National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Post-Optimization Procedures For The Dial-A-Ride Problem In Paratransit

b y
Huashi Wang

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Computer Science

School of Computer Science
Carleton University
Ottawa, Ontario
Canada
July, 1993

©1993, Huashi Wang

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file *Votre référence*

Our file *Notre référence*

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-89815-1

Canada

Name HUASHI WANG

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Computer Science
SUBJECT TERM

0984 U·M·I
SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
General 0289
Ancient 0290
Linguistics 0291
Modern 0401
Literature 0294
Classical 0295
Comparative 0297
Medieval 0298
Modern 0316
African 0591
Asian 031
Canadian (English) 0352
Canadian (French) 0355
English 0593
Germanic 0311
Latin American 0312
Middle Eastern 0315
Romance 0313
Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology 0323

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical 0310
Business Administration 0272
General 0770
Accounting 0454
Banking 0338
Management 0385
Canadian Studies 0501
Economics 0503
General 0505
Agricultural 0508
Commerce-Business 0509
Finance 0510
History 0511
Labor 0358
Theory 0366
Folklore 0351
Geography 0578
Gerontology
History
General

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science 0615
General 0616
International Law and Relations 0617
Public Administration 0814
Recreation 0452
Social Work 0626
Sociology 0627
Criminology and Penology 0938
Demography 0631
Ethnic and Racial Studies 0628
Individual and Family Studies 0629
Industrial and Labor Relations 0630
Public and Social Welfare 0700
Social Structure and Development 0344
Theory and Methods 0709
Transportation 0999
Urban and Regional Planning 0453
Women's Studies

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy
Animal Culture and Nutrition 0475
Animal Pathology 0476
Food Science and Technology 0359
Forestry and Wildlife 0478
Plant Culture 0479
Plant Pathology 0480
Plant Physiology 0817
Range Management 0777
Wood Technology 0746
Biology 0306
General 0287
Anatomy 0308
Biostatistics 0309
Botany 0379
Cell 0529
Ecology 0353
Entomology 0369
Genetics 0793
Limnology 0410
Microbiology 0307
Molecular 0317
Neuroscience 0416
Oceanography 0433
Physiology 0821
Radiation 0778
Veterinary Science 0472
Zoology
Biophysics 0786
General 0760
Medical

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Polynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences 0566
General 0300
Audiology 0992
Chemotherapy 0567
Dentistry 0350
Education 0769
Hospital Management 0758
Human Development 0982
Immunology 0564
Medicine and Surgery 0347
Mental Health 0569
Nursing 0570
Nutrition 0380
Obstetrics and Gynecology
Occupational Health and Therapy 0354
Ophthalmology 0381
Pathology 0571
Pharmacology 0419
Pharmacy 0572
Physical Therapy 0382
Public Health 0573
Radiology 0574
Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences
Chemistry 0485
General 0749
Agricultural 0486
Analytical 0487
Biochemistry 0488
Inorganic 0738
Nuclear 0490
Organic 0491
Pharmaceutical 0494
Physical 0495
Polymer 0754
Radiation 0405
Mathematics 0605
Physics 0986
General 0606
Acoustics 0608
Astronomy and Astrophysics 0748
Atmospheric Science 0607
Atomic 0798
Electronics and Electricity 0759
Elementary Particles and High Energy 0609
Fluid and Plasma 0610
Molecular 0752
Nuclear 0756
Optics 0611
Radiation 0463
Solid State
Statistics
Applied Sciences
Applied Mechanics 0346
Computer Science 0984

Engineering 0537
General 0538
Aerospace 0539
Agricultural 0540
Automotive 0541
Biomedical 0542
Chemical 0543
Civil 0544
Electronics and Electrical 0348
Heat and Thermodynamics 0545
Hydraulic 0546
Industrial 0547
Marine 0794
Materials Science 0548
Mechanical 0743
Metallurgy 0551
Mining 0552
Nuclear 0549
Packaging 0765
Petroleum 0554
Sanitary and Municipal 0790
System Science 0428
Geotechnology 0796
Operations Research 0795
Plastics Technology 0994
Textile Technology

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

**Post-Optimization Procedures For
The Dial-A-Ride Problem In Paratransit**

Submitted by Huashi Wang, B.Eng.,
in partial fulfillment of the requirements for the degree of
Master of Computer Science

J. Fiala

Thesis Supervisor

J. R. P.

Director, School of Computer Science

Carleton University
July 1993

Abstract

The Dial-A-Ride Problem (DARP) is one of routing vehicles of given capacities from a depot to service customers, each wishing to go from an origin to a destination, while minimizing the total distance travelled. The design problem for DARP in paratransit is concerned with service time windows (DARPTW). Customer precedence, vehicle capacity and other related paratransit constraints must be also satisfied. Since the problem is NP-hard, we are mostly interested in heuristic algorithms. Our post-optimization procedures consist of intra-route and inter-route methods. For intra-route post-optimization, we use 2-opt, 3-opt, Or-opt backward and forward interchange schemes to improve the initial routes, which are set up by using insertion heuristics, according to different objective functions. For inter-route post-optimization, we develop and present a new algorithm based on the insertion heuristics. Finally, a comparison of the computational results obtained from six test cases is made among the different intra-route post-optimization procedures and some conclusions are drawn.

Acknowledgments

First I would like to gratefully thank my supervisor Professor Frank Fiala for his supervision of this thesis. His kindness and constant academic support made my graduate studies both an enriching and enjoyable experience.

I would also like to thank Marie Fiala Timlin for her thoughtful criticisms and suggestions which have been very valuable.

Very special thanks go to my wife, Yun, my mother, Yaxiu, and my parents-in-law, Vanya and Madam Chen, for their enthusiasm, support and encouragement. Finally I much appreciated the assistance offered by my fellow graduate students during the course of my studies.

Contents

1	Introduction	1
1.1	Overview of Vehicle Routing and Scheduling Problems.	1
1.2	Dial-A-Ride Problem.	13
1.3	The Related Paratransit Environment	17
1.4	Focus of the Thesis.	20
1.5	Outline of the Thesis.	23
2	Route Initialization	24
2.1	The DARPTW Model.	24
2.1.1	Request.	25
2.1.2	Driver Shift	25
2.1.3	The Distance Table.	26
2.1.4	The Speed Table.	28
2.1.5	Handling Time	30
2.1.6	The Quality of Service	30
2.1.7	Setting Up Time Window.	32
2.1.8	Clustering the Requests.	35
2.1.9	The Measure Matrix	35
2.1.10	The Horizon Measure	42
2.2	The Insertion Heuristic Algorithm	43
2.2.1	Route Initialization.	43
2.2.2	Selection of the Best Route for a Given Request	43
2.2.3	Insertion of a Request into a Route.	44
2.2.4	The Heuristic Algorithm.	44
2.3	Measures of Quality.	45

3	Post-Optimization Heuristics	46
3.1	Motivation.	46
3.2	Intra-route Optimization Heuristic	48
3.2.1	Or-opt Backward Search.	54
3.2.2	Or-opt Forward Search.	63
3.3	Inter-route Optimization Heuristics	68
4	Computational Results	77
4.1	Test Problem Description.	77
4.2	Test Results.	79
4.2.1	Initial Routes.	79
4.2.2	2-opt and 3-opt Solutions	81
4.2.3	Or-opt Solutions	81
4.3	Conclusions	85
5	Summary and Conclusions	88
5.1	Summary of the Thesis	88
5.2	Directions of Future Work	90
	References	92

List of Tables

1.1 Three main operating scenarios in paratransit system	18
4.1 Initial routes	80
4.2 Initial routes (continued)	80
4.3 Results of improvement heuristics (Case1).	82
4.4 Results of improvement heuristics (Case2)	82
4.5 Results of improvement heuristics (Case3)	83
4.6 Results of improvement heuristics (Case4)	83
4.7 Results of improvement heuristics (Case5)	84
4.8 Results of improvement heuristics (Case6)	84

List of Figures

1.1	Clarke and Wright savings heuristic	6
1.2	Gillett's sweep heuristic (cluster-first route-second)	9
1.3	Haimovich's heuristic (route-first cluster-second)	9
1.4	A 2-interchange.	11
1.5	An Or-interchange	12
1.6	A practical structure of general paratransit system	20
2.1	A single depot DARPTW.	26
2.2	Two ways to calculate distance $D(p,d)$	27
2.3	The defined customized zones.	28
2.4	A case with same distance.	29
2.5	Speed function	29
2.6	Time window (departure time desired).	33
2.7	Time window (arrival time desired).	33
2.8	Two examples of time windows	34
2.9	Insertion of pickup stop(PICK) to the given ROUTE(r).	37
2.10	Insertion of a REQUEST(i) to the given ROUTE(r).	39
3.1	Intra-route optimization	47
3.2	Inter-route optimization	48
3.3	All possible 3-interchanges	50
3.4	Backward relocation for Or-interchange	52
3.5	Forward relocation for Or-interchange	52
3.6a	Vehicle with only ambulatory handicapped people	60
3.6b	Vehicle with only wheelchair handicapped people	60
3.6c	Vehicle with both types of handicapped people	60
3.7	Method to calculate "how close" the pair of requests is	72
3.8	Insertion heuristics for swapping REQUEST(g) with REQUEST(c).	73
4.1	The distribution of the given stops	79

Chapter 1

Introduction

1.1 Overview of Vehicle Routing and Scheduling Problems

Vehicle routing and scheduling is among the major activities related to the distribution of goods or services from the point of production (pickup) to the point of delivery (dropoff) for both the private and public sectors. The goal is to provide the best service to the customer at minimum cost to the producer. As these two objectives are often contradictory, namely that better service is more costly, the effective and efficient management of the distribution is becoming more and more important. Often the use of mathematical programming and combinatorial optimization is needed to analyze and solve such problems to permit the realization of cost reductions or profit improvement.

Usually, vehicle routing and scheduling problems can be divided into three categories: vehicle routing, vehicle scheduling and combined vehicle routing and scheduling problem [Bodin 83]. The *vehicle routing problem* (VRP) considers the spatial aspect of routing which involves the routing of (capacitated) vehicles through a collection of points to pick up or deliver goods originating and terminating at a central depot. The *vehicle scheduling problem* (VSP) considers the temporal aspect of scheduling which involves the scheduling of vehicles to meet time constraints imposed upon their routes. The problem that is the focus of this thesis falls into the third category of *combined vehicle routing and scheduling problem* (VRSP) [Savelsbergh 85] where the spatial aspect of routing is blended with the temporal aspect of scheduling, because time, precedence and capacity constraints must be satisfied.

