis Vindue Kort Kunde- og ordrebehandling Hjælp
Kort1
skovbo_utm32
Klar

Figure 9.2: InfoRoutes appearence immmediately after it is opened

all times by clicking the usual icon (the "x" in the top right corner).

As is apparent on the picture, there are two toolbars present, one horizontal and the other vertical. On the picture, the icons on the horizontal toolbar have the numbers 1-5 attached to them, while the vertical toolbar icons are numbered 6-12. The numbered icons will be briefly explained here (The icons not numbered are inactive)

1. Open new map
2. Get map from disk
3. Save map to disk
4. Print currently active map
5. Information concerning the program's version
6. Zoom in

7. Zoom out
8. Move map
9. Select part of the map (by clicking on it)
10. Select parts of the map (by drawing a box around the wanted parts)
11. Select parts of the map (by drawing a circle around the wanted parts)
12. Pick layers on the map

Several of the functions mentioned above are also available as menu items (the menus are located just above the horizontal toolbar). The purpose of the toolbars is simply to make access to the most commonly used features in the program easier.

9.2.2 The Horizontal Toolbar

The icons 1-3, also accessible through the menu item “Filer” (Eng.: Files) - “Åbn” (Eng.: open), “hent” (Eng.: get) and “gem” (Eng.: save), respectively, are used for disk operations in connection with the maps. The icon marked 4 is used for printing out the currently active map. 5 gives information about the program version in use.

By clicking 1, alternatively by selecting the menu item “Filer - Nyt” (Eng.: new), a new map will appear. Since it makes no sense to start with an empty map, (it goes beyond the specifications of this product to be able to draw a new map), a copy of the map currently in use will appear. If no map is being used, the map last viewed will appear. The user must note, however, that any changes made to the new map, such as marking routes and addresses, will take place on the map first opened. This old map must be closed, before manipulating the new one becomes possible.

By clicking 2, alternatively by selecting the menu item “Filer - Åbn”, a map can be retrieved from the harddrive (or other media). The user must obviously have a map (of type MapInfo) ready, if he wishes the program to be able to use it).

By clicking 3, alternatively by selecting the menu item “Filer -

Gem", the currently active map can be stored on disk, including any modifications made to it, such as marked routes and addresses, Which layers are selected or deselected etc. *NOTE: It is important, that the user remembers not to overwrite the map Skovbo_utm32, that the programs starts with, since InfoRoute will start with a manipulated map afterwards. To avoid this, the user should always click on the menu item "Filer - Gem som..." (Eng.: Save as...) and save a copy of the map elsewhere and under a different name.*

By clicking 4, alternatively by selecting the menu item "Filer - Print", the map will be printed out on a printer, that the user chooses from the menu appearing before printout. If no printer is selected, the windows default printer will be used. It is a good idea to use a colour printer, since the marked routes and addresse will be invisible if a black and white printer is used.

By clicking 5, alternatively by selecting the menu item "Hjælp - Om InfoRoute..." (Eng.: Help - about InfoRoute...) a window containing information about which version of the program is in use.

9.2.3 The VerticalToolbar

The vertical toolbar with the icons 6-12 is used for manipulating the active map. This includes selecting parts of the map and selecting and deselecting layers.

By clicking 6 and 7, it becomes possible to zoom in, resp. out, on the map. The way this works is, that the mouse pointer is turned into a looking glass with a tiny "+" or "-" in it. By left-clicking on a point on the map with this new pointer, the map will zoom either in or out, centered around the position of the pointer. Zoom always takes place using a 1:2 scale. By clicking 8, the mouse pointer will turn into a hand. This can be used to move the map around. By placing the "hand" over a point on the map, the user can "grab hold" of this point by left-clicking, thus enabling him to move the map around by keeping the left mouse button down and moving the mouse. If the mouse is moved downward, the hand and the map will move to the south, enabling the user to view more northernly parts of the map.

By clicking 9, 10 or 11, it becomes possible to select a part or parts of the map. Clicking 9 turns the mouse pointer into an arrow, slightly different from the usual windows arrow. By clicking on different parts of the map with this new arrow, the parts will be highlighted. If a piece of road is clicked on, this road will be highlighted. If an area not occupied by a road is cliked on, the entire municipal district will be selected (requires the municipal district layer to be active). 10 and 11 basically does the same, but allow more parts of the map to be selected at once. 10 allows the user to draw a rectangle around the parts of the map to be selected and 11 allows the user to draw a circle for the same purpose. The effect of this will prove useful later on.

Clicking 12 brings up the menu for selecting and deselecting map layers and manipulate their characteristics. This menu will be explained in detail later.

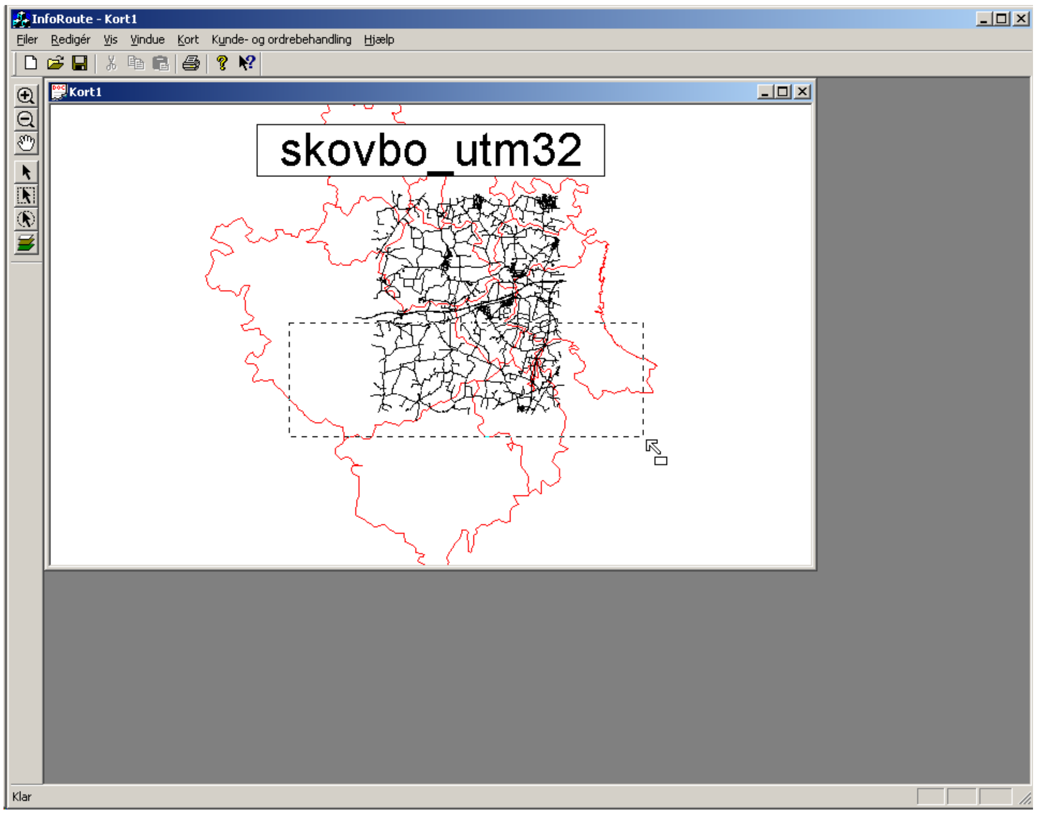

InfoRoute - Kort1
Filer Redigér Vis Vindue Kort Kunde- og ordrebehandling Hjælp
Kort1
skovbo_utm32
Klar

Figure 9.3: The user in the process of selecting the lower half of the map using a selection tool, in this case 10

9.2.4 The Menu Items

Besides the tool bars, it is also posssible for the user to use menu items for employing various functions. As has been shown earlier, the user can in some cases gain access to the same funtion by means of either a toolbar icon or a menu item. However, the menu items also contains functions not accessible through the toolbar. The basics of these other functions will be described here.

The menu "Filer" also contains, other than the items described earlier, the items "Luk" (Eng.: Close), "Se udskrift" (Eng.:Print Preview), "Printerinstillinger" (Eng.: Printer settings), "Forlad programmet" (Eng.: Quit Program) and "Initialisér" (Eng.: Initialize). "Luk" closes the active map, corresponding to clicking the "x" in the top right corner of the map. "Se Udskrift" shows how the map will look when printed out. "Printerindstillinger" activates the windows dialog box, making it possible to change various settings concerning the printout. "Forlad programmet" quits the program.

"Initialisér data" is special. By selecting this item, a message will appear at the bottom of the application window (the status line), saying "Etablerer vejnet i hukommelsen" (Eng.: "Creating road network in memory"). Next to the text is a staus bar, showing how far along this process is. Note that the entire road network is highlighted during the process. This merely means, that the process is underway.

Initializing the road data is necessary in order to use the programs build-in Shortest Path algorithm. It is a little time consuming (Approx. 30 sec. on a fairly modern PC running 700-800 MHz), but only have to be performed once (unless the road network is changed). So, instead of doing this every time the algorithm is used, the initializing process has been separated from the rest of the shortest path module to save time.

The menu "Redigér" (Eng.: Edit) is at present inactive, The same applies to the icons on the horizontal toolbar corresponding to this menu (The scissors, clipboard and clips).

In the menu "Vis" (Eng.: Show) the user can choose whether or not

to have the horizontal toolbar and the status line displayed on the screen. The staus line is the line at the bottom of the application window. The position of both horizontal toolbar and status line is displayed on figure 9.4.

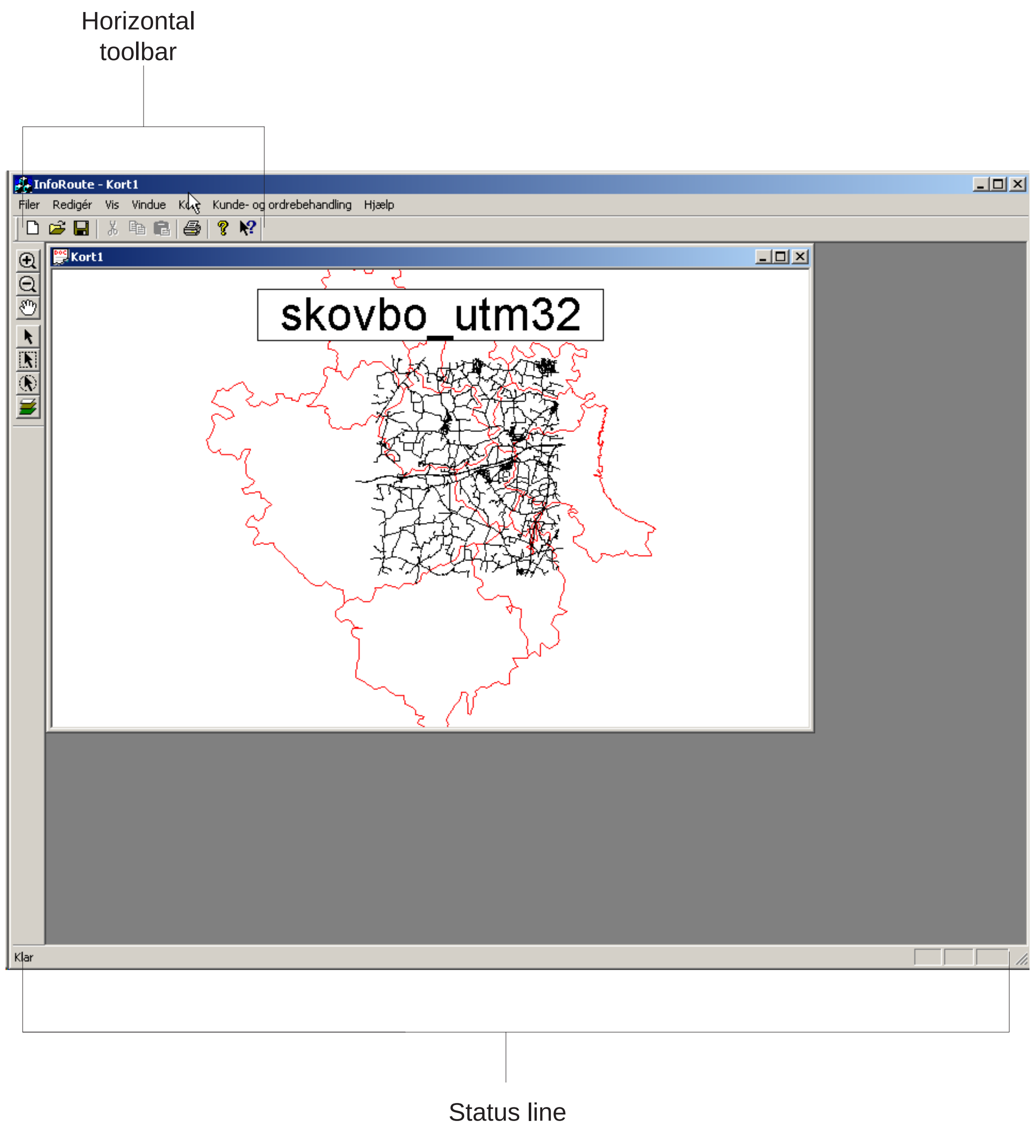


Figure 9.4: The location of the horizontal toolbar and the status line

In the menu “Vindue” (Eng.: Window) the usual options from windows regarding the placement and position of any or all active

subwindows in the application can be selected. Which window is to be the active one is also picked here.

The menus "Kort" and "Kunde- og ordrebehandling" (Eng.: Map and Customer- and requestdealings, resp.) contains tools for the location of addresses and shortest path routes as well as tools for creating and maintaining a customer and request database. The will be described later on.

The menu "hjælp" contains, besides the already mentioned "Om InfoRoute", the item "hjælp". In the present version of InfoRoute, it has not been deemed necessaery to provide the user with on-line help. Instead, the user should refer to this chapter.

9.2.5 "Find adresse"

"Find addresse" allows the user to localise an address and highlight it on the master map. By selecting this item, a dialog box appears as shown in figure 9.5.

As is apparent, The box consists of a minimap and a series of fields and checkboxes. The numbered items on the figure are utilized as follows:

1. **In this combobox, the road data to be used in the search is selected. It must be noted, that the user should make sure, that the selected data is actually road data, since road data and other geographical data, ZIP codes for example, appears in the same format. For the present task, it is recommended to use the preselected data (kdv_utm32)**

2. **Check this checkbox if the search is to be reffined by using ZIP codes.**

3. **If 2 is checked, this combobox will activate. Once again, the user should make sure to select correct data. Currently, the 2 and 3 items are not necessary, as no similar addresses with different ZIP codes exists within Skovbo municipal district. It will, however, become necessary when larger maps are used.**

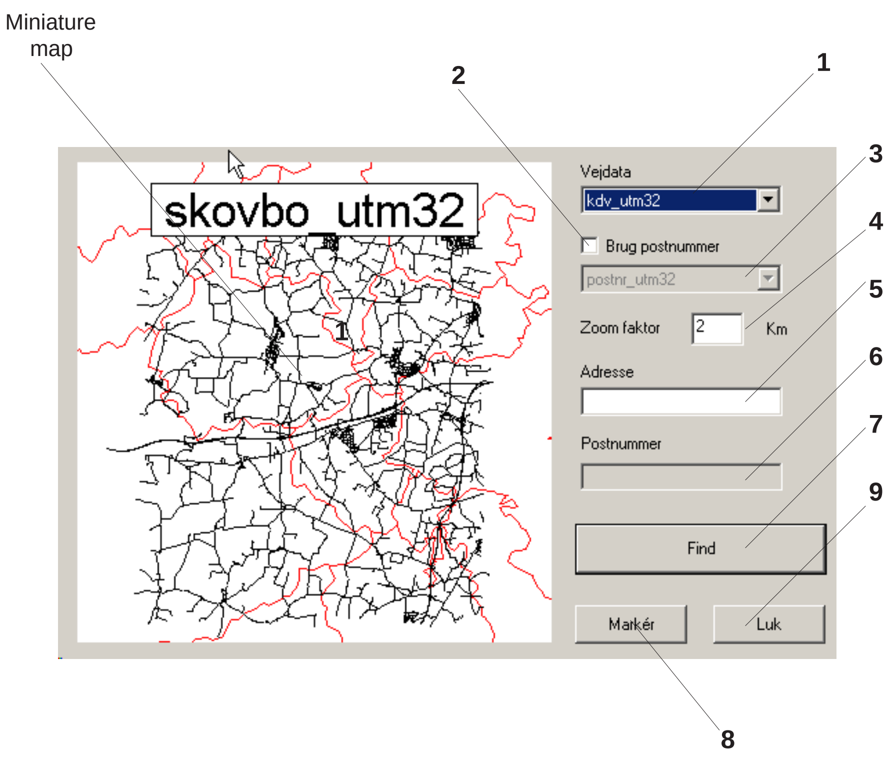


Figure 9.5: The Dialog box Find addresse

4. In this field an integer is entered. This number will designate the length of the sides of the square minimap, that will appear if an address is found. The center of this zoomed-in map, marked by a "+" will be the wanted address.

5. In this field, the wanted address is entered. Since the underlying map software is american, the house number is entered first, rather than using the danish method with the house number last. If no address is found, an error message will appear.

6. If 2 was checked, a ZIP code is entered here.

7. After having filled out the above, click "Find" (or press return).

8. If an address was found, it can be marked on the master map by clicking here.

9. 9 closes the find address dialog box

As can be seen, this functionality makes finding an address easy, and the build-in zoom makes spotting it just as easy.

9.2.6 "Find Korteste Vej"

"Find korteste vej" (Eng.: Find Shortest Path) makes InfoRoute's Shortest Path Algorithm avaliable to the user. By selecting the menu item "Kort - Find korteste vej", a new dialog box appears as shown in figure 9.6.