A suitable classification scheme for VRSPs is not easy to find because of the wide variety of routing and scheduling problems. In most cases, the classification can be made

based on the final output of the system, that is, a route and a schedule for each vehicle or each driver where a route specifies the sequence of stops to be visited, and the schedule specifies the time at which an activity should be carried out at each stop. Also the difficulty is whether the constraints of the problem or the solution technique should be used as the basis for classification. For example, suppose a problem is classified as a *vehicle routing problem with time windows* (VRPTW), i.e., the customers have fixed time intervals during which they must be serviced. For certain distributions of the time windows, it may be feasible to solve the problem by an algorithm that seeks good spatial configurations for the routes. In fact, the use of a pure routing algorithm reflects a decision for reclassifying the problem based on the flexibility of the time constraints. Consequently the vehicle routing and scheduling problems are classified by their characteristics as follows: ([Golden 88] and [Bodin 83])

*	Underlying Network:	directed / undirected / mixed / Euclidean
*	Nature of Demand:	pickups only / dropoffs only / mixed pickups and dropoffs
*	Vehicle Capacity:	imposed and identical / imposed and different / not imposed
*	Demands Location:	nodes / arcs / mixed
*	Demands Type:	known in advance / uncertain / probabilistic / real-time inflow / partial satisfaction
*	Size of Vehicle Fleet:	single / multiple
*	Type of Vehicle Fleet:	homogeneous / heterogeneous / special types
*	Number of Depots:	single / multiple
*	Time Window:	soft / hard
*	Precedence Constraint:	yes / no
*	Route Times:	travel time / excess travel time / handling time / waiting time
*	Type of Passengers:	ordinary / handicapped (ambulatory or in wheelchair) / accompanying people
*	Type of Vehicles:	Van / Car / Minibus / Orion

-
*	Data Requirements:	road networks / customer addresses and locations / driver shifts / vehicle location information / vehicle speed / customer credit and billing information
*	Costs:	vehicle / routing and scheduling / fixed operating or vehicle acquisition / common carrier for unserved demands
*	Objectives:	minimize the actual travel time
minimize the total completion time
minimize the partial completion time & actual total travel time
minimize the loaded vehicle waiting time
minimize the excess travel time
minimize the number of vehicles required
maximize service or convenience
maximize the vehicle productivity rate (VPR)

Due to the intrinsic complexity of vehicle routing and scheduling problems, a great number of algorithms and methods have been developed. In almost all cases, the solutions are heuristic ones because no efficient exact algorithms have been developed for the sizes required for practical problems [Lapalme 92]. In fact, almost all problems of vehicle routing and scheduling are *NP-hard* [Lenstra 81]. Obtaining an optimal solution of a VRSP, like that of the well-known *travelling salesman problem* (TSP), is widely considered to be unsolvable by exact, polynomial time bounded algorithms. One technique which has been adopted by many algorithm designers, is to abolish the ideal of optimization and settle for an approximation of the optimal solution. As the VRSP researchers recognized early that important improvements could be made by non-optimal, directed investigation into the mathematical and computational structures that describe VRSPs, progress is marked by the design and analysis of heuristic algorithms that produce feasible, hopefully near-optimal results. These methods quickly provide a good solution which often yields immediate distribution cost savings.

There exists a large number of algorithms and practical applications in the area of vehicle routing and scheduling problems, such as *Vehicle Routing with Backhauls*, *Vehicle Routing with Site Dependencies*, *Deadline Vehicle Routing Problem*, *Location Routing Problems*, *Dynamic Vehicle Routing Problems*, *The Probabilistic Vehicle Routing Problem*, *Routing Planning for Coast Guard Ships*, *Scheduling in the Post Carriers for the US Postal Service*, and *Chinese Postman Problem*, etc. Traditionally, most of the existing algorithms had been designed to solve pure routing problems and hence only deal with the spatial aspect. So these algorithms cannot handle all kinds of constraints that frequently occur in practice. One practical constraint is the specification of time windows at customers, i.e., time intervals during which they must be serviced. As time windows arise naturally in problems faced by business organizations that work on fixed schedules, these lead to combined *routing and scheduling problems with time windows* (VRSPTW) and ask for algorithms that also take temporal aspects into account. Now more and more practical applications fall into this class. Specific examples include *bank deliveries*, *postal deliveries*, *industrial refuse collection*, *dial-a-ride service*, and *school bus routing and scheduling*. So far, this type of constraint has been handled mostly by manual adjustments to routing-based schedules.

Problems with time windows are quite difficult from a computational complexity perspective. Since the VRSP is **NP-hard**, by restriction, the VRSPTW is **NP-hard** (by reduction from the VRSP). Furthermore, [Savelsbergh 85] has shown that even finding a feasible solution to the VRSPTW when the number of vehicles is fixed is itself an **NP-hard** problem. Therefore, the development of heuristic algorithms for this class of problems is of primary interest.

[Solomon 88a] reviews the research relating to the study of special problem structures with time windows as follows: (obviously, all the problems are **NP-hard**)

- 1) Pickup and Delivery Problem (PDPTW)
- 2) The Travelling Salesman Problem (TSPTW)

- 3) The Multiple Traveling Salesman Problem (m -TSPTW)
- 4) The Vehicle Routing and Scheduling Problem (VRSPTW)
- 5) Dial-A-Ride Problem (DARPTW)
- 6) The Multi-Period Vehicle Routing Problem (MPVRPTW)
- 7) The Shoreline Problem (SHPTW)

We are mostly interested in heuristic algorithms. The optimization algorithms are usually presented by using branch and bound, dynamic programming and set partitioning methods. Approximation algorithms are based on construction, iterative improvement and incomplete optimization. These heuristic algorithms can be broadly classified into several types according to [Gendreau 91], [Solomon 87], and [Golden 88].

• Constructive Heuristics (Tour-building Algorithms)

This category can be divided into *sequential* and *parallel* methods. Sequential procedures construct one route at a time until all customers are scheduled. Parallel procedures are characterized by the construction of a number of routes simultaneously. The routes are either allowed to form freely or their number is fixed a priori. Several heuristics belong to this category:

1) Savings Heuristics

Clarke and Wright [Clarke 64] proposed a savings algorithm which is one of the earliest heuristics and is without doubt the most widely known one for the VRP. Starting with n distinct routes in which each customer is serviced by a dedicated vehicle, the parallel version is characterized by the addition, at every iteration, of a link of distinct, partially formed routes between two end customers, according to a saving criterion:

$$sav_{ij} = d_{i0} + d_{0j} - \mu d_{ij}, \mu \geq 0$$

where d_{i0} is the distance from customer i to depot 0, and sav_{ij} is the savings in distance that results from servicing customers i and j , when $\mu=1$, on one route as opposed to servicing them individually, directly from the depot (see Figure 1.1). The main drawbacks

Figure 1.1: Clarke and Wright savings heuristic

of this algorithm are that it can be quite time consuming (it runs in $O(n^2 \log n)$ time) and it has a tendency to produce circumferential routes. Computational complexity can, however, be reduced through the use of appropriate data structures. In addition, a number of authors have proposed algorithmic improvements and variants to the savings method in order to arrive at more acceptable solutions ([Yellow 70] and [Paessens 88]). Others, such as [Mole 76], have used constructive methods that favour radial shaped routes. Recently more sophisticated algorithms that consider the savings associated with all possible merges of two routes, by solving a weighted matching problem, have been developed ([Desrochers 89] and [Alinkemer 85]).

2) Time-Oriented, Nearest-Neighbor Heuristics

These algorithms belong to the class of sequential, tour-building algorithms. Initially, a route consists of the depot only. At each iteration, an unvisited customer who is the closest to the current end point of the route is selected and added to the route to become its new end point. The selection is restricted to those customers whose addition is feasible with respect to capacity and time window constraints. A new route is started any time the search fails, unless there are no more customers to schedule. Suppose the last customer on the current partial route is customer i , and let j denote any unrouted customer that could be visited next. For the following description, we need to introduce some notation. The service at a customer, say i , $i=1, \dots, n$, involving pickup and/or delivery of goods or other

services for s_i units of time, can begin at time b_i , within a time window defined by the earliest time e_i and the latest time l_i that customer i will permit the start of service. We use c_{ij} , defined below, to account for both geographical and temporal closeness of customers. Let t_{ij} be the direct travel time between i and j , d_{ij} be the direct distance from i to j , T_{ij} be the time difference between the completion of service at i and the beginning of service at j , $b_j = \max(e_j, b_i + s_i + t_{ij})$ be the time for j at which service can begin, and v_{ij} be the urgency of delivery to customer j . Then we can get:

$$T_{ij} = b_j - (b_i + s_i)$$

$$v_{ij} = l_j - (b_i + s_i + t_{ij})$$

The measurement of closeness can include spatial as well as temporal aspects as follows: ([Solomon 87])

$$c_{ij} = \delta_1 d_{ij} + \delta_2 T_{ij} + \delta_3 v_{ij}, \text{ with } \delta_1 + \delta_2 + \delta_3 = 1$$

In this way, we can choose the minimum value of c_{ij} ($j \neq i, j = 1, \dots, n$) to get the closest customer j .

3) Insertion Heuristics

These algorithms belong to the class of sequential tour-building heuristics. After initializing the current route, the method uses two criteria, $c_1(i, u, j)$ and $c_2(i, u, j)$ (they can be chosen based on the actual conditions/cases), at every iteration to insert a new customer u into the current partial route, between two adjacent customers i and j on the route. Let $(i_0, i_1, i_2, \dots, i_m)$ be the current route, with $i_0 = i_m = 0$ (depot). For each unrouted customer u , we first compute the best feasible insertion place in the emerging route as

$$c_1(l(u), u, j(u)) = \min [c_1(i_{p-1}, u, i_p)], \quad p=1, \dots, m$$

Obviously, inserting u between i_{p-1} and i_p could potentially alter all the service times at customers $(i_p, \dots, i_m)$. So feasibility constraints must be satisfied (we will discuss this

later in detail). Next, the best customer u to be inserted in the route is selected as the one satisfying

$$c_2(i(u^*), u^*, j(u^*)) = \max \text{ or } \min [c_2(i(u), u, j(u))],$$

u is unrouted and feasible customer

Customer u^* is then inserted into the route between $i(u^*)$ and $j(u^*)$. When no customer with feasible insertions can be found, the method starts a new route, unless it has already routed all customers.

• Two-phase Algorithms

These algorithms can be subdivided into three main classes. In *cluster-first route-second* method, customers are first assigned to vehicles and each route is then optimized separately by using any of the known Travelling Salesman Problem (TSP) algorithms. [Gillett 74] proposed a sweep heuristic: Starting from an arbitrary customer, the algorithm "sweeps" the plane by rotating a ray with the end point at the depot, until a customer is met where vehicle capacity becomes exceeded. The process is then repeated starting from the last customer until all customers have been swept (see Figure 1.2). This initial assignment is then updated and routes are reoptimized. Due to the time window constraints, some customers in a cluster could remain unscheduled. To preserve geographical cohesiveness, one might consider different seed selection criteria for the next cluster. [Solomon 87] uses a simple rule that bisects the sector just considered and, assuming a counterclockwise sweep in the counterclockwise half-sector, lets the seed for a new cluster be the unscheduled customer with the smallest angle formed by the ray from the depot through that customer and the bisector. The intuition for partitioning the unscheduled customers in the sector into two subsets is that the customers in the clockwise half-sector will be relatively far away from the new cluster. By inserting these customers at a later stage, a better schedule may be generated. This process will be repeated until all customers have been scheduled.

Figure 1.2: Gillett's sweep heuristic (cluster-first route-second)

The *route-first cluster-second* methods are described from the work of [Haimovich 85]. Typically, these algorithms first construct a near optimal TSP tour and then break it into several feasible VRP routes (see Figure 1.3). Finally, a number of *iterative* procedures have been proposed. These algorithms start from a feasible solution and seek to improve it through a sequence of local modifications, such as [Fisher 81], which use

Figure 1.3: Haimovich's heuristic (route-first cluster-second)

integer linear programming to execute the two phases, coupled with Bender's decomposition technique to iterate between the solution of a generalized assignment problem and that of a TSP. [Noon 91] uses a Lagrangean relaxation method in which

customers are assigned "reward values" and the constraint that each customer is visited exactly once by a vehicle is relaxed. Rewards are updated so as to move closer to a feasible solution. The process does not, however, always converge towards a feasible solution.

- **Incomplete Optimization Algorithms**

These methods use a combination of enumeration of the solution space and heuristic rules to truncate the search. Two principal ideas are the heuristic generation of columns and the partial exploration of the branch and bound tree. For heuristic generation of columns, while solving the relaxed master problem, we will eliminate vertices, arcs and states in a heuristic fashion. The elimination rules are not relaxed, so that an approximate solution to the linear program is obtained. A partial exploration of the search tree can take place in several ways. One is to obtain an integral solution by depth-first search and then to explore the tree for the remaining available time. Another way is to use an invalid branching rule, i.e., to eliminate branches on heuristic grounds. A combination of these ideas has been used to obtain feasible integral solutions within two percent from the optimum with highly reduced running times. For this, any of the known enumerative algorithms can be used (see [Laporte 84]).

- **Improvement Methods**

The most powerful improvement methods are *k-opt* (proposed by [Lin 73]), *Or-opt* (proposed by [Or 76]) and *Tabu search* (proposed by [Glover 77]).

1) **k-opt and Or-opt**

k-opt and *Or-opt* interchange schemes can be used as post-optimizers on any individual route produced by a VRSP algorithm, or they can be modified so that they can be applied to a global VRSP solution, thus allowing for reassignments of customers to different routes. Briefly, a *k-interchange* is a substitution of *k* arcs of a route with *k* other arcs. A route is *k-opt* if additional *k*-interchanges will fail to produce shorter routes. For example, the 2-opt algorithm searches for the best 2-interchange possible by examining all

pairs of nonadjacent links. A pair of links can be replaced uniquely by two other links that still leave the route connected but reverse the direction of the middle route segment (see Figure 1.4, where we replace $[i, i+1], [j, j+1]$ by $[i, j], [i+1, j+1]$ and reverse the direction of $[i+1, \dots, j]$ to $[j, \dots, i+1]$). In the TSP, the processing of a single k -interchange takes constant time for any fixed value of k . One only has to test whether the exchange is profitable and does not have to bother about other feasibility constraints. In the case of TSPTW, the processing of a k -interchange may take $O(n)$ time. This is because a modification at one point may affect the departure times on the entire route, so that feasibility questions arise. Obviously, the total number of possible 2-interchanges equals

Figure 1.4: A 2-interchange

the number of subsets of two links that can be formed from the set of N links that make up the tour. This number is equal to $\binom{N}{2}$, which implies a time complexity of $O(N^2)$.

Therefore, when we consider the feasibility constraint, it will need $O(N^3)$ to verify 2-opt. Similarly, we can get that $O(N^4)$ time complexity is needed for the verification of 3-opt.

Because the computational requirement to verify 3-opt becomes prohibitive if the number of vertices increases, proposals have been made to take only a subset of all possible 3-interchanges into account. [Or 76] proposes to restrict attention to those 3-interchanges in which a string of one, two or three consecutive vertices (a path) is relocated between two others. An Or-interchange is depicted in Figure 1.5. The path $[j, \dots, i]$ is

relocated between k and $k+1$. Note that no paths are reversed in this case and there are only $O(N^2)$ exchanges of this kind.

The advantage of using Or-opt improvement scheme is that constant time suffices for the feasibility check. [Savelsbergh 88] has applied the Or-opt technique to the VRSPTW. In this thesis, we will use insertion heuristics to set up the initial routes, then use an Or-opt scheme to develop a post-optimization procedure for the *dial-a-ride problem with time windows* (DARPTW) in paratransit. Detailed discussions can be found later.

Figure 1.5: An Or-interchange

2) Tabu Search

Tabu search is a new improvement method ([Gendreau 91]). Here, successive “neighbours” of a solution are examined and the best is selected. In order to avoid local optima, the objective is allowed to deteriorate, and to prevent cycling, solutions that were recently examined are forbidden and inserted in a constantly updated tabu list. There are three VRSP algorithms based on this approach. The first one is in [Willard 89] where the problem is first transformed into a TSP by replication of the depot, and the search is restricted to neighbour solutions that can be reached by means of 2-opt and 3-opt interchanges, while satisfying the VRSP constraints (no time windows). In the second approach [Pureza 91], the search proceeds from one solution to the next by swapping customers between two routes. The third tabu algorithm [Semet 91] was proposed for the solution of a real-life VRSP containing several features. The basic tabu move in [Gendreau

91] consists of moving a city from its current route into an alternative route. In all these cases, a feasible solution is never allowed to become infeasible with respect to the feasibility constraints.

1.2 Dial-A-Ride Problem

A generalization of VRSPTW is the *pickup and delivery problem with time windows* (PDPTW). The problem is to construct optimal routes satisfying transportation requests, which require both pickup and delivery under precedence, capacity and time window constraints. The VRSPTW is a particular case of PDPTW where all the pickups or destinations are the common depot.

A particular instance of the PDPTW in transportation is the *dial-a-ride problem* (DARP) which deals with precedence constraints existing in transportation systems. Basically, the problem is one of routing vehicles of given capacities from a depot to service customers, each wishing to go from a distinct origin to a distinct destination, while minimizing the total distance travelled. There are several versions of the problem. In the dynamic or real time dial-a-ride problem, the customers need immediate service (*demand responsive*), so routing and scheduling is done in real time. In the subscriber or static dial-a-ride problem, the customers call before they need service (*advanced reservation*) so that all the demands are known before the routing and scheduling is done. The precedence relationships must always hold in both cases since the customer has to be picked up before being dropped off. Another version of DARP is single or multiple dial-a-ride problem. There may also be time windows since either the pickup time or delivery time is specified, and the corresponding delivery or pickup, respectively, must be carried out within a certain interval of that specified time. In these situations, the temporal constraints imposed by the customers strongly restrict the total vehicle load at any point of time, and the capacity constraints are of secondary importance. The cost of a route is a combination of travel time and customer dissatisfaction.

The DARP was first examined by [Wilson 71] in connection with the development of real-time algorithms and many concepts such as sequential insertion of customers and form of the objective function are derived from that work. Although single-vehicle dial-a-ride systems do not exist in practice, single-vehicle dial-a-ride algorithms can be used as subroutines in a large scale multi-vehicle dial-a-ride environment.

[Psaraftis 80] developed an exact algorithm, a backward dynamic programming method, for solving the single vehicle many-to-many immediate-request dial-a-ride problem. A generalized objective function was examined, consisting of a weighted combination of the time to service all customers and the total degree of dissatisfaction experienced by customers while waiting for service. This dissatisfaction was assumed to be a linear function of each customer's waiting and riding time. But rather than imposing time window constraints at the origins and destinations, he imposed a maximum position shift with the respect to the ordering of pickup and delivery times requested. In addition, vehicle capacity constraints and special priority rules were included, and both "static" and "dynamic" versions of the problem were examined. The computational complexity for N customers is $O(N^2 3^N)$. [Psaraftis 83a] developed a forward dynamic programming method to solve the problem where each customer has specified upper and lower bounds for his pickup and delivery times and where the objective function is to minimize the time needed to service all customers. Unfortunately, this algorithm was independent of the tightness of the time window constraints, and in practice it cannot solve problems with more than about eight to ten customers. [Psaraftis 83b] also analyzed an $O(N^2)$ heuristic algorithm for solving the uncapacitated single-vehicle dial-a-ride problem by constructing minimum spanning trees.

[Sexton 85a,85b] gave a mixed integer nonlinear programming formulation for the single vehicle dial-a-ride problem with given maximum delivery times or specified minimum pickup times to minimize customer inconvenience. The result is a heuristic routing and scheduling algorithm, based on Benders' decomposition, which is known to

produce high quality solutions. Its computational tractability stems from the noniterative nature of the scheduling algorithm. Successful computational experience on moderately sized real data is reported. Finally, [Desrosiers 86] proposed an optimal solution to the single-vehicle problem with clock time windows by using a forward dynamic programming approach that significantly reduces the number of states generated. The objective function of minimizing the total distance travelled, while respecting vehicle capacity and time windows, is less general than others proposed for minimizing user inconvenience. However, it is possible to take user inconvenience issues into account through the definition of the time windows. Furthermore, two labels (time, cost) are being used for solving the shortest path problem with time windows by dynamic programming. The development of criteria for the elimination of infeasible states results in solution times which increase linearly with problem size.