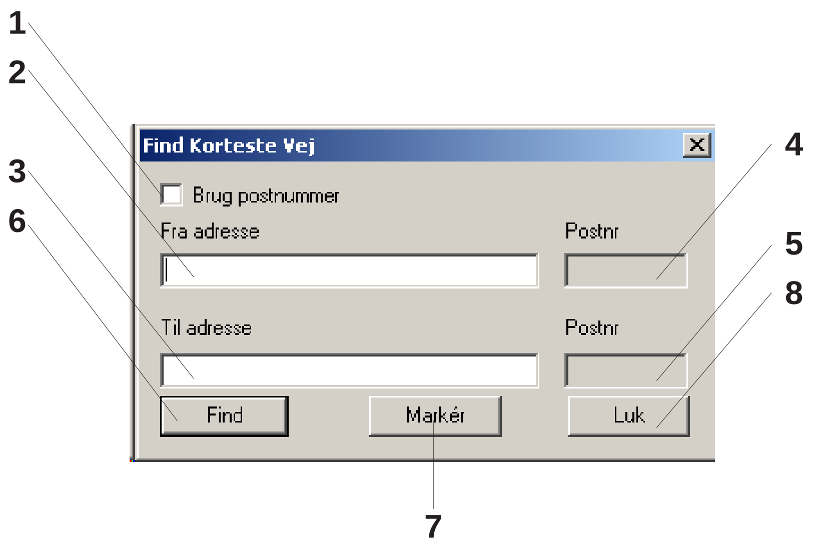


Figure 9.6: The dialog box Find korteste vej

The dialog shown in figure 9.6 is intuitively easy to understand. Here is a short explanation using the numbered items:

1. By checking here, the address can be further ascertained by using phone numbers.

2. The starting address of the calculation is written here.

3. The destination address is written here. Both 2 and 3 recquires the house number to be entered first.

4. This item is used, if 1 was checked, to designate the ZIP code of the starting address.

5. This item is used, if 1 was checked, to designate the ZIP code of the destination address.

6. Upon entering the above items, click Find. If both adresses exist, a message telling the distance between them will appear. If one or both addresses are not found, an error message will appear instead.

7. If the route found should be marked on the map, click Markér. The road pieces constituting the route will appear in red.

8. Closes the window.

The shortest path function thus makes it easy to find and highlight a route between to addresses. By utilizing this function the user does not have to worry about whether or not the routes he is using are the shortest possible.

9.2.7 Manipulating Map Layers

In both the find address and find shortest path functions, the user can alter how the map looks. Each of the functions use different layers on the map. How to manipulate these layers will therefore be explained now.

By clicking the layer control icon on the vertical toolbar, a dialog shown in figure 9.7 appears.
As can be seen in figure 9.7, this dialog is in english, since it is a feature of the underlying map software. The numbered items are used as follows:

1. This window allows selection of the layers, that the user wants to manipulate. This can be done using the mouse or 2.

2. As an alternative to 1, these buttons kan be used for selcting layers.

3. This item is used to remove a layer or add a new one. It is not recommended to use this option at present, unless the user has experience with map software.

4. This box is checked, if the selected layer is to be visible on the map. It carries no risks to make the layer invisible.

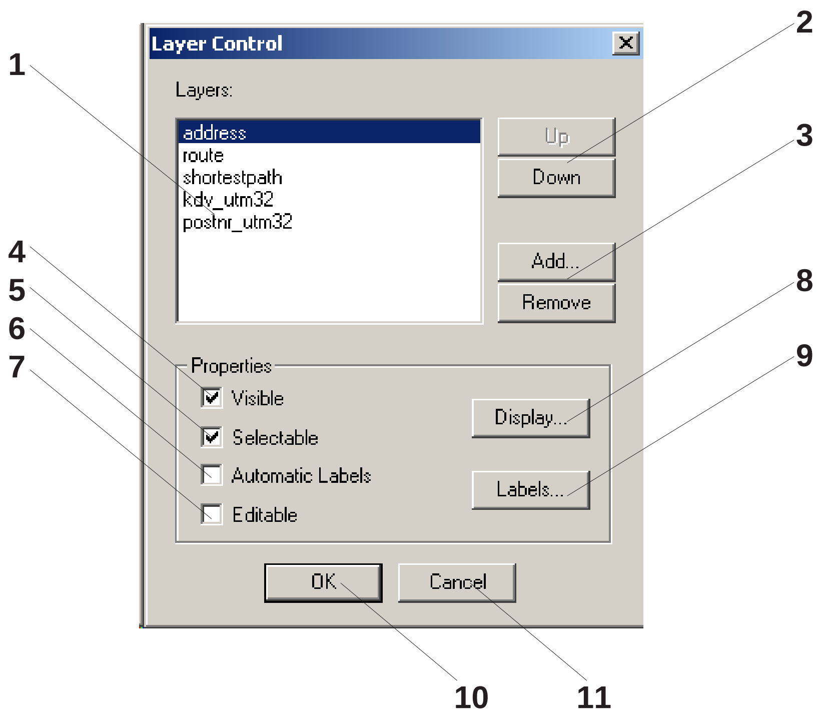


Figure 9.7: The dialog box layer control

5. **This box is checked, if the user wants to be able to select parts of the map.**

6. **By checking this box, InfoRoute will write location names on the map. This means, that the road network will have road names attached to it, while the ZIP code layer contains names (not numbers) of the postal districts. This checkbox only affects these two layers.**

7. **Checking this box will make the chosen layer susceptible to change, meaning that it becomes possible to remove parts of (or all of) the layer, using the tools described in the section concerning the vertical toolbar. It is not advisable to choose this option for the road network and ZIP code layers. However, it is quite useful on the layers with addresses and routes on them, to prevent the map from getting "crowded".**

8. **Clicking Display... provides access to a dialog, concerning the more advanced options for displaying map layers. This**

is for use by advanced users and is not recommended for the ordinary user.

9. After selecting the desired options, OK is clicked. The dialog then closes and the options are activated. If the user is not satisfied with his choices, clicking cancel will cause the window to close without the option selected options being activated.

The layers presently selectable on the map are:

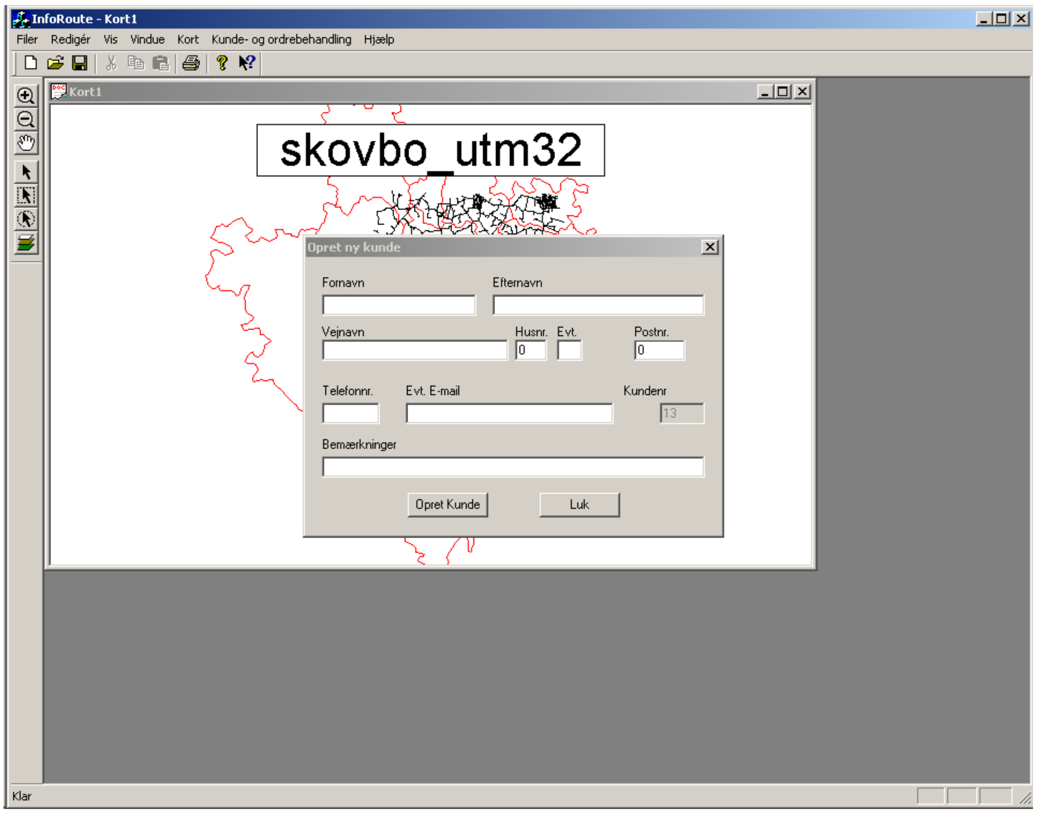
- address: This layer is used for marking addresses found in the find address dialog.
- route: This layer is empty at present. It will be used for route planning in later version of the program.
- shortest path: This layer is used for drawing the shortest paths found in the shortest path dialog.
- kdv_utm32: This layer is used by the program to draw the road netwok.
- postnr_utm32: This layer is used to draw the ZIP code boundaries. It is adviced to render this layer invisible when drawing shortest paths, as these might be mistaken for boundaries.

9.2.8 Create New Customer

By selecting the menu item "Kunde- og ordrebehandling - Opret ny kunde" (Eng.: Create new customer), a dialog appears, used for placing new customers in the customer database (see figure 9.8).

The dialog is easy to understand for which reason the items have not been numbered. The user enters First name, last name, roadname and number, letter (if any), ZIP code, telephone number, e-mail address (if any) and any notes, that might be useful. After making sure, that the information entered is correct, "Opret kunde" is clicked to save the information to the harddrive.

The effect of this is, that the customer information entered will be inserted into a new row in the customer database. Notice, that it is not possible to gain acces to the entire customer database from inside InfoRoute since even a moderate size customer dtatabase



Husnr. 0
Postnr. 0
Kundenr 13

Figure 9.8: InfoRoute with the dialog box Opret ny kunde open

would cause InfoRoute to attempt to write hundreds of names and addresses on the screen. Instead, a search function has been implemented (explained later), and the user can explore the database with Microsoft Access if needed.

9.2.9 Editing and Erasing a Customer

The menu item “Ret/Slet eksisterende kunde” (Eng.: edit/erase existing customer) is used for just that. By clicking this item, a dialog almost identical to the create customer dialog appears. The only difference is the buttons at the bottom. To find a customer to edit or erase, a phone number must be entered in the correct field, and the Find kunde (Eng.: Find Customer) -button is clicked. Upon this, the other fields of the window will fill up with the data on the customer. If no customer is found, an error message will appear.

If a customer is found, the user is free to correct any information he wants. Until the button “Ret kunde” (Eng.: Edit customer) is clicked, nothing will be saved. If the user changes his mind, the windows is simply closed without clicking this button. If the wishes is to permanently remove a customer from the database, click “Slet kunde” (Eng.: Erase customer) and the active customer will be removed from the database.

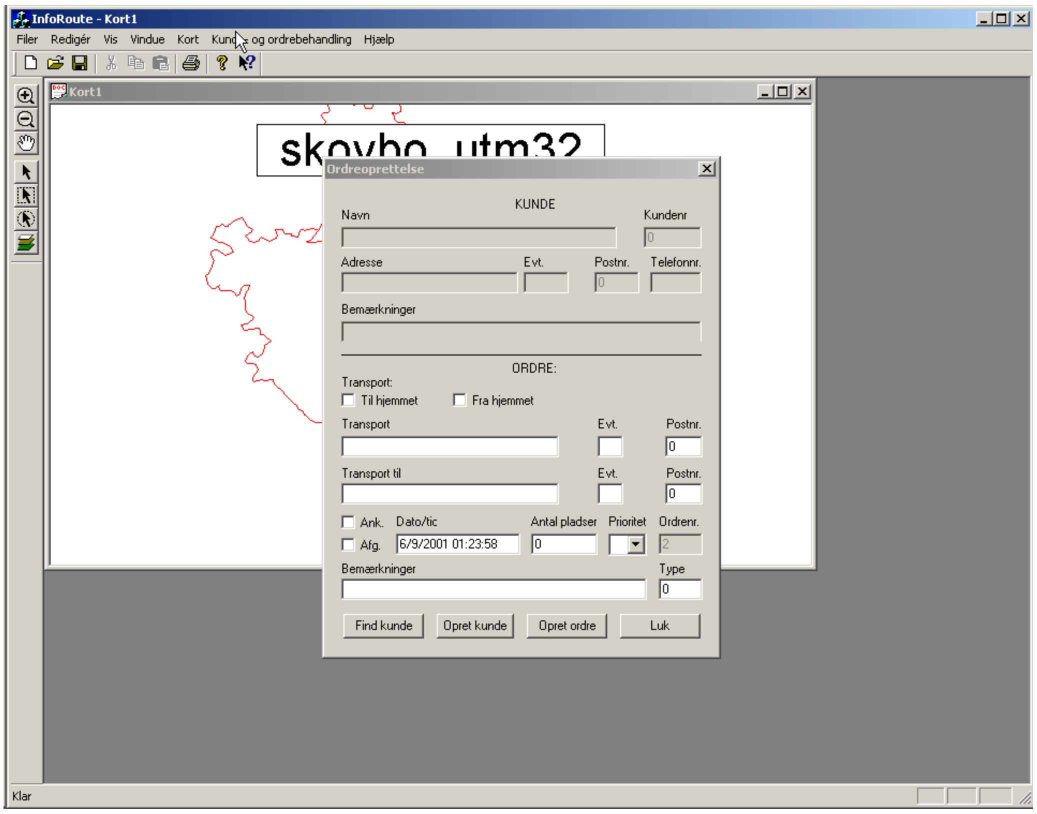
If a phone number with multiple customer attached to it is entered in the search for the customer to be edited, a message appears to relay the fact. After this, the first customer will fill the fiels of the window. Pressing “Find Kunde” again brings up the next etc.

9.2.10 Creating a Request

By selecting the menu item “Kunde- og ordrebehandling - Ny ordre” (Eng.: New request), the dialog (figure 9.9) for creating a request is activated. To use this function, a customer must first exist. It is not possible to make a request without attaching it to a customer in the customer database.

Therefore, the user must first click “Find kunde” at the bottom of the dialog. This calls up a search dialog identical to the one used for editing the customers. Upon having located a customer, the top half of the window will display the customer data. To ease the process of creating a new customer for a first-time order, there is a shortcut button to the create customer dialog, also at the bottom of the screen.

After having searched for or created a customer, the data concerning the requested transportation is filled in. this data consists of desired starting and destination adresses, the roadname and number in the same field, and a ZIP code. Furthermore, a date/time must be entered, and a checkbox designating the date/time as either the start or destination left checked or unchecked. finally, the number of seats and the priority of this request must be entered. Optionally, the bottom field can be used to note any special circumstances. This field is the only one not being directly tranferred from the customer data. Instead, it is left blank and it is up to the user to engage in a dialog with the customer to ensure, whether



Kundenr: 0
Postnr.: 0
Transport Postnr.: 0
Transport til Postnr.: 0
Dato/tic: 6/9/2001 01:23:58
Antal pladser: 0
Ordrenr.: 2
Type: 0

Figure 9.9: InfoRoute with the dialog box Opret ordre open

the customer information should be transferred, a new note entered or the field left blank, if no special circumstances apply to this particular request.

9.2.11 Editing and removing requests

The final menu item in "Kunde- og ordrebehandling" is "Ret/slet eksisterende ordre" (Eng.: Edit/remove existing request). As the name hints at, the item is used to maintain the request database. By selecting the menu item, a dialog box appears (figure 9.10).
The dialog shown in figure 9.10 is used for finding requests in the request database. As mentioned earlier, requests are connected to customers. Again, the dialog starts with the user finding a customer. The figure shows, that all fields except for the phone number field in the dialog are diabled. After having filled in the telephone number field, "Find kunde" is clicked. As before, if more