For the multi-vehicle DARPTW version, we present several studies here. The only known exact algorithm for the multiple vehicle routing problem with both pickup and delivery in the presence of time window constraints has been developed by [Dumas 85]. This column generation algorithm has been applied to small problems in regards to the transportation of goods with tight vehicle capacity constraints. The [Hung 82] procedure schedules one vehicle at a time and determines the best customer to be inserted into the route. When no more customers can be added to the route, this route is frozen and the process is repeated over the remaining set of unassigned customers. This is a sequential insertion procedure and generally, the routes generated first are the most productive, while the routes generated last tend to be inferior in quality. A procedure, which uses concurrent insertion based on proximity measures, has been developed by [Ray 84a,84b]. This algorithm takes a set of known requests and simultaneously constructs routes for all vehicles starting at the beginning of the day by using proximity measures. It can also insert new requests into a set of existing routes with the possibility of initializing new routes as needed.

[Jaw 86] presents a concurrent procedure. It uses a parallel insertion algorithm which appears to be very effective and efficient in minimizing a weighted combination of customer inconvenience and system costs. The day is split a priori into time slices of 20-30 minutes in length. Assuming that partial vehicle schedules exist over all preceding periods, this procedure forms clusters of customers for each of the vehicles in the given period. It successfully solves very large scale problems of 2000 requests. [Bodin 86] presents another procedure which is a traditional "cluster-first route-second" approach. For a fixed fleet size, it partitions the set of requests into vehicle clusters and solves the resulting single vehicle dial-a-ride problems using a heuristic based on Benders' decomposition. Requests are then moved one at a time from one vehicle to another while attempting to reduce total user inconvenience.

[Desrosiers 86] solves the problem by mini-clustering first and optimal routing second, an innovative modification of the "cluster-first route-second" method which permits a more intelligent clustering of customers. A heuristic algorithm groups together customers who can be efficiently serviced by the same vehicle route segment to form miniclusters. An optimal algorithm then constructs routes corresponding to drivers' workdays by stringing together these vehicle route segments. In a methodology of "cluster-first route-second" type, a cluster is formed from the set of customers assigned to a vehicle, and routing is carried out separately for each vehicle. The routing problems are usually easy to solve. But the most important decisions are taken at the clustering stage and it is very difficult to globally construct a good set of clusters. The authors' approach moves a part of the clustering problem into the routing problem. Large problems are solved by dividing the day into time slices and applying the algorithm several times. [Dumas 89] presents a revised heuristic to solve the same problem. The main contributions are first, the innovative concept of mini-clusters; second, the generalization of the column generation algorithm for the m-TSP with time windows to solve problems with several

depots and resource constraints; and, finally, a method of decomposition into time slices and sub-areas to handle very large problems.

1.3 The Related Paratransit Environment

There will be some special conditions and constraints which must be satisfied when we apply the dial-a-ride problem to paratransit service system. The environment is different from that of the normal DARP. It is, therefore, important to review the vehicle routing and scheduling process, understand the objectives and constraints, and identify desirable features. So we will briefly introduce some basic characteristics of the paratransit environment here.

The vehicle operating efficiency and service quality of demand-responsive transportation systems for elderly and handicapped persons are influenced greatly by the way vehicles are routed and scheduled [Kikuchi 87]. The operating efficiency is generally much lower than that for conventional fixed-route, fixed-schedule transit service. Service quality is characterized by personalized "demand responsive" service. However, schedule reliability and variations of on-board travel time are often cited as service problems. Operating efficiency improvement often degrades passenger service quality, such as trip departure time adjustment, circuitous travel routes, and many intermediate stoppings. Furthermore, there exist unique vehicle operating characteristics and passenger service requirements to be considered for the transportation of elderly and handicapped persons. Thus, the development of vehicle routing and scheduling is a very complex process, requiring a systematic analysis of the entire process

Although great variations exist among the agencies, generally, the process of developing vehicle routing and scheduling consists of four sequential phases: (1) reservation; (2) vehicle routing and scheduling (stop-list preparation); (3) vehicle location monitoring and schedule adjustment; (4) statistics and accounting records preparation.

Reservation

Most demand-responsive systems operate on a reservation basis in which passengers phone the reservation office to reserve trips from 2 to 3 weeks to (normally) 24 hours before their actual trips. The telephone receptionist verifies passenger eligibility, and records the information on the person and the trip, including trip origin-destination, desired departure or arrival time and special instructions. All passenger trip requests are organized in sequence of departure time or sorted by origin location or destination location to facilitate the vehicle routing/scheduling that takes place in the next phase.

Vehicle Routing and Scheduling (Vehicle Stop-List Preparation)

Based on the list of passenger trip requests for the following day, the scheduler develops the route and times for passenger pickup and dropoff for each vehicle or driver. The sequential listing of vehicle stop locations and the corresponding times is called a "stop-list". The development of the stop-list is the most difficult and crucial phase in the vehicle routing/scheduling process. Three main operating scenarios had been developed [Belisle 84] (see Table 1.1), where a call-back policy is that the request is noted down at the time the user calls and the user is called back later to confirm the service and time of departure, a no call-back policy is that the final response is given to the user at the time of calling. Obviously the operating efficiency, service level, service policy, system limitations, geographical conditions, traffic conditions, and a host of uncertainties and constraints must be considered by the scheduler, when preparing the stop-list.

Scenario number	Vehicles used to transport users	Advance request handling policy
1	minibuses	call-back
2	minibuses	no call-back
3	mainly taxis	no call-back

Table 1.1: Three main operating scenarios in paratransit system

Vehicle Location Monitoring and Schedule Adjustment

Once the vehicle routing and scheduling is determined and the vehicles leave the garage or driver homes for revenue service, the dispatcher or supervisor will monitor the movements of vehicles and gives schedule adjustment instruction when unexpected events or delays occur. This task is continued on a real-time basis, requiring thorough knowledge of the system and area conditions.

Statistical and Accounting Records Preparation

This phase begins when vehicles return from revenue service and drivers turn in their log sheets containing the information on the pickup and dropoff times of each passenger trip and odometer readings. The activities of each driver and vehicle are then summarized and accounting reports are prepared for sponsoring agencies. The driver and vehicle activities information is used to analyze vehicle routing and scheduling efficiency and passenger service quality.

The entire process must be executed in a well coordinated and organized manner. Traditionally, for most systems, vehicle routing and scheduling has been performed manually by "experienced" dispatchers who are thoroughly familiar with the characteristics of the passengers, drivers, road network, trip patterns and system limitations. In recent years, microcomputer based vehicle scheduling programs have been marketed to improve the service quality and operating efficiency of the dispatch system. Many agencies are contemplating the computerization of vehicle dispatching, statistical analyses and accounting functions now. So for a large system, each phase is conducted by a separate group of employees, while for a small system, the entire procedure may be completed by one person. Figure 1.6 shows a practical structure of a general paratransit system, while the work involved in the thesis is part of the scheduler.

Figure 1.6: A practical structure of general paratransit system

1.4 Focus of the Thesis

The main objective of this thesis is to develop post-optimization procedures for the DARPTW in the transportation of elderly and handicapped persons. Current methods generate feasible routes and schedules, but so far little work has been done on improving the quality of the solution obtained. In this thesis, different post-optimization procedures are provided to improve the initial routes, and computational results obtained from these procedures are analyzed and compared as well.

Briefly, there are three parts of the work involved. The first part is route initialization by using the insertion heuristics. The second part is our post-optimization procedures which use 2-opt, 3-opt, and Or-opt backward and forward search algorithms to improve the quality of initial routes based on the chosen objective functions. The third part is the implementation and testing of our algorithms. The results obtained are used to compare the different post-optimization methods and draw some conclusions about them. A more detailed description of the three parts follows:

1) Route Initialization (Insertion Heuristics)

Since the primary requirement for demand-responsive paratransit systems is scheduling speed, the route construction algorithm is based on the least-cost insertion heuristics (see Section 1.1). It consists of two components: demand-responsive single request insertion and batch parallel route construction.

The demand-responsive request insertion routine is used to immediately insert a request into the service day schedule at the time of call entry. This insertion routine calculates, in real-time, the best feasible insertion point of the pickup and dropoff locations for the request, within all existing routes for the required day. Optimality is based on the chosen objective, and feasibility is with respect to pickup and dropoff precedence, service time windows, and vehicle capacity. If it is less expensive to initialize a new route, and if an additional staffed vehicle is available, a new route is started.

The batch parallel route construction routine is based on a "cluster-first route-second" algorithm [Roy 84a]. Since we do not have a map software package, the requests cannot be initially grouped by geographical proximity and travel direction. So we group the requests by their desired service time first. Then, a number of routes are initialized with these request clusters. This requires matching the request requirements with the route's service limitations. Later an insertion routine iteratively computes the best feasible insertion point for each request in each existing route and inserts that request into the route that minimizes the insertion cost. The calculation of the "best" insertion point depends on the

company's chosen objective. In addition, the dispatcher/scheduler can change operating parameters or objectives, and invoke the batch routine to generate alternate schedules.

2) Post-optimization Heuristics

The post-optimization heuristics consist of two components: *intra-route* and *inter-route* optimization methods. Intra-route optimization involves stop interchanges within a single route, while inter-route optimization involves request interchanges between routes. Both algorithms are based on the k-interchange concept (see Section 1.1).

For intra-route optimization method, testing the feasibility of a given solution in the straightforward way requires time which is linear in the number of stops. [Savelsbergh 85] and [Savelsbergh 88] introduce a new method, which we will use later, to reduce this effort to constant time. The objective functions we considered here are as follows:

- Minimize the actual total travel time
- Minimize the total completion time
- Minimize the partial completion time & actual total travel time
- Minimize the loaded vehicle waiting time
- Minimize the excess travel time
- Maximize the vehicle productivity rate, where

$$\text{Vehicle Productivity Rate (VPR)} = \frac{\text{number of people transported}}{\text{travel time of vehicle} + \text{handling time}}$$

Meanwhile, both backward and forward searches are being used within the Or-opt scheme to post-optimize each separate route. The output of the procedures are the schedules, that is, stop-lists, for each vehicle / driver.

For the inter-route optimization method, we design a new algorithm which is based on the insertion heuristics. Since the customer precedence constraint has to be satisfied, and pickup, dropoff stops must be serviced by the same vehicle, we can only interchange

two requests from two different routes instead of stop interchanges in the case of intra-route post-optimization. This is the main difference between intra and inter route post-optimization. Based on the objective functions, we will check the feasibility constraints before doing the inter-route interchange. The most difficult part here is to choose the proper requests for interchanging. Since the post-optimization procedures are very time consuming, we only consider those requests with longer excess travel time for the interchange so that we can minimize the customer's inconvenience.

3) Implementation

To test the computational performance of the described algorithms, we use the data of a specific day in a middle size city of Ontario, Canada. After generating the initial routes by using our insertion heuristics, we use 2-opt, 3-opt, Or-opt backward search and Or-opt forward search algorithms on the initial routes to do our intra-route post-optimization. The results obtained from the improved routes are based on the different subtests and different objective functions. The programs were written in C language and run on the IBM PC-486 on SCO UNIX operating system.

1.5 Outline of the Thesis

Chapter 2 introduces the route initialization which includes the problem description, the kind of data needed for the problem, the assumptions that have been made and the constraints and parameters. Chapter 3 first introduces the intra-route post-optimization procedures which use 2-opt, 3-opt and Or-opt backward and forward schemes to improve the initial routes. Inter-route optimization algorithm based on the insertion heuristics is discussed then. Chapter 4 describes the implementation and computational results obtained from 6 test cases. A comparison of the results is made among the different post optimization procedures. Finally Chapter 5 gives a summary of the work and some suggestions for possible additions and improvements in the future to the DARPTW in paratransit.

Chapter 2

Route Initialization

The DARPTW in paratransit contains two types of problems. The first problem involves the construction of routes to respond to a set of requests known in advance. The second involves the integration of a new request into an already existing route. We use insertion heuristic algorithm to handle both problems.

The reason for using the insertion heuristics to initialize routes depends on the reported computational experiments for VRSPTW. [Solomon 87] indicates that the insertion heuristic algorithm performs the best, when the solution quality is measured in terms of the minimum number of vehicles, minimum schedule time, minimum distance and minimum waiting time. The behavior of the insertion heuristics is very stable across all problem environments, and the efficiency increases with the increase of the percentage of time window constraints and their tightness. Further support for this conclusion is provided by [Jaw 86] who shows the effectiveness of an insertion-based procedure in a time-window constrained real dial-a-ride environment.

2.1 The DARPTW Model

In this problem, vehicles pick up handicapped people at their places of origin and take them to their destinations, without exceeding vehicle capacities, and respecting the precedence relationships and time window constraints at both the origins and the destinations. The basic objective is to find an itinerary which minimizes the total distance travelled.

Before describing the heuristic algorithm, we state the assumptions and introduce the data required and the notation to be used throughout the remainder of this thesis.

Usually there is a set of driver shifts which must be input into the heuristics. These shifts are related to the number of vehicles required for each period of the day. Within a day, each driver can have several shifts, and each shift contains only one task and uses one vehicle. So each route produced by the heuristics corresponds to a task (in this thesis, a task will be equivalent to a shift). For each SHIFT(k) in workday j, we have the following information:

Figure 2.1: A single depot DARPTW

- origin(k)** : indicator of location where SHIFT(k) begins,
destination(k) : indicator of location where SHIFT(k) finishes,
start_time(k) : starting time for SHIFT(k),
finish_time(k) : finishing time for SHIFT(k),
vehicle_num(k): the vehicle number for this SHIFT(k).

2.1.3 The Distance Table

Since each REQUEST(i) corresponds to two stops: the pickup(i) and the dropoff(i), four ways can be used to calculate the distance $D(p,d)$ of two stops, p and d, depending on

the real situation. The first way is to use so called Manhattan distance to estimate $D(p, d)$, where (x_p, y_p) and (x_d, y_d) are the Cartesian coordinates of stops p and d :

$$D(p, d) = |x_p - x_d| + |y_p - y_d|$$

The second way is to use the Euclidean distance (crow fly distance) to estimate $D(p, d)$ (see Figure 2.2)

$$D(p, d) = \sqrt{(x_p - x_d)^2 + (y_p - y_d)^2}$$

Figure 2.2: Two ways to calculate distance $D(p, d)$

In most practical cases, each service company (e.g., taxi company) will define its own zones first by using the city street files from the Ministry of Transportation (see Figure 2.3). Then the service company can use some software packages (e.g., MAPINFO) to create the street files for their database systems, each file containing the defined customized zones. In this way, the city road network can be transformed into a graph where an arc represents a street segment and a node represents an intersection. Each stop is associated with a node of the graph. So the third way to calculate the distance is to use a shortest path algorithm to estimate $D(p, d)$, where the algorithm is applied to the transformed graph.

The fourth way is to use the longitude and latitude to calculate the distance from pickup stop p and dropoff stop d . Suppose that the longitudes and latitudes for stop p and stop d are $(\text{lon}_p, \text{lat}_p)$ and $(\text{lon}_d, \text{lat}_d)$ respectively, the following formula will be used:

$$D(p,d) = \begin{cases} 0 & \text{if } \text{lon}_p = \text{lon}_d \text{ and } \text{lat}_p = \text{lat}_d \\ \text{EARTH_RADIUS} * \text{acos} (\sin (\text{lat}_p * \text{PI}/180) * \sin (\text{lat}_d * \text{PI}/180) \\ + \cos (\text{lat}_p * \text{PI}/180) * \cos (\text{lat}_d * \text{PI}/180) * \cos (\text{dlon} * \text{PI}/180)) & \text{otherwise} \end{cases}$$

where $\text{dlon} = \text{lon}_p - \text{lon}_d$, $\text{PI} = 3.1415926$ and $\text{EARTH_RADIUS} = 6378.15 \text{ km}$

Figure 2.3: The defined customized zones

By using any of these four ways above, we can get the estimated distance. In this thesis we use the longitude and latitude to get the distance because of the test data. Since an estimation of the distance between two stops is needed frequently in the heuristic algorithm, we can set up a distance table first during a pre-processing stage to save time.

2.1.4 The Speed Table

The system allows the user to define speed as a function of distance where distances are those between the major intersections along the shortest path. But the same distance does not mean the same travel time is used. Figure 2.4 shows that the distances of A-->B and C-->D are the same, but the time to go from A-->B will be longer than the time to go from C-->D because of more intersections. Usually for short distances, there is always a fixed time required for acceleration and deceleration, and for long distances, higher speeds are possible because of the ability to use fast lanes. So if the shortest path is not stored, the time between major intersections should be calculated immediately in parallel with the distance calculation and then a TIME matrix should be stored. More specifically,

we should use the time between adjacent intersections to calculate fastest path. In this way, the speed function is specified by using following six parameters (see Figure 2.5):

Figure 2.4: A case with same distance

Figure 2.5: Speed function

VMIN: minimum speed (km/hr),

VAVG: average speed (km/hr),

VMAX: maximum speed (km/hr),

d₁ : the distance at which VAVG begins,

d₂ : the distance at which the speed becomes higher than VAVG,

d₃ : the distance at which VMAX begins.

For a distance d , the function gives a speed $V(d)$ where d is the distance between the major intersections along the shortest path. Then the travel time t can be calculated as $t = d/V(d)$. It is important that a given travel time corresponds to only one distance. To ensure this, the speed function must satisfy the condition that any straight line passing through the origin must intersect the speed function at only one point. This constraint ensures that a given time corresponds to a single distance, enabling this distance to be found.

2.1.5 Handling Time

Normally the handling time is a fixed time for ambulatory or wheelchair people at each stop unless there are two or more requests having the same pickup and/or dropoff stops. In this situation, the handling time will be a function of the number and type of handicapped persons included at the same stop for different requests. For each stop, suppose that

HT_WC: the time required for handling a wheelchair person for pickup or dropoff,

HT_AM: the time required for handling an ambulatory person for pickup or dropoff,

NUM_WC: the total number of wheelchair people being picked up or dropped off,

NUM_AM: the total number of ambulatory people being picked up or dropped off.

Obviously we have to consider the time when the driver alighted the vehicle and opened the door. So the complete handling time for the first individual is included plus half the basic handling time for each additional person. In this way, for REQUEST(i), the handling time at STOP(pd) is given by:

$$HT(pd) = \begin{cases} 1/2 * HT_WC * (1 + NUM_WC) + 1/2 * HT_AM * NUM_AM & \text{if } NUM_WC > 0 \\ 1/2 * HT_AM * (1 + NUM_AM) & \text{if } NUM_WC = 0 \end{cases}$$

The scheduler may specify the handling time at the origin or destination of a request to take into account special cases. If no time is specified, the function given above is used.

2.1.6 The Quality of Service

We will use several parameters, for each REQUEST(i), to represent the quality of service:

AVD(i): the actual time at which the vehicle leaves the stop pickup(i),

AVA(i): the actual time at which the vehicle arrives to the stop dropoff(i).

We define the passenger excess travel time $ET(i)$ for $REQUEST(i)$ as follows:

$$ET(i) = AVA(i) - AVD(i) - t(\text{pickup}(i), \text{dropoff}(i)),$$

where $t(\text{pickup}(i), \text{dropoff}(i))$ is the direct travel time from: $\text{pickup}(i)$ to $\text{dropoff}(i)$.