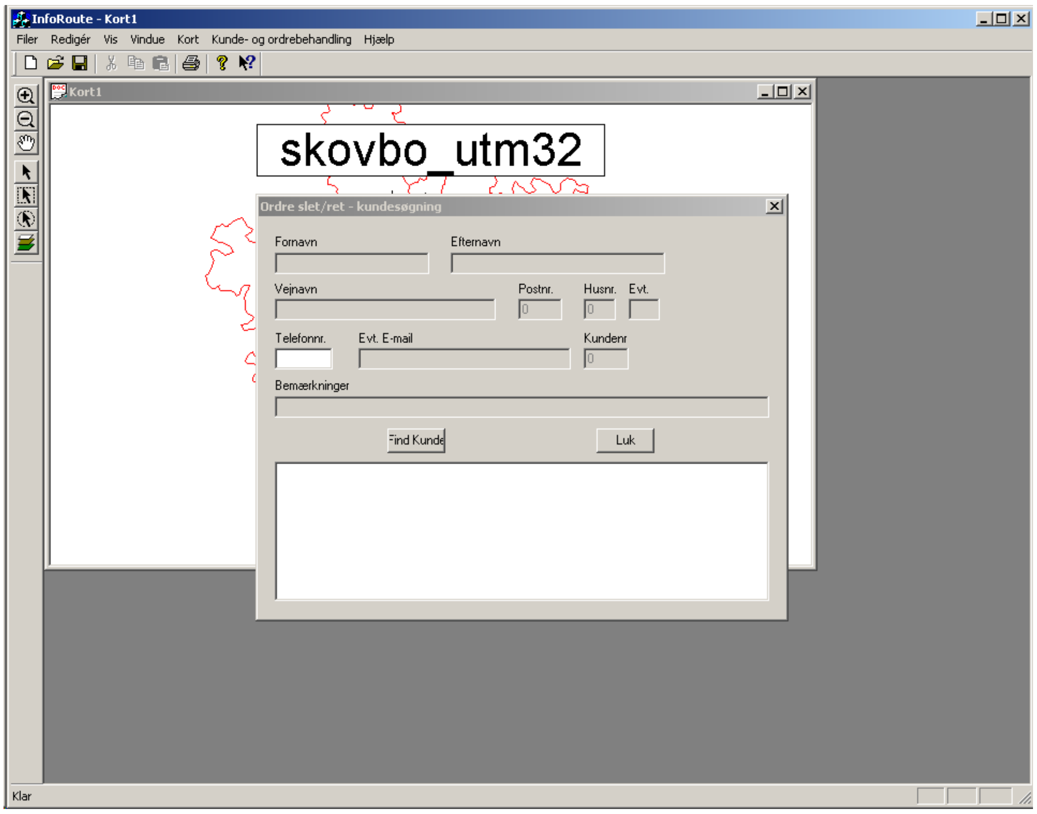
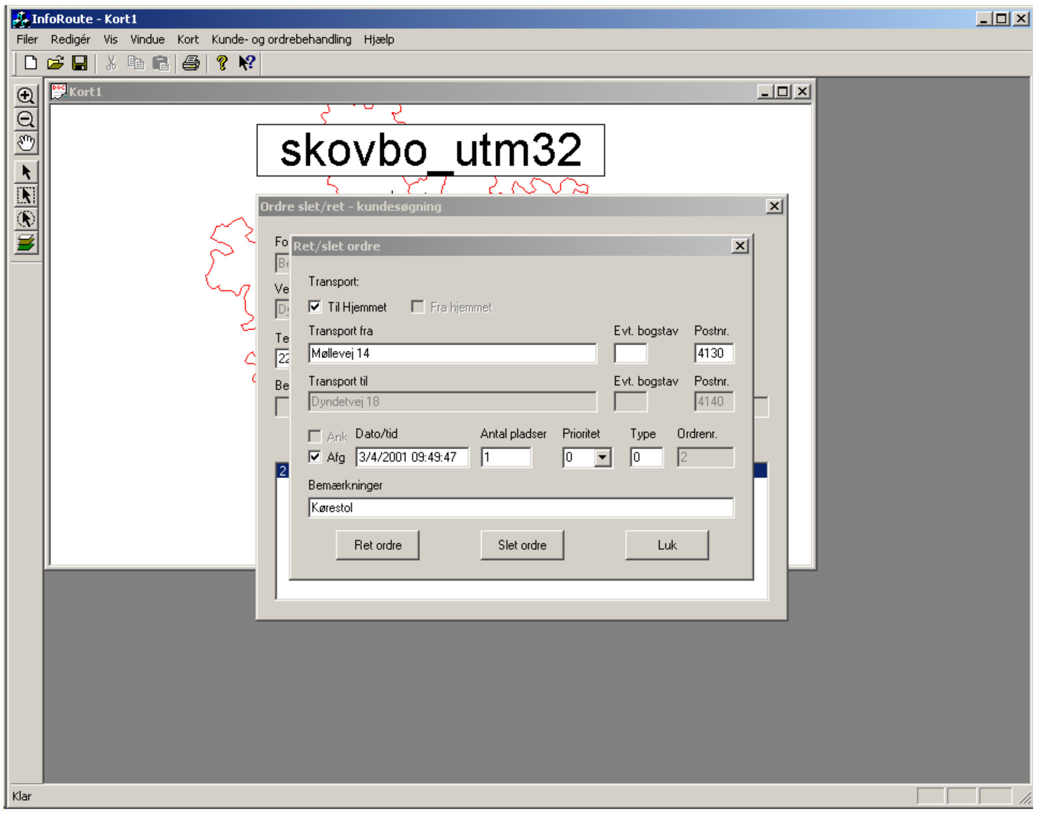


Figure 9.10: InfoRoute with the dialog box Ordre slet/ret - Kundesøgning open

than one match is found, a message will display this and the different matches can be viewed by clicking "find kunde" again.

When the desired data is displayed, the requests already made by this customer will appear in the field at the bottom of dialog box. The request, that the user wishes to alter is clicked. This in turn will open a window, similar to the bottom half of the create request window (figure 9.11).

After opening the dialog displayed in figure 9.11, the data can be edited at will. By clicking "Ret ordre", the changes will be saved to the harddrive. If the entire request has been cancelled, clicking "Slet ordre" will remove it from the database.


InfoRoute - Kort1
Filer Redigér Vis Vindue Kort Kunde- og ordrebehandling Hjælp
Kort1
skovbo_utm32
Ordre slet/ret - kundesøgning
Ret/slet ordre
Ret ordre
Slet ordre
Luk
Klar

Transport:
[✓ Til Hjemmet] Fra hjemmet

Transport fra	Evt. bogstav	Postnr.
Møllevej 14		4130
Transport til	Evt. bogstav	Postnr.
Dyndetvej 18		4140

Ank [✓ Afg]

Dato/tid	Antal pladser	Prioritet	Type	Ordrenr.
3/4/2001 09:49:47	1	0	0	2

Bemærkninger
Kørestol

Figure 9.11: InfoRoute with the dialog box Ordre slet/ret open after a customer with an active request has been located.

9.2.12 Summing up

As described in this manual, InfoRoute enables the user to create and maintain a customer database, create and maintain a request database, view a map, manipulate it's layers, find and mark an address, and find and mark a shortest path between two addresses. Furthermore, the program has been readied for further development, with the goal of making it a complete tool for route planning in every category of the field of transportation planning.

9.3 Other Results

In the above, the functionality of InfoRoute has been described in its present state. But since the entire design process has been focused on the general aspect of the task at hand, a lot of the results generated are not directly visible in the program. These results, though less apparent than the application, are never the less an important part of the project - maybe it's most important part. These results will be summed up here.

First of all, the database has been designed with focus on future use in various transportation systems. The design allows for advanced algorithms to be easily inserted. The program has been prepared for complete route planning for a fleet of heterogeneous vehicles with capacity substitution. describe the depot. Map layers for advanced uses are reserved. Several advanced map features are actually implemented.

The results, therefore, surpass the product readily avaliable. Developers can, by using the database and application, visualize their results, thus simplifying their work with algorithms and applications that use them.

9.4 Representation of road data - MapX

The following will provide a brief description of the advantages and disadvantages of utilizing MapX for displaying data graphically. MapX is developed and sold by the american MapInfo Corporation, one of the leading suppliers of GIS-software. The MapInfo Corporation supplies a long list of products, all of which stems from the program MapInfo Professional - a program for processing geographical information much the same way as does ArcInfo etc. MapInfo supplies programs developed for use on the internet, with Visual Basic etc., making MapX suited for inclusion in either a Visual Basic or a Visual C++ project.

MapX consists of several components, where GeoSetManager is the most important non-C++ specific component. Using this program, maps (geosets) of layers, each consisting of tables in the MapInfo format, are build. The layers of such a map are then registered in GeoDictionary, from where it can be used by outside code, C++ for example. Described in specific terms, GeoDictionary is a file that the Windows Registry refers to in such a manner, that a GeoSet-file is coupled to its layers when it is opened.

The above mentioned characteristics may provide some problems conserning the distribution of programs developed with MapX, because GeoSetManager (and thereby MapX) will have to be installed on all workstations that are to use the software developed for use with MapX. The C++-specific part of MapX must be installed in order to use it with programs developed for this purpose.

With regard to the map consisting of several layers, this provides the possibility of a user adding self generated layers to the program. An excellent example is the map in this application, which consists of a layers representing road-pieces and a layer with the ZIP code boundaries represented. The layer with the road-pieces is made up by lines, where the layer with the ZIP codes consists of polygons. A specific layer on the map can only contain one type of objects (points, lines or polygons). Thus, InfoRoute establishes layers for recording the location of addresses, Shortest Paths and routes, and these layers can then be shown, edited and erased individually.

In short, MapX consists of a series of C++ classes, all connected to a comminucationsclass (CMapX). These classes contains a significant amount of functionality, too much to be treated in depth here. This section is therefore limited to provide only a short description of the functionality used by InfoRoute.

By including MapX in the code, and then creating an instance of it in MFCs view-class, the developer gains access to all functionality in MapX. Following this, MapX is set to use GeoSets and by altering various parameters, the different layers of the map can be displayed on the screen.

Some of the functions from MapX that are used here is address search, search for road pieces when these are in close proximity to an address, highlighting objects on the different layers, zoom, selecting objects etc. Searching for the road pieces in close proximity to an address is necessary, since addresses are not linked directly to a road piece. Instead, the addresses are points, located a short distance from the road. The fact, that addresses are points also means, that they cannot be included in the layer containing the road pieces.

The functionality mentioned above can be called directly from their respective classes, but must of course be called in a way consistent with what functionality the developer wishes use and combine. Therefore, there is still a lot of development work to be performed, even when using MapX, but MapX surely simplifies the process Furthermore, most digitalized roadmaps today can be converted to the MapInfo format, allowing them to be used directly in applications developed with the help of MapX.

9.5 Implementation

As the project primarely consists of programming, this chapter will provide a thorough description of the development work in connection with the completion of the application. This work will obviously be founded on the structure of the database, and the implementation has been made parallel to the functionality analysis. In other words, during the programming phase, it was found necessary to adjust the functionality analysis. This occured when the programming revealed, that the conclusions made during the original functionality analysis where impractical. However, these adjustments hasn't changed the program in any basic way, but has merely served to make the endproduct more user friendly.

9.5.1 Structure

Figure 9.12 shows the structure of end product. As can be seen, the possibility inherent in the programming language of encapsulating diferent functionalities in individual modules has been exploited. The boxes on the figure, enveloping the various functionalities shows how the internal communication has been limited. These limitations are among the most important features of the program and the following description will reveal how the communication takes place between the modules and the classes, respectively. The headlines used in the rest of this chapter corresponds with the names of the modules.

9.5.2 Database

This module takes care of the communication with the database. The individual classes of the module are named CDb (Class Database),

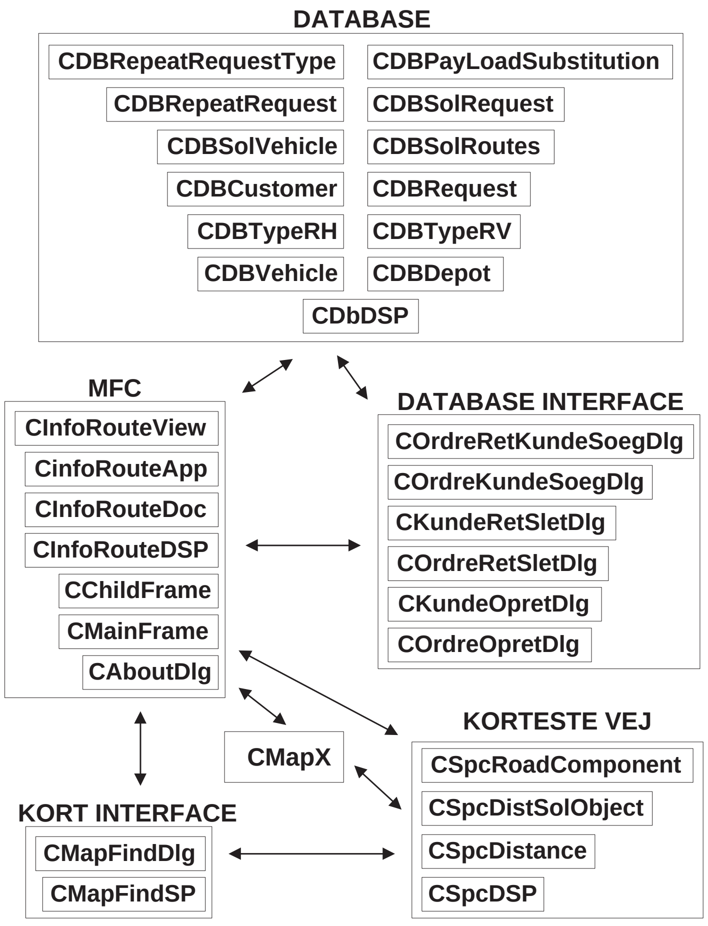


Figure 9.12: An overview of the structure of the program and the internal communication between modules.

followed by the name of the table it communicates with in the database. The modules internal communication is shown on figure 9.13. This figure also shows, that CDbDSP is the only class communicating with the rest of the program.

Figur 9.13 shows, how the individual classes communicate with their corresponding table. Each instance of a classe establishes a connection with its underlying table. Initially, this connection points to the first element of the table. Subsequently its possible to point to other elements using various build-in functions, ie. it is possible to find an element, that satisfies certain criteria or to move

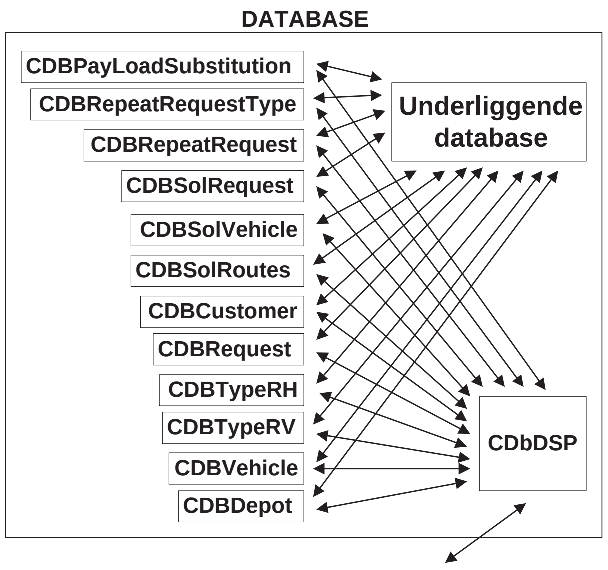


Figure 9.13: An overview of the internal communication of the database module.

to a specified element. When the connection has been established, data from the database can be read directly from the table.

When establishing and editing elements, the classes provide the possibility of calling certain functions, such as Edit(), Update(), AddNew() etc., still bearing in mind the importance of the connection pointing to the correct element. To control this functionality, it is kept together in a communications class named DSP (Data Storing and Processing), where the connections to the tables can be created and controlled.

The DSP-class provides other parts of the program with access to the database using its own implementation of the functionality needed to do so. Thus it is ascertained, that the database is updated or edited with correctly formatted data. It also eases the process of replicating functionality between other classes. Notice, that the database is never opened directly in the communication. Instead, data is written into the database by calling the Update() function on the database, thus protecting future users against loss of data.

In the database interface module the possibility for the user to interact directly with the customer- and request parts of the underlying database is established. Several Dialogs have been made available, that read and write data to and from the files of the database.

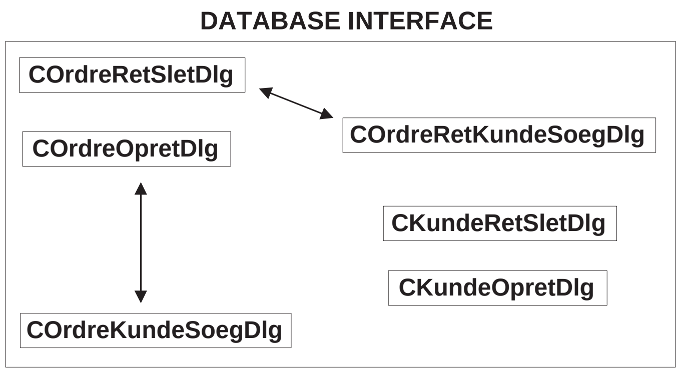


Figure 9.14: An overview of the internal communication in the database interface module.

Figure 9.14 shows 6 classes, each representing a dialogbox in the program. COrdreRetKundeSoegDlg and COrdreKundeSoegDlg are such dialogs, helping the user find customers in order to edit them or enter them into the system as new customers, respectively. By utilizing this approach, using separate dialogboxes for processing customers and requests, the GUI acquires a uniform design, thats makes the handling of it by an end user much easier.

Each class in figure 9.14 communicates independently with the communications class of the database module. This design is chosen for two reasons: First, this minimizes the respons time of the program. Secondly, it allows the various dialog boxes to be used elsewhere in the program without having to modify them. The downside of this form of communication between a module's classes is, that it makes implementation of the module in other classes or programs much harder. This can be changed later, however, by adding a communication class to the module and using this to handle the modules external communication.

9.5.3 MFC

The following describes the basic structure of the use of MFC in the implementation. Apart from MFC themselves, a class named CInforRouteDSP is included. This class handles the tasks, that MFC do not traditionally take care of, for example making sure, that other classes gain access to the functionality of the MFCs.

MFC has been included in the development work to easy the process. By using these classes, a lot of functionality known from the Windows environment has been implemented beforehand, including menu-lines, print functions, I/O functions etc. This functionality can be used and modified relatively easy. In addition to this, some of the functionality used in the graphic representation of roadmaps, routes etc. is already implemented in MapX, which itself use MFC for exactly that purpose.

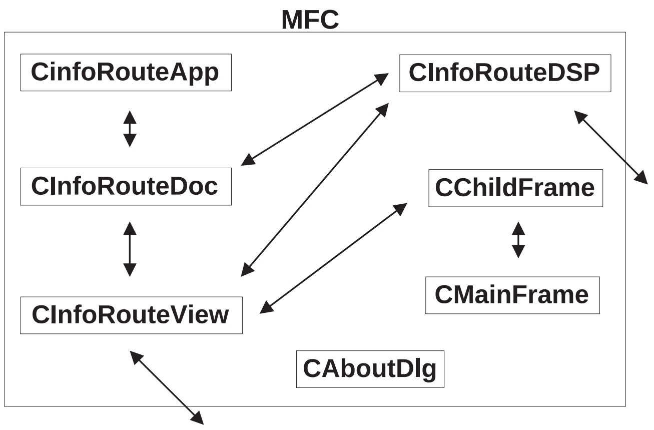


Figure 9.15: An overview of the generel structure of MFCs used

In short, the MFC-structure basically allows an application to have a number of documents of different kinds. A document is an instance of a document template. The template specifies the various characteristics of its type of document. A document template could for example contain functionality in connection with displaying textfiles in the screen. Each instance (each document) would then be a text file or a text document.

Just like each document template can have different documents attached to it, each document can in turn have views attached to it. This means, that a textfile can be shown on the screen like we usually see it, it can be shown inverted, without consonants etc. These different representations of the document are its views. In this case, a document template, a document and a view containing road maps presented graphically have been made. This functionality has been implemented in Multiple Document Interface (MDI) to make it possible to extend the functionality using more document templates at a later stage.