As we mentioned in 2.1.1, $DST(i)$ is the desired precise service time depending on the request type (departure or arrival time desired). Here we define:

$PWT(i)$: deviation from desired service time (passenger waiting time) where

$$PWT(i) = \begin{cases} AVD(i) - DST(i) & \text{if request_type}(i) = 'D' \text{ for departure time required} \\ DST(i) - AVA(i) & \text{if request_type}(i) = 'A' \text{ for arrival time required} \end{cases}$$

The service company may constrain $PWT(i)$ to be always non_negative, that is, a departure required request must not be picked up prior to the requested service time, and an arrival required request must not be dropped off later than the requested service time. If $PWT(i)$ is less than 0, we will define the *maximum passenger time violation* $MAX_PTV(i)$. Other constraints on the quality of service provided to users are specified by the following parameters: *maximum passenger waiting time* $MAX_PWT(i)$, *maximum excess travel time* $MAX_ET(i)$ and *maximum travel time* for a passenger staying in the vehicle MAX_VOYAGE , that is,

$MAX_PWT(i)$: maximum deviation from desired service time of $REQUEST(i)$,

$$PWT(i) \leq MAX_PWT(i) \quad \text{if } PWT(i) > 0,$$

$MAX_PTV(i)$: $|PWT(i)| \leq MAX_PTV(i) \quad \text{if } PWT(i) < 0,$

$MAX_ET(i)$: maximum excess travel time of $REQUEST(i)$,

$$ET(i) \leq MAX_ET(i),$$

MAX_VOYAGE : absolute maximum travel time for any passengers,

$$AVA(i) - AVD(i) \leq MAX_VOYAGE.$$

Usually we formulate $MAX_ET(i)$ as a percentage of the duration of a direct trip, e.g., 100% of direct travel time $t(\text{pickup}(i), \text{dropoff}(i))$. If the actual excess travel time is less than $MAX_ET(i)$, the service agency will not be penalized. Otherwise we use $MAX_ET(i)$ instead of $ET(i)$. For the MAX_VOYAGE , it is a user-defined parameter that is set to ensure that long direct trips are not prolonged unreasonably. For example, if $MAX_ET(i) + t(\text{pickup}(i), \text{dropoff}(i))$ equals to 80 minutes, a value of $MAX_VOYAGE = 60$ minutes would limit the trip to one hour instead of 80 minutes. Similarly we can get:

$$MAX_VOYAGE \geq t(\text{pickup}(i), \text{dropoff}(i)) + MAX_ET(i) \quad \text{for any REQUEST}(i).$$

2.1.7 Setting Up Time Window

Each $REQUEST(i)$ involves two stops, an origin and a destination. For each of these stops, we define a *time window* as the time interval during which the stop may be visited. Setting up time window at stop pd will depend on the $request_type(i)$ and the desired service time $DST(i)$ of $REQUEST(i)$. Other constraints must be also satisfied, such as $MAX_PWT(i)$, $MAX_ET(i)$ and MAX_VOYAGE constraints. Usually when working with time windows, the maximum excess travel time $MAX_ET(i)$ and maximum total travel time MAX_VOYAGE constraints are relaxed at the beginning. At post-optimization stage, we restrict these constraints to get better results. So for each stop pd , we define

$e(pd)$: the earliest arrival time at stop pd ,

$l(pd)$: the latest arrival time at stop pd ,

$A(pd)$: the actual vehicle arrival time at stop pd ,

$D(pd)$: the actual vehicle departure time at stop pd ,

$W(pd)$: the vehicle waiting time at stop pd .

When vehicle arrives to stop pd earlier than its earliest arrival time $e(pd)$, the vehicle waiting time $W(pd)$ occurs. We define $W(pd)$ as follows:

$$W(pd) = \begin{cases} e(pd) - A(pd) & \text{if } e(pd) > A(pd) \\ 0 & \text{otherwise} \end{cases}$$

Consider a REQUEST(*i*) with pickup(*i*), dropoff(*i*) and request_type(*i*). If request_type(*i*) = 'D', where departure time is desired, then we will set up time windows for both pickup(*i*) and dropoff(*i*) as follows (see Figure 2.6):

Figure 2.6: Time window (departure time desired)

Figure 2.7: Time window (arrival time desired)

If request_type(*i*) = 'A', where arrival time is desired, then we can set up the time windows as in Figure 2.7.

Figure 2.8: Two examples of time windows

Figure 2.8 shows two examples for REQUEST(i) and REQUEST(j) based on the following data:

REQUEST(i)

request_type(i) = 'D'

DST(i) = 9:00 a.m.

MAX_PWT(i) = 30 minutes

 $t(\text{pickup}(i), \text{dropoff}(i)) = 15 \text{ minutes}$
 $\text{MAX_ET}(i) = 100\% t(\text{pickup}(i), \text{dropoff}(i))$
 $= 15 \text{ minutes}$

HT(i) = 0

REQUEST(j)

request_type(j) = 'A'

DST(j) = 9:00 a.m.

MAX_PWT(j) = 30 minutes

 $t(\text{pickup}(j), \text{dropoff}(j)) = 15 \text{ minutes}$
 $\text{MAX_ET}(j) = 100\% t(\text{pickup}(j), \text{dropoff}(j))$
 $= 15 \text{ minutes}$

HT(j) = 0

MAX_VOYAGE = 45 minutes

MAX_VOYAGE = 45 minutes

2.1.8 Clustering the Requests

The most popular method used by insertion heuristics is the “cluster first, route second” method. It is proven to be a very efficient method when we consider the multiple vehicle DARPTW. So the multiple vehicle problem can be reduced to several problems with one vehicle by determining the requests to be handled by the same vehicle. The difficulty with this method lies in the construction of clusters. Obviously requests with stops which are close to one another in both space and time are likely to use the same vehicle. [Roy 84a] provides an algorithm to get clusters by using a concept of “neighbours” of requests. In this thesis we cluster the requests based only on their time, not space. The reason for doing this is the difficulty of clustering the requests according to their geographical locations, unless we have a map software package, e.g., MAPINFO, to do our tests.

All requests are sorted by their desired service time or pickup stops’ earliest arrival time during the pre-processing stage. Then we cluster these requests according to the CLUSTER_TIME_INTERVAL set up by the distribution of the desired service time of each request or the earliest arrival time of each pickup stop. For each cluster, we try to use the vehicles for which their drivers have the same shift to get the initial routes. The quality of these routes will be influenced by the quality of the clusters. It is still a very difficult job to find more efficient algorithms for clustering the requests at the very beginning.

2.1.9 The Measure Matrix

Now we wish to define a measure matrix $INSERT(i, r)$ between a $REQUEST(i)$ and a $ROUTE(r)$ to guide the parallel insertion heuristics used later. This measure matrix must reflect the ease with which the request can be inserted into the route. Since each request corresponds to two stops, the precedence relationship must be requested, that is, the customer must be picked up before being dropped off. Because of the time window

constraint at each stop, we only consider the feasible insertions which will not violate the time window of any stops. So this measure must integrate not only “space” (distance) but also “time” dimensions.

We first introduce a method of checking whether the insertion of a REQUEST(i) into the given ROUTE(r) is feasible or not. Because of the time windows, precedence and capacity constraints, we insert the pickup stop of REQUEST(i) into ROUTE(r) first, then we insert the dropoff stop into the segment beginning at the stop following the pickup stop inserted to satisfy the precedence constraint. If one of these two insertions fails, then we keep moving the insertion stop for the pickup until all the stops of ROUTE(r) have been tested. If we still cannot insert both the pickup and the dropoff stops of REQUEST(i) into ROUTE(r), then REQUEST(i) is not allowed to be inserted into the ROUTE(r) feasibly. The feasibility conditions are based on the distance detour, the time window of every stop in ROUTE(r), the time windows of the pickup and the dropoff stops of REQUEST(i) and the time loss caused by the insertion. Usually the time windows limit our searches for an insertion stop.

Now let us consider the general case. Suppose we have a given ROUTE(r) and we want to insert a REQUEST(i) which contains pickup stop(PICK) and dropoff stop(DROP) into ROUTE(r) (see Figure 2.11). The time windows for stop(PICK) and stop(DROP) are $[e(\text{PICK}), l(\text{PICK})]$ and $[e(\text{DROP}), l(\text{DROP})]$, respectively. First we try to insert pickup stop(PICK) into the position between stop(i) and stop(j) of the ROUTE(r) where the time windows for stop(i) and stop(j) are $[e(i), l(i)]$ and $[e(j), l(j)]$, respectively. Obviously we can do it immediately if

$$D(\text{PICK}) \leq l(\text{PICK}) \text{ and } D^{\text{new}}(j) \leq l(j) \quad \text{---feasibility condition 1}$$

holds, where $D(\text{PICK}) = \max\{e(\text{PICK}), D(i) + t(i, \text{PICK})\} + \text{HT}(\text{PICK})$, $D^{\text{new}}(j)$ is the new vehicle departure time at stop(j) after the insertion, and $D^{\text{new}}(j) = \max\{e(j), D(\text{PICK}) + t(\text{PICK}, j)\} + \text{HT}(j)$. After we have inserted stop(PICK), the distance detour and time

Figure 2.9: Insertion of stop(PICK) to the given ROUTE(r)

loss will incur. Here we use $D_DETOUR(PICK)$ and $T_LOSS(PICK)$ to represent them respectively. We can get:

$$D_DETOUR(PICK) = d(i, PICK) + d(PICK, j) - d(i, j)$$

$$T_LOSS(PICK) = A^{new}(j) - A(j)$$

where $A^{new}(j)$ is the new vehicle arrival time at stop(j) after the insertion of stop(PICK). If $T_LOSS(PICK) > 0$, some of the stops m , $j \leq m \leq \text{depot}$, could become infeasible depending on the difference of $D^{new}(j) - D(j)$ at stop(j). If $D^{new}(j) - D(j) > 0$, then we will push forward $D^{new}(j) - D(j)$ units of time in the rest of the route beginning at stop(j). Here we define $pfs(j)$ as *push forward shift* at stop(j) as:

$$pfs(j) = D^{new}(j) - D(j).$$

Furthermore,

$$pfs(x + 1) = \max\{0, pfs(x) - W(x+1)\}, j \leq x \leq \text{depot} - 1.$$

It is easy to see, when $pfs(j) > 0$, we should examine the stops sequentially for time feasibility until we find a stop(k) with $k < \text{depot}$ for which $pfs(k) = 0$, or stop(k) is time

infeasible, or, in the worst case all the stop(k), $j \leq k \leq \text{depot}$, are examined. We have just proved:

Lemma 1. The necessary and sufficient conditions for time feasibility when inserting a stop, say u , between stop(i) and stop(j), $1 \leq i, j \leq \text{depot}$, on a partially constructed feasible route (1, 2, ..., depot) are:

$$D(u) \leq l(u), \text{ and}$$

$$D(x) + pfs(x) \leq l(x), \quad j \leq x \leq \text{depot}.$$

Note that if we assume non-Euclidean travel distances, then it is possible that $pfs(j) < 0$ which leaves all the customers time feasible. Also if we can find a stop(k), $j \leq k \leq \text{depot}$, where $pfs(k) = 0$, then we stop checking the feasibility of the segment beginning at k to depot, because all the stops at that segment will be time feasible. Furthermore, since the depot has its own time window for a specific vehicle, using Lemma 1 will ensure that any request that does not permit the vehicle to return to the depot within the scheduled time will not be added to the partial route.