MFC also contains frames corresponding to the application and its windows. The application itself contains a main window contained in the class CMainFrame. This in turn contains some static functionality, that is not affected by whichever functions or views the application is actually working with. That the information is static is ensured by passing on the functionality to all the frames of the application. In this case there is one instance of CChildFrame, the subwindow containing the road map. In the various subframes or childframes, it is possible to implement menu items, status bars etc. that are specific to the subwindow. This structure also have the added bonus, that menu items etc. changes dynamically depending on which subwindow the user activates.

CAboutDlg is made up of a dialog box, that displays the name and version number of the program. Apart from this, there is no other functionality. The class is included as a standard feature within MFC and needs no further implementation. Thus, it was decided to include it.

9.5.4 Map interface

The interface for the map displayed on the screen primarely consists of the two classes shown in figure 9.12. Each class represents a dialog box for displaying addresses and shortest path between addresses, respectively. This funtionality is included, as it seems to be the most natural extension of the graphical view. In section 9.2 a more thorough description of the functionality of the module is provided.

A major part of the functionality used in handling the map has been implemented by use of a toolbar, which is a vital part of the view (MFC) containing the map. The implementation of the toolbar will, however, be described here, since it solely concerns handling the displayed road map.

The toolbar enables moving the map around on the screen, zooming etc., but the most important functionality is the implementation of layer control on the map (The make up of the map is described in section 9.4). This includes making various layers on the map visible or invisible, layers can be protected from tampering, explanatory labels can be displayed etc. The standard map contains five layers: Roads, ZIP codes, addresses, shortest paths and routes. However, more layers can be added as they are needed.

9.5.5 The Work Process

As was mentioned in the beginning of this chapter, it has been necessary to make changes to the implementation and the functionality simultaniously during the development work. This was primarely caused by the continous testing of the programs user friendliness giving cause for updates in dialog boxes and functionality. Some functionality, which at first seemed effective, would later prove to be impractical when tested. Among other things, it quickly proved necessary to be able to find customers directly from the request dialog window. The option of showing a list of requests from an active customer was also found to make the program easier to handle.

Looking at the road map and its functionality it became obvious, that a user during his daily work would need to be able to find and display customers on the map, as well as having the program do a single range calculation when necessary. Another discovery was how frustrating the calculating time when initialising the map at program start was. This waiting period can most likely be reduced significantly by trimming the implementation. Also, a status indicator was created, showing the remaining time of the initialising process. This greatly reduces the frustration by giving an indication of the amount of time left to wait.

As just described, the implementation has taken place alongside extensive testing of the various parts of the program. During these tests, besides making generel corrections to the code, the focus has been on improving the user friendliness of the application. This also means, that the final application acts as stable as has been at all possible to determine and on the test machine used, it proved impossible to locate any errors in the code affecting the stability of the program.

9.6 Perspectives

This section concerns the future use of the developed product in connection with testing algorithms to solve transportation problems. The chapter follows the structure of the report, adding comments to the database, the implementation and the map viewing.

9.6.1 Possibilities for Adaption to Transportation Branches

Probably not all problems concerning transportation can be readilly solved with the underlying database's structure. In particular, a practical problem can have some very specific related parameters, that it is necessary to take into account when dealing with the problem. The database is, as mentioned before, of a very general nature and its simplicity and easily grasped structure should be able to help future developers, should they wish to add to it or modify it.

The application is prepared for reading parameters by use of text files. This form has been selected to meet the needs from many different algorithms, that require all sorts of parameters in order for them to run properly. These parameters are the domain of the developer. It would not be practical to include them in the database.

9.6.2 Inserting new Algorithms in the Program

Besides the data connections in the algortihms, the implementation of InfoRoute has focused on partitioning the functionality to provide easy access to the information for the road network and other data, This means, that is is easily understood where in the program the information about the shortest path is retrieved, where the new menu item can be placed, where the map manipulation takes place etc. Therefore, the problem of implementing other algorithms in InfoRoute should not be problematic.

However, there are certain requirements to the implementation of future algorithms, if the usability of the interface is to stay unchanged.

- The implementation of the algorithm and its administrative code must be encapsulated and its communication handled by one class, which job it is to maintain a connection with InfoRoutes various communications classes.
- The functionality of the algorithm module must be clearly documented in its communications class, for example by means of comments for each of its functions.
- No communication must be directed against any InfoRoute class other than the above mentioned. This goes for MapX as well, which must not by used directly.
- If any new funtionality is desired within InfoRoute, it must be implemented as described in this report, and its functionality accessible through the communications classes.
- Extra menu items in the interface must be implemented in the algorithms own document with its own views. This helps to avoid a crowded and confused menu structure to appear as more and more algorithms are used.

By having developers comply with the rules outlined above, InfoRoute will maintain its present simple structure, since the present documentation will still be usable for implementing even more algorithms. This will hopefully allow the application to grow without compromising its existing modularity.

9.6.3 MapX

As mentioned in chapter 9.4, it is not solely an advantage to use MapX with InfoRoute. It can prove a problem that seperate licenses and installations of MapX are needed to run InfoRoute.

Also, a small alteration in CInfoRouteView is needed at present in order to be able to use other maps than the one currently used,

making it necessary for a full version of MapX to be installed in order to be able to compile new maps. If it proves necessary, changing maps can of course take place as part of the compiled version, and consequently InfoRoute is prepared for opening map files.

9.7 The Users Experience

A special warning when enhancing InfoRoutes functionality in the future: It is important to keep menu structure etc. at a minimum to avoid flooding the user with functionality that might be confusing. Functionality not used wery often should be buried deeply. Also, waiting time in any program is irritating and should be avoided. If unavoidable, use the progress bar already implemented and connect it to the algorithm. Its functionality is already available through the communications class.

Apart from the possible things mentioned in the above, no problems should occur when including other algorithms in InfoRoute, and the development time for connecting 3. party algorithms to it is minimal. This makes it possible to present the results from such algorithms in a graphics user interface.

Chapter 10

Multi capacity Cluster algorithm - McCluster

his chapter descibes how the cluster first insertion second algorithm developed at CRT in Montréal has been extended to solve the problem at Jensens Turisttrafik (see section 4.1 and the following section 10.1). Also in this chapter the reason for choosing the algorithm from CRT as the basis of McCluster (Multi capacity Cluster algorithm) is found.

10.1 Problem characteristic

As described in section 4.1 we have a fixed set of vehicles operating three different routes. The capacity of the vehicles is heterogeneous and divided into a capacity for sitting passengers and capacity for wheelchairs. The two types of capacity are fully compatible meaning that one wheelchair uses space equal to two seats. However not all seats can be converted to space for wheelchair, which results in two different subcapacities and one total capacity.

The routes are flexible with required stops at certain points in time to ensure correspondance with train schedules, school opnening and closing hours, and activation center opening and closing hours. It is preferred to pickup and deliver passengers at the fixed set of stops belonging to a route, but DtD transportation is at times required. At certain time intervals during the day different types of passengers have priority and must be serviced. In the evening the three routes are replaced by an exclusive DAR transportation system without fixed stops serviced by one bus.

At the beginning of each day, a large number of requests for transportation that day is known in advance. During the day, new requests are called in at the call center and the planner immediately informs the caller if the request can be met and the time of pick up. Requests have to be made no later than one hour before requested pick up, and the bus is allowed to arrive within a 15 minute time interval around the requested time of service. Passengers calling in can request respectively a desired time of pick up or a desired time of delivery. The bus then has to arrive no earlier than the desired time of pick up respectively no later than the desired time of delivery.

When a request is recieved and accepted by the planner it is dispatched to one of the drivers. Routes are planned by the driver from the list of requests to be included in the route. In other words the purpose of the automated planning is to accept or deny service to requests and to dispach accepted requests to the correct route. Accepted requests will also recieve a time stamp from the automated planning stating the required time of pickup or delivery within the system parameters. System parameters are length of time windows and maximum excess ride time for customers.

The objective is to maximize utilization of the fixed set of vehicles, which transfers to maximizing availability for new passengers. Minimizing the total distance driven is secondary, and may in some cases conflict with the necessary minimization of time driven. Driver scheduling, breaks and other driver related issues are not included in the problem. From the passenger viewpoint the objective is to drive as directly as possible between destinations and to spend as little time in the vehicle as possible.

10.2 Choosing the structure of the algorithm

Based on the previous section 10.1 we can list some requirements the algorithm used to solve the DAR problem must meet. The following items are listed according to their importance.

- Low computational time. As we are dealing with the dynamic case of the DAR Problem, the ability to obtain results instantaneously is very important.

- Problem adaption. As the problem changes over time from fixed routes to flexible routes, a versatile algorithm must be used to obtain the solution.

- Manual input. The planner needs the ability to manually fix assignments of requests to vehicles.

- Fast reoptimization. When realtime information such as delays enters the planning process, reassignment of requests to vehicles must happen in low computational time.

- Common sense. Since the algorithm is going to be used in software to assist a planner, the construction of plans must make sense to the planner.

The last item concerning the planners perception of the quality of the solution might actually be the most important item. However this is a “soft” item which is hard to quantify, and thus placed at the end.

When studying the algorithms described in chapter 5 it is clear that the requirement of low computational time effectively eliminates a large number of possibilities. Thus dynamic programming methods, column generation and metaheuristics all require too much computational time to obtain comparatively good solutions. This leaves only insertion and clustering first insertion second.

The difference between a pure insertion and a clustering first insertion second heuristic is small. The cluster first heuristic simply seeks to obtain better results by utilizing available knowledge of the problem at the beginning of the planning process. This knowledge is used to initialize the routes in a reasonable manner, where requests located near each other in space, time, or both are chosen to be inserted into the same route.

Since we operate with a large number of requests known in advance, it seems reasonable to use clustering. Clustering might also lead to better solutions when we have fixed routes because of the ability to cluster the requests with stops on the routes. As requests are called in during the planning horizon, a greedy insertion can then be applied to add the requests to the initial solution. Greedy insertion might not be the most intelligent approach, but the computational time required by simple insertion is very small.

The most flexible of the clustering heuristics is the one mentioned in Roy et al.[43][42][44]. Although applied only to the statical problem by the authors, it seems simple to extend the insertion procedure to the dynamic problem. As we will see in the following part of this chapter, extending this algorithm to consider all the complex additions to the original problem stated by the authors (Roy et al.), is comparatively uncomplicated.

10.3 Implementation

The structure of McCluster is illustrated in figure 10.1. The labels in the figure include the subsection in which each box is explained.

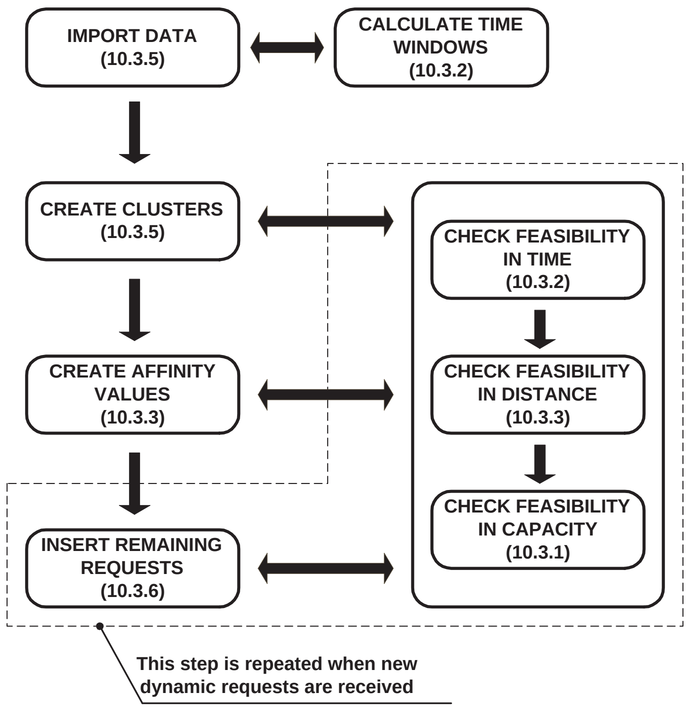

IMPORT DATA (10.3.5)
CALCULATE TIME WINDOWS (10.3.2)
CREATE CLUSTERS (10.3.5)
CREATE AFFINITY VALUES (10.3.3)
INSERT REMAINING REQUESTS (10.3.6)
CHECK FEASIBILITY IN TIME (10.3.2)
CHECK FEASIBILITY IN DISTANCE (10.3.3)
CHECK FEASIBILITY IN CAPACITY (10.3.1)
This step is repeated when new dynamic requests are received

Figure 10.1: The structure of McCluster.

10.3.1 Capacity

The special problem covered in this thesis involves capacity substitution as described in the chapter concerning the mathematical formulation in section 6.4. In the mathematical formulation, we introduced the possibility of having different capacities all relating to a total capacity. The example used was a bus with twenty seats, of which half could be rearranged for wheelchairs. Each wheelchair then require the space of two regular seats. To check if the constraint on capacity of a vehicle is violated by including a new request in the route of the vehicle, it is necessary to first check violation on each type of capacity. Thereafter each type of capacity is multiplied with a substitutional factor (this factor is two for wheelchairs since they require two seats), and the sum of these multiplications must then be less than the total number of seats.

In section 6.4 it is noted that practical DAR problems might be more complicated, since it is not always possible to just convert all capacities to a single unit (eg. seats). An example is a bus with sixteen seats of which six can be rearranged to accommodate up to three wheelchairs. In addition to the sixteen seats, the bus also has permanent space for two wheelchairs and this space can also be converted to hold one bed. The possible variations of the capacity of such a bus are many, but the difficult part is the lack of an unambiguous total capacity, since it is invalid to convert the space for one bed into seats.

While the problem of arbitrary substitutions between capacity types is very complicated to handle in a mathematical formulation, the algorithmic solution is fairly simple as seen in algorithm 10.1.

As mentioned in chapter 11 it was not possible to get data specifying the exact type of capacity required, which unfortunately made the use of multi-substitutional capacity unnecessary. The implementation of McCluster is however prepared for such a definition of capacity, just as the database used in InfoRoute (see section 9.1) is prepared for storage of substitutional factors and multi dimensional vehicle capacities.

Algorithm 10.1 Dealing with multi-substitutional capacity.

```
List of vehicle capacities (VCap) = ${c_0, c_1, ..., c_n}$

Matrix of substitution factors (SFactor) = $\begin{bmatrix} 1 & s_{1,2} & \dots & s_{1,n} \\ s_{2,1} & 1 & \dots & s_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n,1} & s_{n,2} & \dots & 1 \end{bmatrix}$

List of vehicle load (VLoad) = ${l_0, l_1, ..., l_n}$

i = 1;
j = 1;
while(i < n+1){
    load = 0;
    while(j < n+1){
        load = load + VLoad[i]*SFactor[i][j];
        j++;
    }
    if(load > VCap[i])
        return(capacity is exceded);
    i++;
}
return(feasible in capacity);
```

10.3.2 Time windows

As seen in figure 10.1 time windows are calculated when requests are imported. The method used to calculate these time windows is the same as shown in figure 6.1 where the customer specifies either a desired time of pick up or a desired time of delivery. At a fixed level of service defined by the maximum excess ride time and maximum time span of the time windows, the initial time windows can be derived.

This method of calculating the time windows does however prove to be somewhat difficult to handle, since the time interval at the destination (pick up or delivery) where nothing is specified by the customer will exceed the allowed time span. To overcome this, the time windows in this step are treated as temporary values. When a solution is found, the actual time of service is known. As this is done at the time of request, a time window based on the actual time of service is given to the customer.

If the passenger requests a desired time of pickup, the time windows ($\{a_i, b_i\}$ and $\{a_j, b_j\}$ denotes respectively time window of pick up and time window of delivery) given to the passenger at the time of request can then be derived from the solution as follows. Since the desired time of pick up is specified, the vehicle can not arrive earlier than the desired time, thus a_i is given by the passenger. Latest time of pick up b_i is then found by adding the allowed time span to a_i. Running the algorithm resulted in an actual time at each of the vehicles destinations, which corresponds to knowing the expected distance in time between the stops. We can then just add the expected distance to the time window at pick up to get the time window at delivery. However when the resulting time window at delivery makes it possible to exceed maximum excess ride time a downward (backward in time) adjustment of the time window at delivery will be made. The situation is of course reversed, when a desired time of delivery is given at the time of request.

Narrow time windows at destinations leaves little room for planning when new requests are recieved. However when adjusting the time windows according to the expected ride time, situations severely limiting the ability to accomodate new passengers can occur. When the routes are initialized with a small set of requests known in advance, the expected ride time of these requests will be close to that of a taxi service. This is especially the case, when the first request on a new day is recieved. In this case the expected ride time will equal the time of driving directly between the two stops.

To maintain room for dynamic requests in the planning process, a minimum excess ride time can be included in the initial phase. The optimal value for minimum excess ride time could be estimated empirically, and by using historical data to forecast expected requests. In this thesis a minimum excess ride time is not used, since a very large number of requests are known at the beginning of the planning process.

When requests are imported and time windows are set, we face the problem of checking feasibility of inserting request j after re-

quest i in a route. As this insertion is limited by precedence constraints (pick up of a passenger must occur before delivery of the passenger), there is a total of three possible sequences of stops:

1. $p_i - d_i - p_j - d_j$
2. $p_i - p_j - d_i - d_j$
3. $p_i - p_j - d_j - d_i$

where p_i is pickup of the i'th passenger and d_i is delivery of the i'th passenger (see figure 10.2. Of course there is also the possibility of inserting passenger i after passenger j in a route which adds three more sequences equal to the three above with i and j swapped.

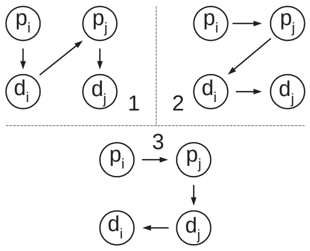


Figure 10.2: The three possible ways of linking two requests.

In order to set up constraints to check the feasibility of inserting passengers in the different sequences, we consider the general sequence of stops:

- $a - b - c - d$

Each of the stops a through d have time windows denoted $\{a_l, a_u\}$, $\{b_l, b_u\}$, $\{c_l, c_u\}$, and $\{d_l, d_u\}$ (l=lower bound and u=upper bound) the distance in time between the stops are $t_{a,b}$, $t_{b,c}$, and $t_{c,d}$. In order for this sequence to be feasible, each of the three sequences of two stops $a - b$, $b - c$, and $c - d$ must be feasible. To ensure this we can construct a constraint for each of the two-stop sequences:

$$min(b_u, min(c_u, d_u - t_{c,d}) - t_{b,c}) - a_l - t_{a,b} > 0 \tag{10.1}$$

$$min(c_u, d_u - t_{c,d}) - max(b_l, a_l + t_{a,b}) - t_{b,c} > 0 \tag{10.2}$$

$$d_u - max(c_l, max(b_l, a_l + t_{a,b}) + t_{b,c}) - t_{c,d} > 0 \tag{10.3}$$

For the first way of inserting a request into a route, constraints 10.1 and 10.3 are automatically feasible, thus it is only necessary to check violation of constraint 10.2, which again can be simplified to:

$$c_u - b_l - t_{b,c} > 0$$

For the second way of inserting a request into a route, all three constraints must be used in the listed form, but in the third case constraint 10.2 is automatically feasible. We then have to check only constraints 10.1 and 10.3, which we can simplify to:

$$min(b_u, d_u - t_{c,d} - t_{b,c}) - a_l - t_{a,b} > 0$$

$$d_u - max(c_l, a_l + t_{a,b} + t_{b,c}) - t_{c,d} > 0$$

Based on these constraints we can now implement a time window feasibility check by looking at two requests and attempt linking the stops of those requests in the three sequences. Naturally we start by checking if the first sequence is feasible, and if not we move on to check the third sequence and at last the second sequence, since the second sequence is the most difficult to handle.

10.3.3 Affinity values

Based on the time window constraints we continue to look at the general sequence of stops $a - b - c - d$ regarding the feasibility of inserting request j after request i in a route. In problems with narrow time windows, only one sequence of stops will be feasible, but as time windows widen, two or all three sequences may be feasible. In general we want to create a value of affinity between a route r and a not yet inserted request j. We do this by first finding all feasible insertions between j and $i \in r$ and then calculate the detour of the route by inserting j in all feasible insertions.

If the first sequence is feasible we risc driving with an empty vehicle between stops b and c. As it is not desireable to drive with an empty vehicle, we introduce a penalty factor *penalty* to be set by the operator. Thus inserting j in r after request i creates the affinity value:

$$penalty * t_{b,c} + t_{c,d}$$

In case of feasibility of sequence two or three, we use the following estimate of the detour:

$$t_{a,b} + t_{c,d}$$

This corresponds to the sum of the distances between respectively the pick up locations and the delivery locations of requests i and j. In the case of no feasible sequences, the value of affinity is set to a large number symbolizing infinity. Note that in contrast to the usual meaning of the word affinity, the larger the value the worse the affinity.

10.3.4 Initializing

The solution obtained in the first part of the planning process is somewhat critical since it will form the basis of the dynamic insertion procedure for new requests during the day. For this reason we develop a clustering technique to use the advantage obtained by knowing large number of requests in advance. The possibility of using an even more intelligent approach exist since this phase could be executed the night before, thus removing the demand for short computational time. A solution enabling the planner to interact with the automated process is however often preferrable. This is caused by the fact, that it is not always possible to incorporate all "soft" knowledge into the automated process.

The clustering technique has the advantage of beeing fast with regards to computational time, just as the principle behind clustering requests is intuitively easy to comprehend. We construct the requests by performing two steps:

- **Check feasibility in time.**
- **Check feasibility in distance.**

The time feasibility check is performed according to the constraints formulated in section 10.3.2. However we want neighboring requests to be close geographically as well as chronologically. This is done by formulating a set of constraints controlling the distance in driving time between the stops of request $i = (p_i, d_i)$ and request $j = (p_j, d_j)$. In the formulations of these constraints we use a parameter $maxridetime$ stating the maximum ride time allowed

between stops of two different requests in order for those requests to be considered neighbors. We consider two stops a and b with time windows respectively $\{a_l, a_u\}$ and $\{b_l, b_u\}$. The first constraint can then be formulated as:

$$b_u - a_l \geq t_{a,b} \leq maxridetime \tag{10.4}$$

If stop a is the delivery stop of request i and b is the pick up stop of request j (ie. the stops are of different type), then constraint 10.4 is sufficient to ensure that the two requests i and j are neighbors. However if the stops a and b are of the same type (ie. both pick up stops or both delivery stops) then two additional constraints are needed. If the stops d_i and d_j satisfy constraint 10.4, then:

$$t_{p_i,p_j} + t_{p_j,d_i} < 1.4 * t_{p_i,d_i} \tag{10.5}$$

and if p_i and p_j are the stops satisfying constraint 10.4 then:

$$t_{p_j,d_i} + t_{d_i,d_j} < 1.4 * t_{p_i,d_i} \tag{10.6}$$

The distance constrains are illustrated in figure 10.3, where constraint 10.4 is satisfied by the stops d_i and d_j. However constraint 10.5 requires all stops to be within an eclipse based on the distance of the stops in request i, and stop d_j is located outside this eclipse. Thus request i and j in figure 10.3 are not neighbors.

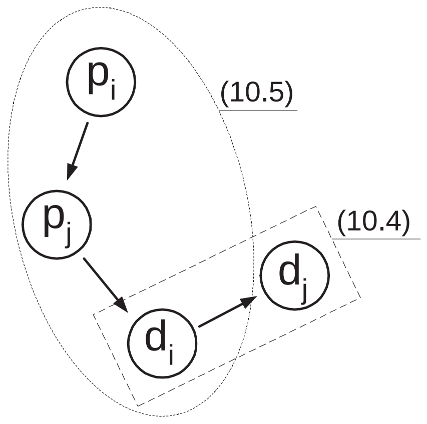


Figure 10.3: Controlling the distance between neighboring requests.

We now construct a set of clusters as described in the source code 10.1 below.

Source code 10.1 Creating clusters.

```
long i=0;
Request *pReq;
Cluster *pCluster;
list<Cluster *>::iterator itlC;

----------INSERTING FIRST REQUEST----------
pReq = FindNextNonInsertedRequest();
pCluster=new(Cluster);
Clusters.push_back(pCluster);
pCluster->listRequest.push_back(pReq);

----------INSERTING REMAINING REQUESTS----------
itlC=listCluster.begin();
pReq = FindNextNonInsertedRequest();
while(pReq!=NULL){
     if(IsNeighbor((*(*itlC)->listRequest.begin()),pReq))
     {
          (*itlC)->listRequest.push_back(pReq);
          itlC=listCluster.begin();
          pReq=FindNextNonInsertedRequest();
     }
     else
     {
          itlC++;
     }
     if(itlC==listCluster.end())
     {
          pCluster=new(CCluster);
          listCluster.push_back(pCluster);
          pCluster->listRequest.push_back(pReq);
          pReq=FindNextNonInsertedRequest();
          itlC=listCluster.begin();
     }
}

----------SORTING CLUSTERS BY NUMBER OF REQUESTS----------
SortClusterList();
```

The procedure of creating clusters is fairly simple. We start by creating the first cluster and insert the first request i into that cluster. Then we examine the next non inserted request j to check weather it is a neighbor of the first request in the first cluster. If the requests are neighbors, we insert j in the first cluster. If not we create a new cluster containing j. Then we examine the next request to check if it is a neighbor of any of the requests placed first in each created cluster. If a pair of requests are neighbors we perform an insertion, else we create a new cluster for the not inserted request.

Creating clusters according to the procedure described by source code 10.1 is not seen anywhere else in litterature. Note that all requests in a cluster are neighbors with the request that created the cluster, but not necessarily with any other request in that same cluster. The reason for this is an attempt to keep cluster size small so as to leave room for the insertion procedure described later in this chapter. If a request is inserted in a cluster by beeing a neighbor of any request already in the cluster, the clusters in themselves would represent almost fully constructed random routes, since the clusters form the basis of the routes.

After having created the clusters, they are sorted according to the number of requests they contain. In principle they are sorted with the largest cluster first in a descending order. However when problems are large, the number of clusters is also large, which makes the sorting procedure expensive in computational time. For this reason the sorting procedure just places the largest cluster as the first cluster in the list without changing the order of the remaining clusters.

Source code 10.2 Initializing routes.

```
CRoute *pRoute;
CRequest *pR1, *pR2, *pDepot;
list<CCluster *>::iterator itlC;
list<CRequest *>::iterator itlR;
----------INITIALIZING A FIXED NUMBER OF ROUTES----------
for(iterroutes=0;iterroutes<nbofroutes;iterroutes++){
     pRoute=new(CRoute); vRoute.push_back(pRoute);
     -----INSERTING DEPOT FIRST IN ROUTE-----
     pDepot=CreateDepot();
     InsertInRoute(pRoute->lId,pDepot,NULL);
     SortClusterList();
     if(!listCluster.empty()){
          itlC=listCluster.begin();
          pR1=*((*itlC)->listRequest.begin());
          pR2=p_Depot;
          while((!IsFeasibleInCapacity(pRoute))
              &&(itlC!=listCluster.end())){
               itlC++; pR2=p_Depot;
               pR1=*((*itlC)->listRequest.begin());
          }
          if(itlC!=listCluster.end()){
          -----REQUEST CAN BE ADDED TO ROUTE-----
               InsertInRoute(iterroutes,pDepot,pR1);
               -----ATTEMPTING TO INSERT ALL FROM CLUSTER-----
               itlR=(*itlC)->listRequest.begin();
               while(itlR!=(*itlC)->listRequest.end()){
                    if(IsFeasibleInRoute(
                       (*itlR),pRoute).first[0]!=0)
                         InsertInRoute(iterroutes,(*itlR),pR1);
                    pR1=(*itlR); itlR++;
                    (*itlC)->listRequest.remove(pR1);
               }
               else itlR++;
          }
          -----REQUEST NOT FEASIBLE IN THE ROUTE-----
          else itlR++;
     }
     if((*itlC)->listRequest.empty())
          -----ALL REQUEST FOM CLUSTER INSERTED IN ROUTE-----
          listCluster.erase(itlC);
}
```

Source code 10.2 describes how routes are initialized on the basis of the created clusters. We first decide on the number of routes we want to initialize. This number is small, if we want to minimize the number of vehicles used, or a fixed number of available vehicles when fleet size is known. In a straight forward manner we now start the construction of the first route by inserting the depot into that route. Using the first request in the largest cluster, we then insert this request into the route after the depot. This is followed by an attempt to insert all of the remaining requests of the largest cluster into the same route. If it is not feasible to insert a request, it is left in the cluster. In case all requests from the cluster are inserted, the cluster is deleted.

Results show, that even when constructing clusters as groups of requests which are neighbors with one specific request in the group, almost all requests from a cluster will be inserted into the same group. This is not surpricing since the largest clusters are concentrated in areas with high density of population generating many requests within the same time horizon. There will be a more complete examination of this in chapter 11.

10.3.5 Insertion

After having generated and used clusters for initializing the routes, we are left with a number of requests not yet inserted. To proceed with the insertion we perform a series of greedy steps based on the affinity values described in section 10.3.3. The pseudo code for generating affinity values is shown in source code 10.3. Note that this time the code is heavily manipulated to improve readability, since the actual source code is based on complex object oriented structures. The detours for inserting a request into a route is found by trying an insertion of the request after all other requests in the route. Naturally, a great part of these insertions are claimed infeasible just by looking at the time windows. We get the lowest affinity value between a not inserted request and a route by pairing the request with all requests in the route choosing the best value.

Source code 10.3 Creating affinity values.

```
vector<CTimeSpan> *vL;
CTimeSpan ctsRes=infeasible;
pair<vector<CTimeSpan>,CRequest *> pvRes;
vector<CTimeSpan> vCts;
CRequest *pNULL=NULL;
vector<CRequest *>::const_iterator itvR;
for(itvR=vRequest.begin();itvR!=vRequest.end();itvR++)
{
   vL=new(vector<CTimeSpan>);
   if(the request is not in a route){
      for(long i=0;i<p_MCSolRoutes->vRoute.size();i++){
         pvRes=IsFeasibleInRoute(request,route);
         if(the request is feasible){
            if(it is the first sequence of stops){
               CalculateDetourSeq1(route,request);
                  if(IsVehicleEmpty(by inserting request into route))
                     ctsRes = penalty * detour;
                  else
                     ctsRes = detour;
            }
            else{
               if(it is the second sequence of stops){
                  CalculateDetour2(route,request);
                     ctsRes = detour;
               }
               else{
                  CalculateDetour3(route,request);
                     ctsRes = detour;
               }
            }
         (*vL).push_back(ctsRes);
         }
         else
            (*vL).push_back(infeasible);
      }
   }
   vvAffinity.push_back(*vL);
}
```

In order to save computational time, we save as much information as possible from the feasibility check. This information is stored in a vector containing:

- The feasible sequence of the stops.
- The distance between the stops.
- The request in the route after which the non inserted request will be inserted should it be chosen for insertion.

This information allows a quick calculation of detour without having to perform any of the shortest path calculations again. Later during the actual insertion phase, this information is also available to allow for quick insertion of requests into routes.

To find the next request to be inserted, we simply find the lowest value of affinity between a request and a route in the affinity matrix. This request is then inserted, the route is updated by rearranging the stops in the route, and the affinity matrix is updated. Naturally it is only necessary to update the column in the matrix describing the updated route. The row concerning the newly inserted request is simply deleted from the matrix.

In this implementation of the insertion procedure, we have the option of pausing after each insertion to perform a trouble check. It might not always be prudent to just choose the lowest value of affinity as an insertion guide. For instance a situation where a request only has one route it can be inserted into to keep feasibility. The trouble check would then use this information to insert the request and update the affinity values for the route. The trouble check is optional and not always desireable, since the affinity value of the troublesome request might be high enough to indicate an extraordinary decrease in the level of service given to the requests already in the route.

The desireability of the trouble check is dependent on the characteristics of the DAR transportation system considered. If we consider a fixed number of vehicles, the trouble check is important to ensure service to as many requests as possible. In this case we might also want to even out the load of the vehicles by balancing the number of requests on the routes. Even loads on the vehicles will make the routes more flexible with regards to accommodating future not yet known requests, thus maximizing utilization of the vehicles in the entire planning horizon. With an unlimited number of vehicles (eg. when taxis are used as a supplement to busses) the trouble check is not entirely necessary.

The insertion procedure continues as long as there are feasible insertions between routes and not yet inserted requests. When operating with a fixed number of vehicles, we might have to deny service to some requests, but with supplementing vehicles, new routes for the difficult requests will be initialized and dispatched to idle vehicles. To summerize, figure 10.4 illustrates the insertion procedure with the trouble check.

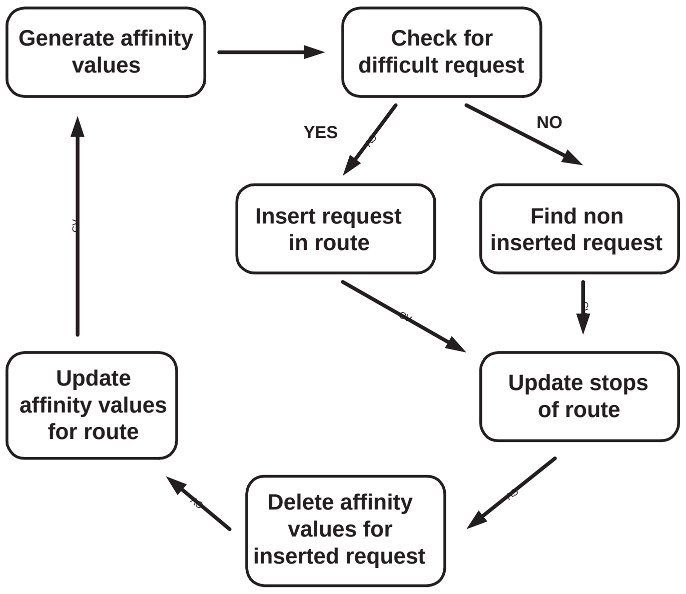

Generate affinity values
Check for difficult request
YES
NO
Insert request in route
Find non inserted request
Update affinity values for route
Update stops of route
Delete affinity values for inserted request

Figure 10.4: The insertion procedure with the additional trouble check.

When the insertion procedure is finished, the static problem is solved, and the resulting routes can be given to the drivers of the vehicles. Since the routes are based on the actual road network, it is of course possible to append driving directions and map printouts to the list of requests each driver must service. We are however dealing with the dynamic case of DAR problems here, indicating that information will change during the day. Not only will new requests arrive, but disruptions might also occur during the planning horizon. Possible disruptions could be a flat tire, causing the bus to get behind schedule, or road construction not yet registered in the database storing the road network.

When new requests are made, the insertion procedure is applied to the request. A vector of affinity values between the request and the existing routes is established, and a greedy insertion is chosen. Denying service to a dynamic request is not as serious as when solving the static problem, since the passenger is notified immediately upon making the request. In the static problem, passengers have already been promised service days in advance. When dealing with dynamic requests it is also noteworthy that the level of service parameters can be set individually for each request. This enables the planner to negotiate the level of service with the passenger when recieving the request to suggest possible insertions.

10.4 Adding McCluster to InfoRoute

To enable the use of the database for storing the requests, display of routes on a map, and the shortest path calculations on the underlying road network, we need to add the McCluster module to InfoRoute. The structure of the InfoRoute software is shown in figure 9.12. We add McCluster as shown in figure 10.5 where the communication between McCluster and InfoRoute is shown.
In figure 10.5 we see a good example of how easy it is to add algorithms to the InfoRoute software. Only one class in the InfoRoute software needs to be available to enable access to the total functionality of InfoRoute. To access the functionality of McCluster from InfoRoute, some additions have to be made to the menus of the user interface. These additions are implemented in the CInfoRouteApp class with little difficulty. An example could be the menu item “Import Requests”. We add this item to the existing menu structure with the command handler source code:

```
void CInfoRouteApp::OnMcclusterImportRequests()
{
    p_InfoRouteDSP->p_McDSP->ImportRequests();
}
```

In InfoRoute we have the ability of adding a request to the database.