A similar way can be used to insert dropoff stop(DROP) into the given ROUTE(r) in which the new updated ROUTE(r) is still a feasible route (see Figure 2.12). The only difference is the value of time loss $T_LOSS(DROP)$. The time loss $T_LOSS(PICK)$ might influence $T_LOSS(DROP)$ depending on the value of $pfs(t)$ at stop(t) after the dropoff stop(DROP) has been inserted. So the total time loss when we insert REQUEST(i) into the ROUTE(r) will be the time loss $T_LOSS(DROP)$ which includes the impact of time loss as a result of the insertion of pickup stop(PICK). Now we can define two parameters for evaluating the insertion of REQUEST(i) into ROUTE(r) as follows:

$$\begin{aligned} \text{ATD_DETOUR} &= \text{actual total distance detours} \\ &= D_DETOUR(PICK) + D_DETOUR(DROP) \end{aligned}$$

$$= \{d(i, \text{PICK}) + d(\text{PICK}, j) - d(i, j)\} + \{d(s, \text{DROP}) + d(\text{DROP}, t) - d(s, t)\}$$

$$\begin{aligned} \text{ATT_LOSS} &= \text{actual total time losses} \\ &= \text{T_LOSS}(\text{DROP}) \\ &= \text{pfs}(t) \end{aligned}$$

Figure 2.10: Insertion of a REQUEST(i) to the given ROUTE(r)

We can define the measure matrix $\text{INSERT}(i, r)$ between a REQUEST(i) and a ROUTE(r) as follows:

$$\text{INSERT}(i, p) = \begin{cases} \text{Min} \{ \alpha * \text{ATD_DETOUR} + \beta * \text{ATT_LOSS} \} & \text{---(for all feasible insertions)} \\ \text{Maxmeasure} & \text{---(if there is no feasible insertion)} \end{cases}$$

where α , β and Maxmeasure are constants, $\alpha + \beta = 1$.

Thus, $\text{INSERT}(i, r)$ indicates either that insertion of the REQUEST(i) into the ROUTE(r) is infeasible, or that the insertion is feasible and in this case, the function also

provides an estimate of the distance detour and the time loss involved in inserting REQUEST(i) into the given ROUTE(r). Obviously, this measure can also estimate the cost of an insertion.

The following is an algorithm for getting the measure matrix INSERT(i, r). Suppose we have M routes, 1, 2, ..., M, and each route contains maximum N stops (N will be a parameter). Now we want to insert a REQUEST(i) into one of the M routes. So we have to calculate INSERT(i, 1), INSERT(i, 2), ..., INSERT(i, M) first, then we choose the ROUTE(r) for which its value of INSERT(i, r) is a minimum. If there are more than one route with the minimum INSERT(i, r) value, we randomly choose one. Otherwise, we cannot insert the REQUEST(i) into any existing route. Then we will initialize a new route for REQUEST(i) if it is possible.

Algorithm for inserting a REQUEST(i) into the existing M routes

Insertion()

```
{
  for( r=1; r≤M; r++ )
  {
    Min=Maxmeasure
    for( j=1; j≤N-1; j++ )
    {
      choose stop(j) and stop(j+1) for insertion of pickup stop(PICK)
      if( feasibility condition 1 holds )
      {
        pseudo_insert stop(PICK) into ROUTE(r)
        for( k=PICK; k≤N-1; k++ )
        {
          choose stop(k) and stop(k+1) for insertion of dropoff stop(DROP)
          calculate pfs(k+1)
          if( necessary and sufficient condition holds )
          {
            pseudo_insert stop(DROP) into ROUTE(r)
            calculate obj fm
```

```

        if( obj fm < Min )
        {
            store obj fm as Min
            store the route number r
            store positions of stop(PICK) and stop(DROP)
        }
    }
}
INSERT (i, r) = Min;
}
if( there exists a r where INSERT (i,r) < Maxmeasure )
{
    Insert pickup stop(PICK) and dropoff stop(DROP) of REQUEST(i) into the
    positions of ROUTE(r) where its INSERT (i, r) value is minimum.
    if( stop(PICK) is inserted after depot )
    {
        modify the actual arrival and departure time of every stop in the route
        modify the route start and finish time
    }
    else
    {
        modify the actual arrival and departure time of the stops after stop(PICK)
        modify the route finish time
    }
}
else
{
    REQUEST(i) cannot be inserted into any one of the M routes
}
}

```

Here the $\text{obj } f^m = \alpha * \text{ATD_DETOUR} + \beta * \text{ATT_LOSS}$. Obviously, it takes $O(N^3M)$ to insert a REQUEST(i) into the existing routes.

2.1.10 The Horizon Measure

Since the order of an insertion of a request is independent of the routes under construction, that is, an afternoon request does not influence the insertion of a morning request, it is useless to consider all the requests for a given day at once. We therefore consider at the same time only requests having their desired service time within a given horizon. This horizon is determined by two values:

HORIZON_START: The starting time of the horizon corresponds to the earliest desired service time among all the requests not yet inserted,

HORIZON_FINISH: The ending time of the horizon corresponds to the desired service time of the next request to enter into the horizon.

The management of horizon is based on the following values:

HORIZON_PREREQ:	The number of requests presently in the horizon.
HORIZON_MAX:	The maximum time interval allowed for the horizon.
HORIZON_MAXREQ:	The maximum number of requests allowed within the horizon.

The horizon moves forward in time by adding the next request as long as

$(\text{HORIZON_FINISH} - \text{HORIZON_START}) \leq \text{HORIZON_MAX}$, and

$\text{HORIZON_PREREQ} \leq \text{HORIZON_MAXREQ}$

In this way, the horizon advances gradually in time. The value of **HORIZON_MAX** corresponds to the parameter **CLUSTER_TIME_INTERVAL** mentioned in 2.1.8 where we use it to set up the clusters.

There are two reasons for using the horizon measure. The first one is that the continual adjustment of the horizon ensures that no requests are left behind without being inserted. The second reason is that the gradual movement of horizon helps us to limit

searches for inserting pickup stop(PICK) and dropoff stop(DROP) of REQUEST(i) into the given ROUTE(r), because the request within the horizon keeps going towards the end of the day. So using horizon measure definitely will give us an efficient way to do the route initialization.

2.2 The Insertion Heuristic Algorithm

The heuristic algorithm is based on an insertion technique in which we try to insert all the requests within the same cluster into the same route by not violating the precedence, time window and vehicle capacity constraints. Eventually every request will be either inserted into one of the existing routes according to its values of INSERT(i,r) or as needed new route will be initialized for it, unless no driver's shift fits this request.

We construct clusters with the uninserted requests by using the way mentioned in Section 2.1.8 in which the groups of requests are used to initialize several routes at the beginning and to determine the next requests to be inserted. So the insertion heuristic algorithm will contain the following parts.

2.2.1 Route Initialization

For each cluster, a route is initialized and an attempt is made to insert all the requests of the cluster into it. The route is initialized by finding an available driver's shift on a workday which can carry out the earliest request in the cluster. The shift chosen is that for which the minimum possible time will be obtained between the departure time of the driver and the time at which the vehicle must pick up the earliest request of the cluster. Normally we can do this by sorting the requests based on their earliest arrival time of the pickup stop(PICK) or their desired service time.

2.2.2 Selection of the Best Route for a Given Request

Since the measure matrix INSERT(i,r) provides an indication of the total distance

detour and total time loss caused by the insertion of a REQUEST(*i*), if this insertion is feasible, therefore the best route for REQUEST(*i*) is the ROUTE(*r**) such that

$$\text{INSERT}(i, r^*) = \min \{ \text{INSERT}(i, r) \} \quad \text{for all existing routes } r \text{ in the cluster.}$$

2.2.3 Insertion of a Request into a Route

After we get the value of INSERT(*i*, *r**), we insert the REQUEST(*i*) into the best ROUTE(*r**) unless INSERT(*i*, *r**) equals to Maxmeasure. In this case, REQUEST(*i*) cannot be inserted into any one of the existing routes. We will initialize a new route for it if there are some available drivers' shifts.

The procedures to insert REQUEST(*i*) into ROUTE(*r**) are discussed in details in Section 2.1.9 where we design an $O(N^3M)$ algorithm to do the work (*N* is the maximum number of stops allowed in each existing route, and *M* is the number of routes).

2.2.4 The Heuristic Algorithm

We will now present the whole insertion heuristic algorithm as follows:

Route_Initialization()

```
{
    sort all requests according to their desired service time or earliest pickup time
    read the requests to initialize the horizon and to construct the clusters
    initialize one route for each cluster based on driver's shift
    while (not all requests have been processed)
    {
        for( c=1; c ≤ NUM_CLUSTER; c++ )
        {
            read every REQUEST(i) in CLUSTER(c)
            calculate INSERT(i, r*) where r is an existing route of CLUSTER(c)
            if( best ROUTE(r*) in CLUSTER(c) found )
            {
                insert REQUEST(i) into ROUTE(r*)
                modify ROUTE(r*)
            }
        }
    }
}
```



```
        else
        {
            if( available driver's shift exists )
            {
                initialize a new route for REQUEST(i)
            }
            else
            {
                put REQUEST(i) into a special request list where all
                the requests cannot be serviced
            }
        }
    }
}
```

After the route initialization, if an additional request REQUEST(x) is given. We calculate the INSERT(x, r), where r is any existing route, to find out the best route r^* for it. The fact that we check all existing routes instead of routes in the same cluster only distinguishes this case of inserting a separate request after the route initialization.

2.3 Measures of Quality

From the insertion heuristic algorithm above, we can find that there are several factors which may influence the quality of our initial routes. The first factor is the size of our horizon period. This will decide the number of the first few clusters. If the horizon period HORIZON_MAX is too big or too small, it will influence the cluster numbers. The second factor is the driver's shift. If the start_time and finish_time of the shift are set improperly, some requests will not be able to be serviced. The third factor is the type of the vehicle chosen. Since some vehicles can only pick up wheelchair or ambulatory people (but not both), the way how we assign the types of vehicle to the cluster will definitely influence the quality of our initial routes.