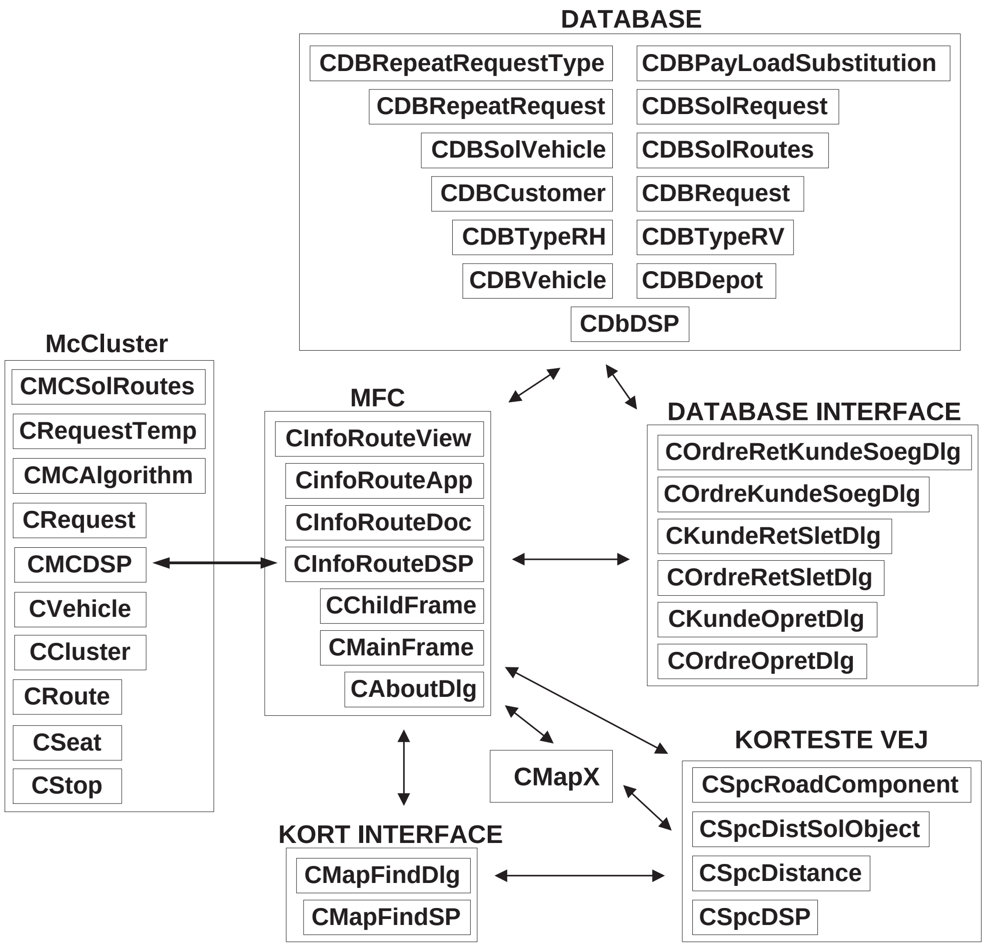


Figure 10.5: The structure of InfoRoute with McCluster added.

Using the McDSP class, we can now solve the insertion of the request into the route in the same step by adding the following line to the source code:

```
void COrdreOpretDlg::OnOpretordre()
{
          .
          .
          p_InfoRouteDSP->p_McDSP->InsertRequest();
          .
          .
}
```

These small pieces of source code capture the essence of introducing interface classes (in our case DSP classes) to all modules in a software program. If the InfoRouteApp class needs to communicate with other modules it calls its corresponding DSP class (InfoRouteDSP). InfoRouteDSP then calls the DSP class of the module with which communication is required (McDSP), and McDSP then assigns the required tasks to its internal classes. Naturally the force of designing software this way has been further explored to obtain a standard of interface classes. This standard is called COM (Component Object Modelling) or DCOM (Distributed COM). COM and DCOM enables software modules running on different operating systems and implemented in different languages to communicate. Although a discussion of the advantages of COM and DCOM is beyond the scope of this thesis, the implementation of InfoRoute and McCluster in a COM like manner is worth mentioning.

Chapter 11

Results

Results will be based mainly on a test dataset generated from addresses in Skovbo Kommune (the municipality in which Jensens Turisttrafik operates). InfoRoute with McCluster (in the following just InfoRoute) was also introduced to Jensens Turisttrafik (JT) and evaluated by the person responsible for planning the DAR routes.

11.1 Jensens Turisttrafik

As stated earlier in chapter 2, JT is interested in automating some of the planning process for the DAR routes they operate. The routes are partly decided by the drivers, who recieve a list of passengers (with addresses) they must serve during their shift. This list is made by the planner manually based on experience. As this procedure has proven very efficient, it is also depending heavily on the planner. JT therefore wants software to help at the call center, when the planner is away. When a call is recieved, the person on duty should be able to enter address and time of service on the computer. The computer then needs to instantaneously respond with an accept of the request or a denial of service. Also the computer needs to allocate the request to an existing route, so the person on duty can inform the correct driver.

It might seem unnecessary for InfoRoute to show the routes graphically and sort the stops, when routes are not a required part of the output requested by JT. In order for InfoRoute to be able to accept or deny a request it is however necessary to calculate the routes anyway to keep an updated solution at available.

The planner at JT was very positive about the functionality of InfoRoute. One feature that is especially usefull is the ability to lookup an address on the map. This feature can be used in the manual planning process to guide the planner in allocating a new request to a route. A demonstration of the automated planning included in InfoRoute convinced the planner that a further developement of the user interface could make the software valuable to JT. The further developement mentioned is actually a downgrading of the user interface. In order for InfoRoute to be of true value, some information will have to be disregarded. Dialogues for entering customer and request information are too complicated to be used in the daily planning, since practical use requires a lot less information. If just the address and time would have to be entered, the ease of using InfoRoute would be improved. JT suggested that automated communication between InfoRoute and the drivers would be of great help.

To sum up, JT would be pleased to continue cooperating in developing a final version for them to use in their planning process. JT suggested a reduction in the available functionality in order to simplify the use of InfoRoute, but also extensions in the form of automated communication between InfoRoute and the drivers were given as a possible improvement.

11.2 Data generation

We developed a data generator to generate problem sets for testing our algorithm. First we read a list L' of all addresses in the road database, which represents the potential pick up or delivery points (service points) for the requests in the system. Let N denote the number of service points in L' and let L'_i denote the ith service point. L' is sorted according to the road name. Next we shuffle the elements of L' in a random order using a method in the Standard Library of C++. We refer to the random order list as L. Now we can take any number $n \geq N$ of service points from L and generate a request for each point i, $i = 1, \ldots, n$.

Let $\mathrm{random}(k)$ be a function which takes an integer $k > 0$ and returns a random number between 0 and $k-1$.

The requests are generated by the following procedure. For each service point L_i we generate the type of service $T = \text{random}(3)$ and an additional random service point $L_j, j = \text{random}(N) + 1$ generated from the list of all service points:

- $T = 0$: return trip, i.e. trip from pick up point L_i to delivery point L_j and back to L_i,
- $T = 1$: trip from pick up point L_i to delivery point L_j,
- $T = 2$: trip from pick up point L_j to delivery point L_i.

Next we generate the demand $d_i = \text{random}(3) + 1$, i.e. the number of passengers requiring pick up or delivery at the service point.

Finally we generate the pick up or delivery time t for each request according to the type of service. We generate service times in a planning horizon of 960 minutes corresponding to the time between 06:00 and 22:00. If we are considering a return trip we generate a delivery time in the morning (0-420) and a pick up time in the afternoon (540-960). For pick up or delivery trips we generate service times in the entire planning horizon (0-960):

Type of service	Service time
0	$t_{delivery} = \text{random}(420) + 1$
	$t_{pickup} = 540 + \text{random}(420) + 1$
1	$t_{delivery} = \text{random}(960) + 1$
2	$t_{pickup} = \text{random}(960) + 1$

11.3 Simulated data

The dataset resulting from the method described in the above section 11.2 is illustrated in figure 11.1, where the geographical location of the pickup and delivery stops are marked with stars on the road network. The location of the stops seems to be fairly representative of a realistic problem, where there is a large concentration of stops in densely populated areas with comparatively few and widespread stops in the rural areas.

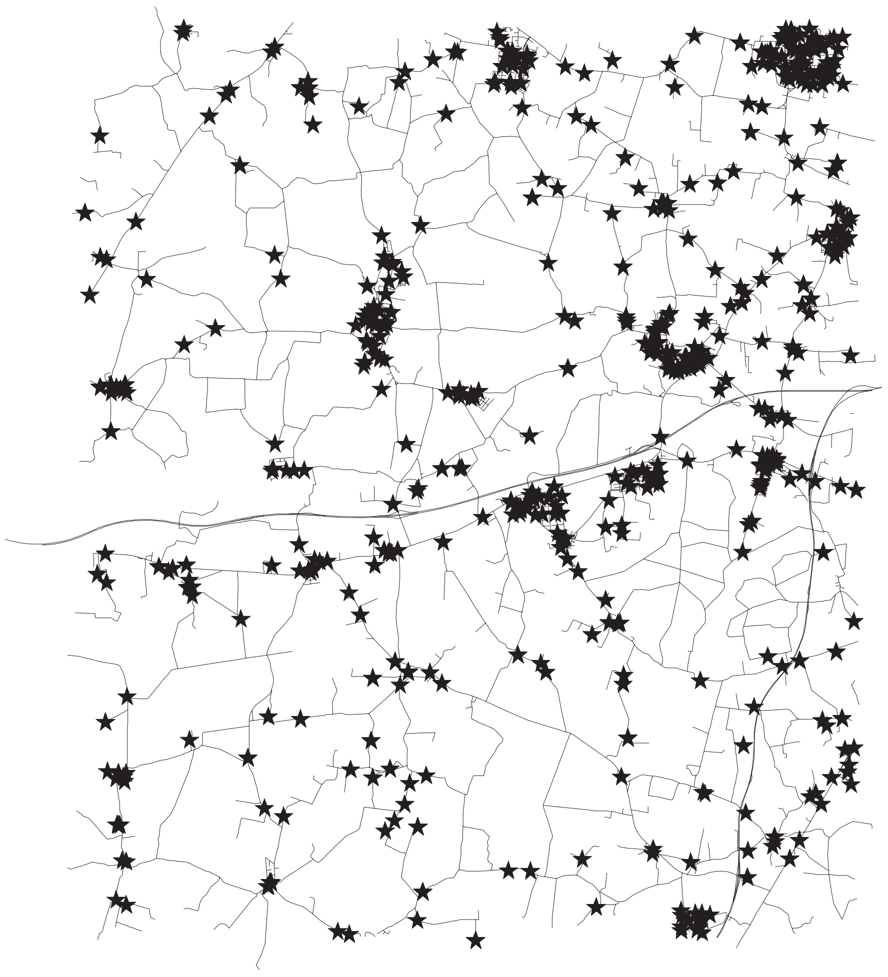


Figure 11.1: The geographical location of the stops belonging to the 300 generated requests.

In figure 11.2 the time windows corresponding to the stops are illustrated. The figure shows a high concentration of stops from 7am to 12pm and again from 3pm to 21pm. Around these two time intervals, the number of stops is limited. This is a behavior we expect to see in realistic problems as well, since people often leave home in the morning to come back in the afternoon/evening.

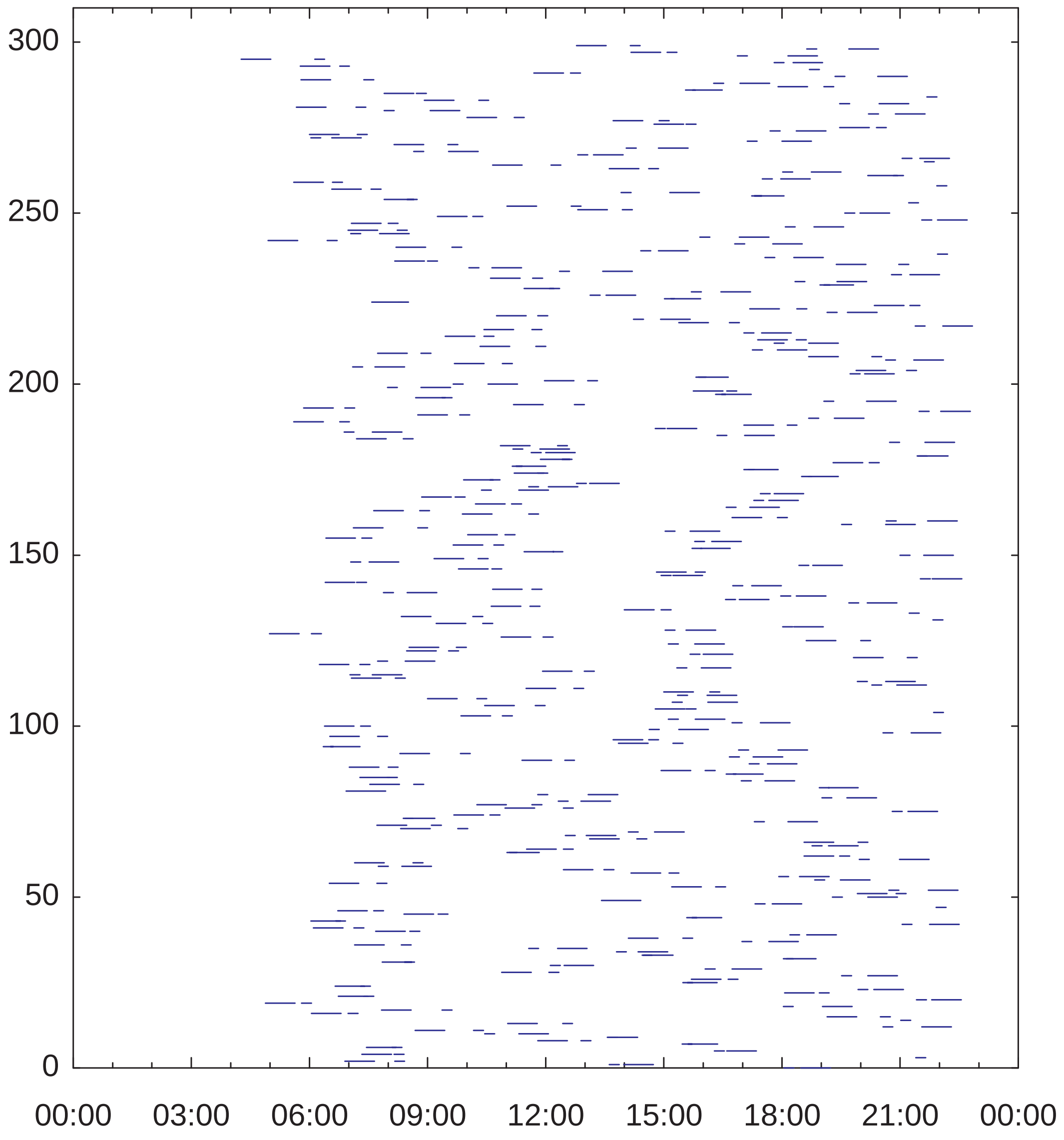


Figure 11.2: Time Windows of the 300 generated requests.

The test dataset does not incorporate the special cases seen at JT, but focuses mainly on more traditional DAR problems with a mix of static and dynamic information. It does not seem possible to generate a test dataset mirroring the situation at JT, but the regular DAR problem is part of almost any realistic situation.

The requests in the test dataset are mainly treated as static information, since in practical problems the requests are known up

to one hour in advance. The solution time of the McCluster algorithm is measured in seconds, giving amble time to solve the static problem each time a new request is added. However, to observe the effect of the cluster generation, this chapter includes a test where 50% of the requests are considered static information, leaving the last half as dynamic requests. The dynamic requests are still known one hour in advance, but instead of solving the entire problem in each instance of a new request, the request is inserted using the insertion scheme described in chapter 10.

Table 11.1 shows results from running McCluster on five instances of the constructed dataset. The five instances are derived directly from the full 300 request dataset simply by extracting the first 25 requests, the first 50 requests etc. Since the requests are randomized when generated, they should be similar in structure. Again trying to simulated part of the situation at JT, we have chosen to use 3 vehicles with 20 seats each, driving at 21.6 km/t in average, based on the actual speed of the vehicles at JT. In near future it will be possible to use a better estimate of the vehicle speed, since this parameter is added to the characteristics of the road network.

No. of req.	Max. cluster size	Avg. cluster size	No. of req. in init.	Dist. driven (Km)	Drive time (h:m)	Idle time (%)	No. of rejected req.
25	4	1.67	10	596.86	27:38	52	0
50	6	2.38	17	1024.48	47:26	17	4
100	11	3.13	32	970.43	44:56	21	51
200	23	4.17	57	1014.43	46:58	18	141
300	34	4.69	67	1160.53	53:44	6	231

Table 11.1: Results with three vehicles driving 6 m/s (21.6 Km/t).

When fixing the number of vehicles the algorithm seeks to maximize the number of requests to be served while minimizing the driven distance. The amount of idle time for the vehicles is usually a good indicator of how well the vehicles are utilized. However, this is not always the case as shown in figure 11.3, where the idle time from table 11.1 is shown as a function of the number of requests.

Idle time in the solution based on 50 requests is lower than that of both 100 and 200 requests, while the actual number of requests served increases. The reason for this behavior is a combination of the following:

- **Idle time for an empty vehicle is “free”.**
- **Limitations on capacity.**
- **Limitations on excess ride time for passengers.**
- **More requests in near proximity.**

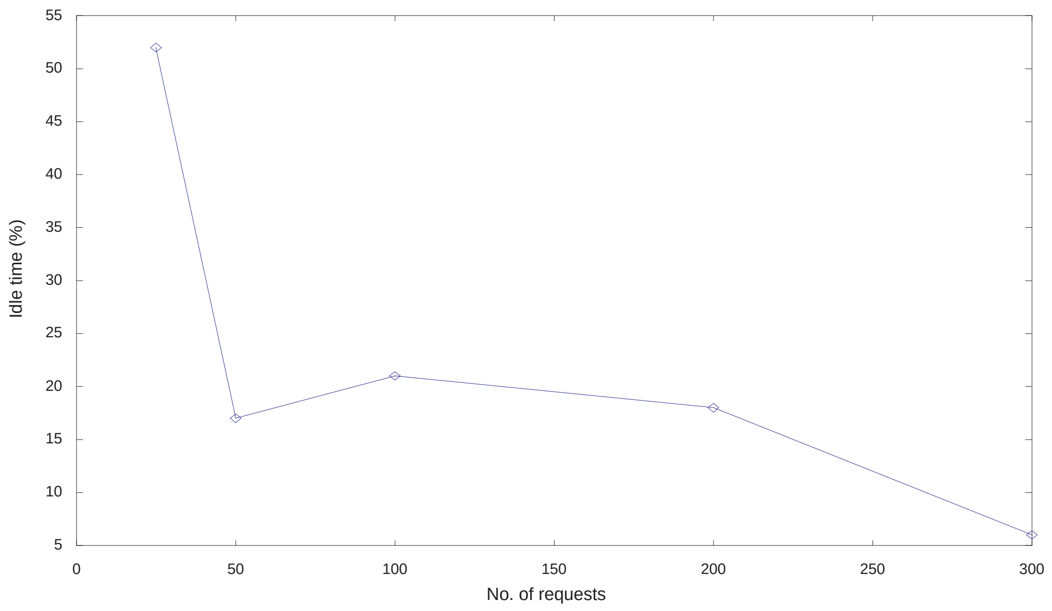


Figure 11.3: Idle time with three vehicles as a function of the number of requests.

When including more requests in the problem, we get larger clusters of requests grouped close together in time and space. This enables the algorithm to replace requests that are “hard to serve” with requests resulting in less driven distance. This can prove a problem in realistic situations if the number of vehicles is too small, since the passengers living in remote locations will be deselected at all times.

As stated before we use an estimate of the average speed of the vehicles. This estimate is critical when maximizing vehicle utilization and the number of accommodated requests. In table 11.2 the results when doubling the average speed of the three vehicles is shown.

No. of req.	Max. cluster size	Avg. cluster size	No. of req. in init.	Dist. driven (Km)	Drive time (h:m)	Idle time (%)	No. of rejected req.
25	6	2.78	14	573.88	13:17	77	0
50	10	4.17	25	1063.08	24:36	57	0
100	21	5.26	52	2230.44	51:38	9	1
200	37	8	93	2272.52	52:36	8	87
300	58	9.38	115	2385.66	55:13	3	178

Table 11.2: Results with three vehicles driving 12m/s (43.2 Km/t).

Naturally a lot more requests can be served when doubling the speed of the vehicles, but when reaching a problem size where vehicle utilization is close to maximum, further increase in problem size results in almost the same number of additional requests served. This is illustrated in figure 11.4, where the number of requests served at the two different average speeds is considered. The extra speed also makes it possible to avoid vehicle idle time, thus halving this from the results driving at 6 m/s.

As the results show, the estimate of vehicle speed has a large impact on the quality of the solution. However, when speed is incorporated in the digitized road map, we be able to estimate this with much greater accuracy since each road will have a speed stamp.

On the more strategic level, McCluster can be used to minimize the number of vehicles. The results of this is shown in table 11.3. Once again we use an average speed of 6 m/s, which means that none of the instances can be solved with just one vehicle, since we have to serve all requests.

The results from minimizing the number of vehicles can be used for many purposes. In Denmark DAR problems are often based

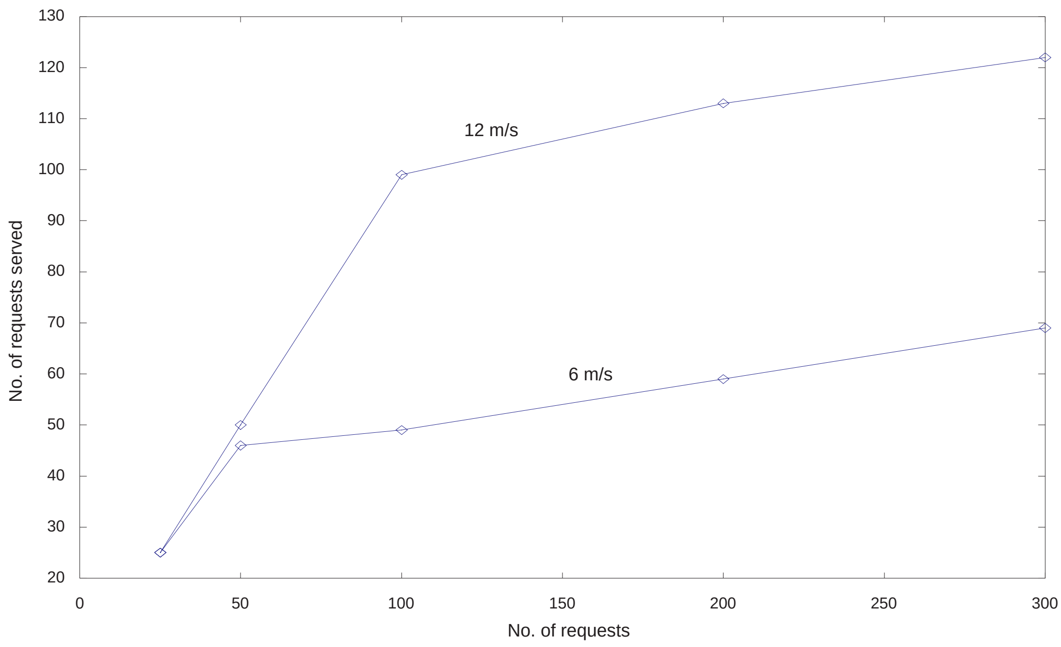


Figure 11.4: Number of requests served driving 6 or 12 m/s.

No. of req.	Max. cluster size	Avg. cluster size	No. of req. in init.	Dist. driven (Km)	Drive time (h:m)	Idle time (%)	No. of veh.
25	4	1.67	8	574.39	26:36	30	2
50	6	2.38	22	1126.49	52:09	31	4
100	11	3.13	55	2144.09	99:16	13	6
200	23	4.17	141	4176.31	193:21	22	13
300	34	4.69	220	6275.04	290:31	24	20

Table 11.3: Results when minimizing number of vehicles.

on regular school bus routes etc. beeing opened for public use. In this case we again have a lot of requests known in advance, and by finding the minimum number of busses needed to serve these requests, we also get an estimate of how well the busses are utilized. From the results shown in table 11.3 we see that requirering service to all requests can result in a somewhat large percentage of vehicle idle time. This figure gives a general idea of how many more requests can be handled with current busses, or how many

requests must be denied service in order to save a bus.

McCluster does not consider load balancing. In some practical cases this can be a problem, since a desired result of the automated planning is for drivers to serve an equal number of requests. For this reason we have run tests with 5 vehicles driving at 6 m/s. Results are shown in table 11.4, where average vehicle load is the number of requests served by the vehicle.

No. of req.	No. of req. in init.	Avg. no. of req. per vehicle	Min. no. of req. per vehicle	Max. no. of req. per vehicle	Deviation from avg. (%)	No. of rejected req.
25	14	5.0	2	14	180	0
50	27	10.0	5	20	100	0
100	49	16.4	12	21	38	18
200	84	21.0	16	27	38	95
300	95	22.0	20	27	23	190

Table 11.4: Deviation of vehicle load with 5 vehicles.

The results from not considering load balancing are very obvious when the number of vehicles is more than adequate to serve all requests. Typically the first vehicle recieves the largest cluster of requests, thereby enabling this vehicle to server many more requests than the next one in line during the planning process. The graph of the results are shown in figure 11.5.

With few requests and a large number of vehicles, the first vehicles will always serve the larger part of the requests. The last vehicles will then serve fewer requests, but it will also be the most "difficult" requests, thus the last vehicles will perform the longest routes. The positive sideeffect of the lack of load balancing, is the ability to spot areas in which it might be prudent to establish a regular bus route. If the first route is small geographically and almost similar every day, it might be reasonable to exclude the area from the DAR transportation system.

While the clustering part of McCluster is responsible for the uneven load of the vehicles by assigning larger clusters to the first vehicles, it also enables the algorithm to quickly identify geographical

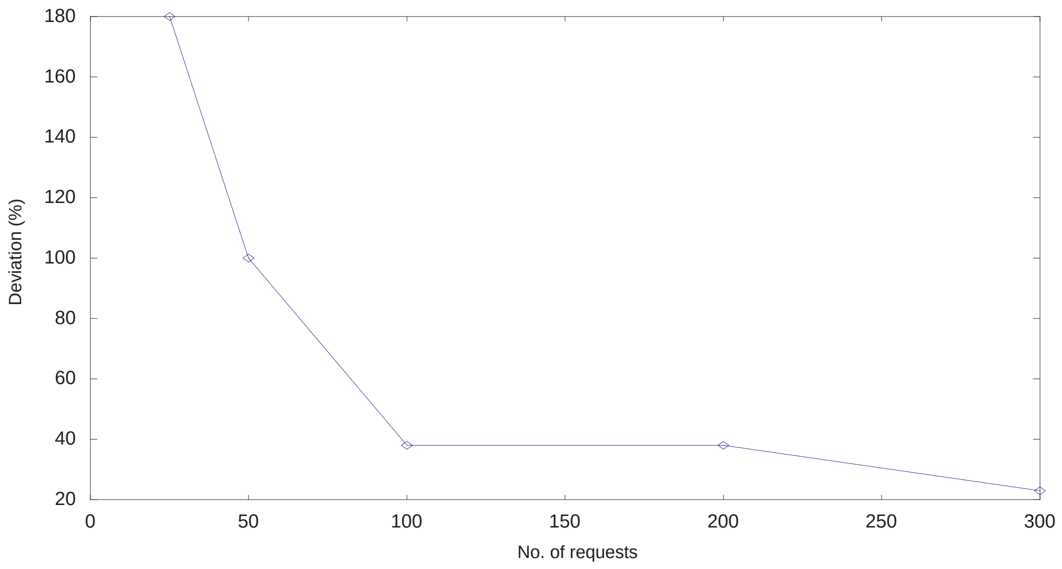


Figure 11.5: Deviation of vehicle load from average.

No. of req.	Max. cluster size	Avg. cluster size	No. of req. in init.	Dist. driven (Km)	Drive time (h:m)	Idle time (%)	No. of rejected req.
25	3	0.91	12	582.44	26:57	72	0
50	4	1.67	18	974.89	45:08	48	0
100	6	2.38	26	1848.65	85:35	11	19
200	11	3.13	49	1935.85	89:37	6	103
300	27	3.82	74	1973.08	91.21	4	193

Table 11.5: Results with five vehicles and 50% known requests.

areas where it is "inexpensive" to service requests. The question is how much the clustering procedure affects the overall solution. In table 11.5 results from tests with half the requests known in advance are displayed.

Comparing the results from the partly dynamic case with the results from table 11.4, we note only a slight increase in the number of rejected requests as shown in figure 11.6. If the extra rejected customers are enough to justify the implementation of a clustering technique is hard to answer, and will depend on the structure of

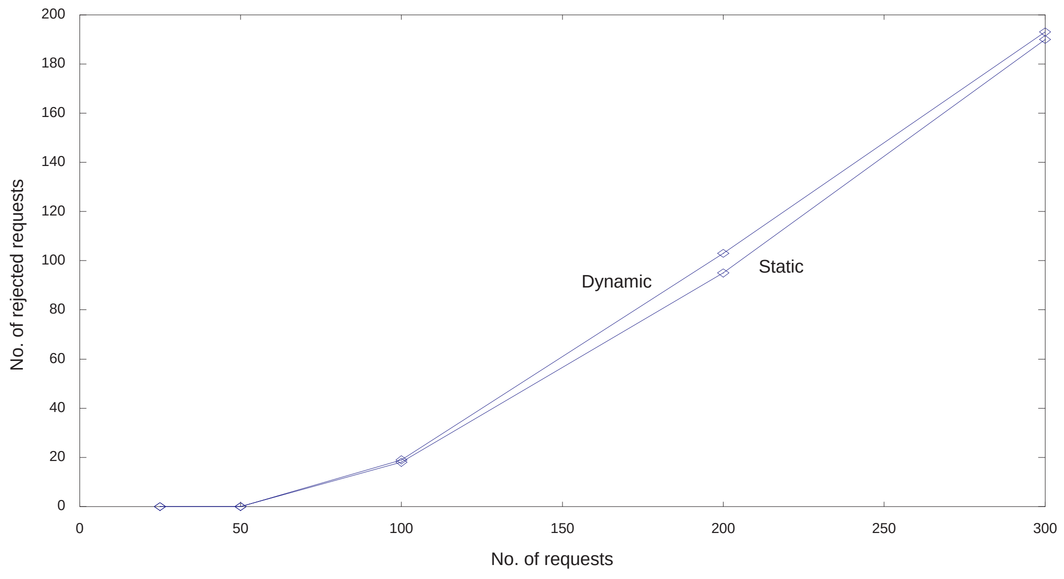


Figure 11.6: Number of rejected requests when considering the problem as static vs. 50% dynamic.

the problem to be solved. When requests are largely clustered geographically, a clustering technique will give a very good starting point for the algorithm. On the other hand, when requests are dispersed throughout an area, using the largest cluster as an initial route may not prove quite as effective as simply choosing "most difficult" or "easiest" requests first.

The tests are performed on a PC with an AMD Athlon 1.2GHz processor and 512MB RAM running Microsoft Windows 2000 Professional. CPU time used to solve the three vehicle 300 request problem was around 20 sec. where half the CPU time was used to solve the SPP. Insertion of a dynamic request into the problem used less than one CPU second.

Whereas McCluster solves the problems fast, the display of routes on the road network is somewhat expensive. Displaying the routes of three vehicles servicing around 20 requests each takes around two minutes of CPU time. The reason for this is the large number of memory allocation and deallocation MapX uses to display data, and it is expected to improve greatly in future versions of MapX.

11.4 General comments

To evaluate the results obtained it is important to keep the framework of the algorithm in mind. While one of the main goals with this project was to incorporate and use digitized road networks in a graphical environment with automated planning for DAR transportation systems, this turned out to be a non-trivial task. The development of the GUI is in itself an entire projekt, which should be carried out in close cooperation with future users, and making MapX work within the GUI was difficult because of insufficient documentation.

When seeking to develop a software package to be used in a practical environment, the algorithm for the automated planning process is not nearly as important as its surroundings. However, there are some aspects of the design of the algorithm that needs attention, such as simplicity and reasonability. It is more important that the structure of the algorithm seems logical to the planner than that the results obtained are the best possible. The planner, who is to use the software, must "trust" the software in order for the project of moving to an automated planning process to become a success.

As the development of the GUI is normally not performed in academia nor a credited part of an academic project, it is, however, very important in real life planning situations. The GUI InfoRoute developed here does not fulfill the requirements in real situations, but is meant as a test environment for developed algorithms. InfoRoute is easily expanded to practical useability, and will in its present form be more than adequate for testing puposes.

Chapter 12

Improving solutions

Depending on the degree of dynamism in a practical DAR transportation system, there will be a certain amount of time in which the computer handling the automated planning process is idle. This CPU idle time could be used to improve solutions obtained by McCluster since these are found with focus mainly on CPU speed rather than solution quality.

Using the specifications from Jensens Turisttrafik, we receive about 50 calls during a 10 hour workday. Each of these calls uses about half a second of CPU time including the administrative functionality (database, shortest paths and mapping). We will leave the computer idle as long as a request is beeing entered at the call center, which is a procedure of approximately two minutes. During these two minutes, the request is negotiated with the passenger over the phone, and entered into InfoRoute. Spending two minutes of CPU time on each dynamic request thus leaves eight hours and twenty minutes available CPU time for improving solutions.

Improving solutions can be seen in various ways depending on the objective:

1. Maintaining an even load of requests on vehicles in order to maximize accommodation.

2. Maximizing level of service by avoiding excess ride time and idle time with passengers in a vehicle.

3. Minimizing total drive time while maintaining service to a maximum number of requests.

4. Minimizing total idle time to improve utilization of vehicles.
5. Minimizing number of vehicles.

When considering a DAR transportation system performing regular public transportation, it is likely that only the first three objectives will be of any interest. Regular public transportation systems usually operate with a fixed number of vehicles on the operational level. When using a fixed number of vehicles in a dynamic environment, it would be natural to maintain an even load on the vehicles hoping that maximizing accommodation will lead to better utilization of the vehicles. In a static environment the diversity of the load is of no consequence, so the objective could be to minimize distance driven and/or maximize level of service. The level of service in public transportation is usually seen as a constraint more than an adjustable parameter to be included in the objective.

Specialized public transportation has quite different objectives from the regular public transportation. Systems with a variable set of vehicles are often used, and even on the operational level, it is possible to cut costs by minimizing the number of vehicles used. This characteristica is caused by the use of taxis as suplementary vehicles in order to cut down fleet size to an absolute minimum. In the case of taxis it is also important to minimize the total distance driven, since the distance has direct influence on the cost of operations.

The remaining part of this chapter will focus on possible heuristics for improving initial solutions with regards to the objectives stated above. Wheater or not we are dealing with regular or specialized public transportation systems is of no consequence to the following discussion.

One of the most obvious methods of improving solutions is a meta heuristic like Tabu Search. The reason for this is the well known neighborhood structure of the DAR Problem. Creating and exploring neighboring solutions based on swapping of two neighboring customers is simple, although implementing a swapping procedure on DAR problems is somewhat more complicated than with other VRP's because of the precedence restrictions. When swapping requests between routes it does also seem natural to consider mainly requests where the difference of the detours (when including the request in a route) for two or more routes is small. Tabu Search can be used to improve solutions with regards to any of the objectives.

The solution found by McCluster might include large deviations between the number of requests allocated to the different vehicles as discussed in chapter 11. This could lead to inefficient solutions when a new request is to be considered. The vehicle operating in closest proximity to the new request might have a larger load than other vehicles, thus causing the request to be served by a vehicle normally operating an area far from the position of the new request. To avoid this situation we can reuse some of the functionality of McCluster by creating affinity values between all requests on the "overloaded" route (the route with highest number of requests allocated) and the other routes. We then use the greedy procedure to keep transferring requests until an adequate size of the route has been reached. To avoid overloading a different route in the process, we specify a maximum number of requests allowed on the routes.

Excess ride time is used in McCluster as an externally given parameter not to be exceeded. To minimize this value in order to obtain a higher level of service, we could penalize excess ride time in the creation of affinity values. However this would not influence the initialization of routes, which is based on generated clusters. It does not make sense to include excess ride time in the construction of clusters, which leaves the choise of allowing only small clusters to add importance to the affinity values.

There are other ways to improve the level of service besides minimizing excess ride time. For example narrowing the time windows. A more exact estimate of actual departure and arrival times is seen as a considerable improvement to the level of service by the passengers. When a comparatively large number of requests are known in advance, or with intelligent forcasting (see section 12.1 later in this chapter) it should be possible to narrow time windows without severely damaging the solution space. In transportation systems with an interactive feed back method (email, sms or phone) the exact departure and arrival times can be communicated to the passenger in advance.

Minimizing total drive time and/or minimizing total distance driven is probably the most widely used objective when solving DAR Problems (or any other VRP). Again swapping of requests between routes can be used to obtain better results if implemented as an improvement to solutions obtained by McCluster. It is also possible to guide McCluster directly when constructing the solution by penalizing distance harder in the creation of affinity values.

The last two objectives regarding idle time and number of vehicles are impossible to separate, since they both aim to improve vehicle utilization. A simple method of reaching these objectives is to eliminate the smallest route by reallocating requests from that route to all other routes. If there are still requests in the smallest route when other routes are filled, a strategy of space creation on the other routes can be applied. Consider a request i from the route r^* to be eliminated. We then find the request j from route $r \in R \setminus \{r^*\}$ that when removed from the route leaves space for request i. We now start a chain reaction by trying to insert j in another route. If the insertion of j expells a request we try inserting that request in another route etc.. If in a certain number of iterations we still have a broken chain, we roll back the procedure, and choose another route for insertion of i. This might at first glance seem like a never ending procedure, but the number of possible insertions is actually quite limited because of precedence, time, excess ride time, and capacity constraints.

12.1 Forecasting

In a dynamic environment forecasting can play a significant role in obtaining good solutions. Depending on the accuracy of a forecast, solutions obtained on data including expected future requests can reach a level of quality near the solutions to the corresponding static problems. There are several statistical methods of forecasting based on historical data of which data aggregation is one of the more simple methods.

The method of data aggregation devides an area into several smaller units. We can then calculate the probable number of stops to be expected within a certain time in that unit. These probable stops are then aggregated to one stop with a summerized demand (each stop has a demand d for pick up and $-d$ for delivery). The aggregated probable stops are situated in the center of their corresponding unit, and added to routes as if they belonged to actual requests. The difficult part of the aggregation is maintaining data integrity. Since we operated with both pick up and delivery stops, we need to pair the aggregated stops according to expectations.

The information obtained by using forecasts can also be usefull in time intervals with traditionally very few requests. In these time intervals, a vehicle spends a substantial amount of time waiting idle for the next request. This idle time could be used to position the vehicle close to the area of the expected next pick up. This would lower response time for the vehicle and allow a narrowing of the notification time interval (the time before earliest pick up a request has to be made). This again would be an improvement of the level of service offered by the transportation system to passengers.

12.2 Graphical User Interface

In order for InfoRoute to become of practical use, there are some small changes and expansions to implement. First, the map must be displayed in more detail with forests, buildings etc. to help locate addresses fast. Secondly the street offset for addresses will need to be displayed on the map in order to give DtD transportation planners a better estimate of the actual ride time to addresses.

With regards to the underlying database, it is preferable to include algorithmic parameters in a separate table to avoid textfiles. It is not included in the present version since it would only be confusing in a test environment for various algorithms. Also functionality for starting a new planning horizon needs implementation. While data is copied manually between days, an automated process in which selected data is copied to a new database at the beginning of a new planning horizon needs to be implemented.

The communication from InfoRoute to a user is also neglected some in the present version. Error messages for the most common errors when using InfoRoute have been implemented, but it is still possible to perform illegal tasks resulting in a somewhat cryptical message from the system. Not that the system shuts down, but

it is comforting to know what went wrong. Also as part of the communication is the online help system. The structure of this is implemented, but the files, indexes and actual help text is still missing.

Chapter 13

Conclusion

This thesis is focusing on solving the practical Dial-a-Ride Problem. To gain specific knowledge about the objectives and constraints in such a problem, a case at Jensens Turisttrafik is chosen because of its diversity, complexity, and relatively small size. The problem is described in detail with emphasis on both the special characteristics and the political aspects in the beginning of the thesis.

Having described the problem, the thesis aims to give an extensive survey of the litterature concerning the Dial-a-Ride Problem. This survey is coupled with a description of experiments and solutions found in Denmark over the years. The survey also provides the background for choosing the type of algorithm to be used.

Before chosing an algorithm to be used in solving this practical case of the Dial-a-Ride Problem, a shortest path algorithm is implemented and documented. This algorithm shows impressive results with regards to CPU-time when run on digitized road networks. The shortest path algorithm is implemented to perform on a special network design allowing for easy handling of practical restrictions on the road network. An important feature is the ability to reduce network size by adding one-way and no-right-turn restriction, thus actually decreasing the CPU-time needed to obtain a solution. Possible modifications and enhancements to the shortest path calculations are also considered in this thesis.

A mathematical formulation of the Dial-a-Ride Problem from existing litterature is presented. However to give structure to the discussion of additional issues in the practical Dial-a-Ride Prob-

lem, an extension to the standard formulation is developed and presented here. This extended formulation captures all aspects of the practical problem. However, because of the complexity of the Dial-a-Ride Problem, using the extended mathematical formulation to solve the problem to optimality is considered unimportant in this context.

Input to and output from the solution of the practical Dial-a-Ride problem is handled by a separate user interface called InfoRoute. InfoRoute is developed and designed to hold and display information in various practical vehicle routing problems. It is based on an extensive database allowing more exotic features such as capacity substitution along with more standard customer and request related data. InfoRoute can display customer, vehicle and routing data on a road map, and allows for independent address lookup, shortest path calculations, and map layout features. It is also possible to visualize real-time information in separate layers on the map. The implementation of separate layers on the map allows for quick and intuitive display of the data necessary in a given situation.

The algorithm McCluster for solving the practical Dial-a-Ride Problem is implemented as an independant module within the InfoRoute environment. InfoRoute sends and receives data to and from the McCluster module by communicating with the modules interface. This implementation ensures easy replacement or further development of the algorithm together with the possibility of having more algorithms running simultaniously. McCluster is based on an algorithm developed at CRT (University of Montréal) in the mid 1980's. McCluster is however modified to include practial aspects uncovered by the problem description and the extended mathematical formulation.

Since data is not readily available, a test dataset with a more simple structure the actual situation at Jensens Turisttrafik is generated. The dataset includes problems ranging from 25 to 300 dynamic and static requests. The tests serve mainly to show that the implemented algorithm is operational. However additional results indicate the importance of a good estimate of vehicle speed. The clustering-insertion technique is also very adept at finding "expensive" requests that are hard to serve because of time/distance

constraints related to customer inconvenience.

The overall result of this thesis is a software package able to solve the practical Dial-a-Ride Problem in a realistic and operationel environment. A few additional features will have to be implemented in order for the product to be used commercially, but the core of the developed software with the database, visualization, shortest path calculations, and automated features for solving the practical Dial-a-Ride problem is performing fast and reliable in a stable software environment.

Bibliography

[1] *The Vehicle Routing Problem.* SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2002.

[2] C. R. Alexandersen. Dial-a-ride - en analyse af ht's handicap service. Master's thesis, University of Copenhagen, 1994.

[3] Bjarke Andersen and Thomas Grejs. Route planning with graphical display of optimal routes (in danish: Ruteplanlægning med grafiske kørselsanvisninger). Master's thesis, The Technical University of Denmark, Dept. of Mathematical Modelling, 1997.

[4] J.-P. Belisle, F. Soumis, S. Roy, J. Desrosiers, Y. Dumas, and J.-M. Rousseau. The impact on vehicle routing of various operational rules of a transportation system for handicapped persons. Technical report, CRT - Université de Montréal, 1984.

[5] M. Ben-Akiva, J. Benjamin, G. J. Lauprete, and A. Plydoropoulou. Impact of advanced public transportation systems on travel by dial-a-ride. *Transportation Research Record*, (1557):72 – 79, 1996.

[6] M. Charikar and B. Raghavachari. The finite capacity dial-a-ride problem. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 458 – 467. IEEE Comput. Soc., 1998.

[7] L. L. Christensen and J. K. Jensen. Patientbefordring - et dynamisk dial-a-ride system. Master's thesis, Technical University of Denmark, 1991.

[8] C. F. Daganzo. Checkpoint dial-a-ride systems. *Transportation Research, Part B (Methodological)*, 18B(4 - 5):315 – 327, 1984.

[9] J. Desrosiers, Y. Dumas, and F. Soumis. A dynamic programming method for the large-scale single vehicle dial-a-ride problem with time windows. Technical report, CRT - Université de Montréal, 1984.

[10] J. Desrosiers, Y. Dumas, and F. Soumis. The multiple vehicles many to many routing problem with time windows. Technical report, CRT - Université de Montréal, 1984.

[11] J. Desrosiers, Y. Dumas, and F. Soumis. The multiple vehicle dial-a-ride problem. In J. R. Daduna and A. Wren, editors, *Computer-Aided Transit Scheduling. Proceedings of the Fourth International Workshop on Computer-Aided Scheduling of Public Transport.* Springer-Verlag, 1988.

[12] J. Desrosiers, Y. Dumas, F. Soumis, S. Taillefer, and D. Villeneuve. An algorithm for mini-clustering in handicapped transport. Technical report, GERAD, 1991.

[13] J. Desrosiers and F. Soumis. Centres de correspondance pour le transport des handicapes. Technical report, GERAD, 1982.

[14] R. B. Dial. Autonomous dial-a-ride transit introductory overview. *Transportation Research Part C: Emerging Technologies*, 3(5):261 – 275, 1995.

[15] Y. Dumas, J. Desrosiers, and F. Soumis. Large scale multi-vehicle dial-a-ride problems. Technical report, GERAD, 1989.

[16] M. F. Fels. Comparative energy costs of urban transportation systems. *Transportation Research*, 9(5):297 – 308, 1975.

[17] Færdselsstyrelsen. Year report 2000 - knowledge center for public transportation in rural areas (in danish), 2000.

[18] Giorgio Gallo and Stefano Pallotino. Shortest path methods: A unifying approach. *Mathematical Programming Study*, 26:38–64, 1986.

[19] Giorgio Gallo and Stefano Pallotino. Shortest path algorithms. *Annals of Operations Research*, 13:3–79, 1988.

[20] Fred Glover, Darwin D. Klingman, and Nancy V. Philips. A new polynomially bounded shortest path algorithm. *Operations Research*, 33(1):65–73, 1985.

[21] Fred Glover, Darwin D. Klingman, Nancy V. Philips, and Robert F. Schneider. New polynomial shortest path algorithms and their computational results. *Management Science*, 31(9):1106–1128, 1985.

[22] P. Healy and R. Moll. A new extension of local search applied to the dial-a-ride problem. *European Journal of Operational Research*, 83(1):83 – 104, 1995.

[23] Richard V. Helgason, Jeffery L. Kennington, and B. Douglas Stewart. The one-to-one shortest-path problem: An empirical analysis with the two-tree dijkstra algorithm. *Computational Optimization and Applications*, 2(1):47–75, 1993.

[24] A. G. Hobeika. Simulation of dial-a-ride bus system in the greater lafayette area; indiana. In W. G. Vogt and M. H. Mickle, editors, *Modeling and Simulation vol.5*. ISA, 1974.

[25] Ming Hung and James Divoky. A computational study of efficient shortest path algorithms. *Computers and Operations Research*, 15(6):567–576, 1988.

[26] I. Ioachim, J. Desrosiers, Y. Dumas, and M. M. Solomon. A request clustering algorithm in door-to-door transportation. Technical report, GERAD, 1991.

[27] I. Ioachim, J. Desrosiers, Y. Dumas, M. M. Solomon, and D. Villeneuve. A request clustering algorithm for door-to-door handicapped transportation. *Transportation Science*, 29(1):63 – 78, 1995.

[28] J. Jang-Jei, A. R. Odoni, H. N. Psaraftis, and N. H. M. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research, Part B (Methodological)*, 20B(3):243 – 257, 1986.

[29] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the Association for Computing Machinery*, 24(1):1–13, 1977.

[30] J. W. Baugh Jr., D. K. R. Kakivaya, and J. R. Stone. Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. *Engineering Optimization*, 30(2):91 – 124, 1998.

[31] M. Kubo and H. Kasugai. Heuristic algorithms for the single vehicle dial-a-ride problem. *Journal of the Operations Research Society of Japan*, 33(4):354 – 365, 1990.

[32] O. B. G. Madsen, H. F. Ravn, and J. M. Rygaard. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60:193 – 208, 1995.

[33] M. K. Mikkelsen. Interaktiv dial-a-ride og matematisk optimering. Master's thesis, Technical University of Denmark, 1994.

[34] Jean-Francois Mondou, Teodor G. Crainic, and Sang Nguyen. Shortest path algorithms: A computational study with the c programming language. *Computers and Operations Research*, 18(8):767–786, 1991.

[35] J. Y. Potvin and J. M. Rousseau. Constraint-directed search for the advanced request dial-a-ride problem with service quality constraints. In O. Balci, R. Sharda, and S. A. Zenios, editors, *Computer Science and Operations Research. New Developments in their interfaces.* Pergamon, 1992.

[36] H. Psaraftis. A dynamic programming solution to the single-vehicle, many-to-many, immidiate request dial-a-ride problem. *Transportation Science*, 14:130 – 154, 1980.

[37] H. N. Psaraftis. Analysis of an o(n/sup 2/) heuristic for the single vehicle many-to-many euclidean dial-a-ride problem. *Transportation Research, Part B (Methodological)*, 17B(2):133 – 145, 1983.

[38] H. N. Psaraftis. An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science*, 17(3):351 – 357, 1983.

[39] H. N. Psaraftis. Scheduling large-scale advance-request dial-a-ride systems. *American Journal of Mathematical and Management Sciences*, 6(3-4):327 – 367, 1986.

[40] D. Roos. An innovative computer based urban transportation system. In A. Lew, editor, *Proceedings of the 5th Hawaii international conference on system science.* Western Periodicals, 1972.

[41] S. Roy, L. Chapleau, J. Ferland, G. Lapalme, and J.-M. Rousseau. The construction of routes and schedules for the transportation of the handicapped. Technical report, CRT - Université de Montréal, 1983.

[42] S. Roy, J.-M. Rousseau, G. Lapalme, and J. Ferland. Routing and scheduling for the transportation of the disabled persons: Test problems and evaluation of results. Technical report, CRT - Université de Montréal, 1985.

[43] S. Roy, J.-M. Rousseau, G. Lapalme, and J. Ferland. Routing and scheduling for the transportation of the disabled persons: The algorithm. Technical report, CRT - Université de Montréal, 1985.

[44] S. Roy, J.-M. Rousseau, G. Lapalme, and J. Ferland. Routing and scheduling for the transportation of the disabled persons: The analyst's manual and computer programs. Technical report, CRT - Université de Montréal, 1985.

[45] K. S. Ruland and E. Y. Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & Mathematics with Applications*, 33(12):1 – 13, 1997.

[46] R. Slevin and A. E. Cooper. Minibus and dial-a-ride: Initial experiences with the abingdon experiment. *Traffic Engineering & Control*, 14(12):586 – 589, 1973.

[47] M. M. Solomon and J. Desrosiers. Time window constrained routing and scheduling problems. *Transportation Science*, 22(1):1 – 13, 1988.

[48] D. M. Stein. Scheduling dial-a-ride transportation systems. *Transportation Science*, 12(3):232 – 249, 1978.

[49] L. Suen, A. Ebrahim, and M. Oksenhendler. Computerised dispatching for shared-ride taxi operations in canada. *Transport Planning and Technology*, 7(1):33 – 48, 1981.

[50] D. Teodorovic. Fuzzy logic systems for transportation engineering: The state of the art. *Transportation Research Part A: Policy and Practice*, 33(5):337 – 364, 1999.

[51] P. Toth and D. Vigo. Heuristic algorithms for the handicapped persons transportation problem. *Transportation Science*, 31(1):60 – 71, 1997.

[52] Trafikministeriet. Public transportation in rural areas (in danish). Technical Report 95757 1999, Trafikministeriet, 1999.

[53] J. B. Williamson. The on-line vehicle scheduler. In *International Conference on Information-Decision-Action Systems in Complex Organisations (Conf. Publ. No. 353)*, pages 49 – 53. IEE, 1992.

[54] N. H. M. Wilson. Routing and scheduling decisions in demand responsive transportation systems. In *Proceedings of the 1972 International Conference on Cybernetics and Society*, pages 484 – 488. IEEE, 1972.

[55] N. H. M. Wilson. Second generation computer control procedures for dial-a-ride. In *Proceedings of the 1975 IEEE Conference on Decision Control including the 14th Symposium on Adaptive Processes*, pages 547 – 552. IEEE, 1975.

[56] N. H. M. Wilson and C. Hendrickson. Performance models of flexibly routed transportation services. *Transportation Research, Part B (Methodological*, 14B(1-2):67 – 78, 1980.

[57] N. H. M. Wilson, J. Sussman, H. Wang, and B. Higonnet. Scheduling algorithms for dial-a-ride systems. Technical Report USL TR-70-13, MIT, 1971.

[58] F. Benjamin Zhan and Charles E. Noon. Shortest path algorithms: An evaluation using real road networks. *Transportation Science*, 32(1):65–73, 1998.

[59] M. J. Zobrak. Mini computer system for dial-a-ride. In *5th annual 1971 IEEE international Computer Society conference on hardware, software, firmware and trade-offs (digests)*, pages 41 – 42. IEEE, 1971.