Chapter 3

Post-Optimization Heuristics

In this chapter, we develop post-optimization heuristics for route improvement based on the k -interchange concept of the Travelling Salesman Problem (TSP). The post-optimization procedures consist of two components: *intra-route* optimization which involves interchanges within a single route, and *inter-route* optimization which involves interchanges between routes. The intra-route optimization heuristics is based on [Savelsbergh 85] and [Savelsbergh 88] where we use local search procedures to limit searches. The inter-route optimization heuristic is a new algorithm based on the insertion heuristics presented in Section 2.2.

3.1 Motivation

The reason we do the post-optimization is because the routes generated by our insertion heuristics are not optimal. There are many factors which can influence the quality of our initial feasible routes (see Section 2.3). From the optimization point of view, intra-route and inter-route post-optimization should be considered separately.

For intra-route post-optimization we use a local search method based on the k -interchange concept, taking care that the feasibility constraints within each route remain satisfied (see Figure 3.1). Usually, in the TSP, the processing of a single k -interchange takes constant time for any fixed value k . One only has to test whether the exchange is profitable and does not have to worry about feasibility. This situation is different for the problem studied in this thesis. When we consider the feasibility constraints, i.e., service time windows, customer precedence and vehicle capacity constraints, the processing of a k -interchange may take $O(N)$ time, where N is the number of stops within this route (each customer corresponds to a request, and each request corresponds to two stops, the origin

and the destination). A modification at one stop may affect the entire route, so that feasibility questions arise. [Savelsbergh 85] and [Savelsbergh 88] introduces a method which only needs constant time for processing of a single 2-interchange or Or-interchange while still satisfying the feasibility constraints. Since the real-time post-optimization procedures are very time consuming, the time complexity reduction from $O(N)$ to $O(1)$ is a big step forward practically.

Figure 3.1: Intra-route optimization

For inter-route optimization, we consider interchanging requests between two different routes which still satisfy the feasibility constraints. Because of the precedence constraint, we must interchange two requests, instead of two stops, from two different routes since each customer has to be serviced by the same vehicle (see Figure 3.2). In this way, the selection criteria of requests for inter-route interchanges will be quite important. Obviously, the easiest way is that, for each request of each route, we try to “swap” it with each request of another route. For example, in Figure 3.2, for each $REQUEST(i)$ of $ROUTE(r_1)$, we try to swap it with every $REQUEST(j)$ of $ROUTE(r_2)$, while satisfying the feasibility constraints and improving the given objective functions. In practice, this approach is not very useful because of the high computational cost that is involved. So, further work is still required for seeking better algorithms for the inter-route optimization. One way to reduce the amount of computation is to adjust the selection criteria. In our algorithm, we only choose the requests with greatest excess travel time. In this way, we

can attempt to minimize customer inconvenience while retaining the productivity level achieved by intra-route post-optimization.

Figure 3.2: Inter-route optimization

Our main motivation at this stage is to optimize the service quality of the initial routes by minimizing customer inconvenience while maintaining vehicle productivity, as measured by the chosen primary objective function.

3.2 Intra-route Optimization Heuristic

Now, we will describe in detail a procedure for the intra-route optimization. It is based on *k*-opt and Or-opt methods outlined in Chapter 1. About thirty years ago, Croes[58] and Lin[65] introduced the notion of *k*-interchange to improve solutions of the travelling salesman problem (Croes for $k=2$ and Lin for $k=3$). Though several other papers that examine issues related to the application of this method have since been written, this method has been proven the most powerful way to improve TSP routes. It has also been applied with considerable success to other classes of problems.

The *k*-interchange is a substitution of *k* arcs of a route with *k* other arcs. A route is *k*-opt if any additional *k*-interchange will fail to produce a shorter route. Usually only the cases where $k=2$ and $k=3$ are considered because the computational effort to verify *k*-

optimality becomes considerable as the number of stops increases when $k \geq 3$. The 2-opt algorithm searches for a favorable 2-interchange by examining all pairs of nonadjacent links (see Figure 1.4). A pair of links can be replaced uniquely by two other links that still leave the route connected but reverse the direction of the middle route segment. Clearly this interchange is feasible with respect to the precedence constraint only if no pickup-dropoff pair is serviced along the middle segment. The interchange also has to result in a shorter route in order to be considered favorable. Obviously the total number of possible 2-interchanges equals to the number of subsets of two links that can be formed from the set of N links that make up the tour. This number is equal to $\binom{N}{2}$, which implies a time complexity of $O(N^2)$ for the verification of 2-optimality. Since the DARPTW feasibility constraints have to be satisfied, a straightforward feasibility testing for a single interchange would require examining, for each stop within the middle segment, the time window, the vehicle capacity and the location of the corresponding pickup or dropoff stop. This takes at most $O(N)$ computation. In total, $O(N^3)$ time is required to verify 2-optimality and route feasibility constraints.

In the case $k=3$, three nonadjacent links may be replaced in four ways by three other links that maintain route connectivity (see Figure 3.3). The first interchange in Figure 3.3a preserves the direction of traversal of all segments but exchanges the order in which the two middle segments are visited. Interchange in Figure 3.3d reverses the direction of both middle segments. Interchanges in Figure 3.3b and 3.3c reverse the direction of one middle segment, while preserving the other's direction. Considering case 3.3a, the precedence constraint will hold if there is no pickup point in the first middle segment that matches a dropoff point appearing in the second middle segment. Again, the straightforward feasibility testing for a single interchange would require at most $O(N)$ time. Since the total number of possible 3-interchanges is proportional to the number of subsets of three links that can be formed from the set of N links that make up the tour, we get that this number equals to $\binom{N}{3}$, which implies a time complexity of $O(N^4)$ for the verification of

3-optimality and the route feasibility constraints. Theoretically we can extend this approach to a fixed k -opt algorithm for $k > 3$. More substantial research is required to find out whether a variable k -interchange algorithm can be developed for the DARPTW, in which the value of k is a variable integer ($k=2,3,\dots,N$).

Figure 3.3: All possible 3-interchanges

As we mentioned above, it takes $O(N^4)$ to verify 3-opt solution if we consider the feasibility constraints. Obviously the computational requirement can become prohibitive if the number of stops increases. Proposals have been made to take only a subset of all possible 3-interchanges into account. [Or 76] proposes to restrict attention to those 3-interchanges in which a string of one, two or three consecutive stops (a path) is relocated between two others. An Or-exchange is depicted in Figure 1.5. The path $/j, \dots, i/$ is relocated between k and $k+1$. Note that the direction of no path is reversed in this case and this fact is important when dealing with time windows. Later we will prove that there are only $O(N^2)$ interchanges of this kind.

There are two possibilities for relocating the path $/j, \dots, i/$. Within the current route we can relocate it earlier, *backward relocation* or later, *forward relocation* (see Figure 3.4 and 3.5). The cases of backward relocation ($j \leq i$) and forward relocation ($j \geq i$) will be handled separately. [Savelsbergh 85] and [Savelsbergh 88] propose a local search procedure on Or-interchange to do the route improvement in constant time $O(1)$, even when we consider the feasibility constraints. In this way, we only need $O(N^2)$ to verify Or-opt. Reducing the time complexity to $O(N^2)$ is a very big improvement, comparing the normal computational effort $O(N^4)$ for verifying 3-opt. This is why we use Or-opt method to do the intra-route optimization.

[Solomon 88b] presents a method based on k-opt to improve the initial feasible routes. In the paper, service time windows and customers precedence constraints are considered as feasibility constraints. It does not consider the vehicle capacity constraint. Since the computational effort to verify k-opt is very high, a pre-processing procedure has been used to streamline the feasibility checking. The method successfully solves the problem of customers precedence checking. At the pre-processing stage, a stop precedence array $NP(i)$ is being calculated (for $k=2$, i represents a customer), and a segment predecessor array $SP(i,j)$ is being calculated (for $k=3$, i, j represent customers). Since there are four ways to do the 3-interchanges, feasibility checking is difficult, especially

Figure 3.4: Backward relocation for Or-interchange

Figure 3.5: Forward relocation for Or-interchange

when the interchange reverse the middle segment(s). For the orientation preserving interchanges (Figure 3.3a), two parameters, *push forward shift* (PFS) and *push backward shift* (PBS), are defined and being used to check the service time windows at each customer site. In conclusion, it is claimed that PFS and PBS concept can lead to substantial reductions in the number of customers being examined for time feasibility. Since the paper does not discuss the way to check the time feasibility when the interchanges reverse the middle segment(s), we claim that the paper only efficiently solves the problem of precedence constraint checking. Within our intra-route optimization heuristics, we will

use a similar way to do the precedence checking and we also use PFS and PBS to check the time windows.

One advantage of using the Or-opt scheme for a route improvement is that Or-interchange will preserve the orientation of all segments which makes the feasibility constraints checking easier. Another advantage is its low computational complexity (only $O(N^2)$). Before the discussion of our intra-route optimization heuristic, we introduce the notation to be used throughout the reminder of this paper. Let

e_i : the earliest arrival time at stop i

l_i : the latest arrival time at stop i

S_i : the service time (handling time) at stop i

A_i : the actual arrival time at stop i

D_i : the actual departure time at stop i

W_i : the vehicle waiting time at stop i

t_{ij} : the direct travel time from stop i to stop j

Q : the maximum vehicle capacity

At each stop, the service time window will be $[e_i, l_i]$, where $e_i \leq D_i \leq l_i$ holds. Usually, it is assumed that service time S_i equals to 0 in order to simplify the problem. Within this thesis, we consider the handling time. In fact, the neglecting of service time S_i does influence the checking conditions for some objective functions.

The feasibility constraints we consider here include the service time windows at each stop which cannot be violated, the customer precedence where a customer must be picked up first before being dropped off, and the maximum vehicle capacity where the vehicle cannot be overloaded.

Three main objective functions are considered as follows to improve the initial routes:

(1) Minimize the actual total travel time

- (2) Minimize the total completion time
- (3) Minimize the partial completion time & actual total travel time

In the future, we may consider the following two extra objective functions:

- (4) Minimize the loaded vehicle waiting time
- (5) Minimize the passenger's excess travel time

For any objective function chosen, we use the value of vehicle productivity rate (VPR) to judge the quality of the improved route, where

$$\text{Vehicle Productivity Rate (VPR)} = \frac{\text{number of people transported}}{\text{travel time of vehicle} + \text{handling time}}$$

Here the people transported includes wheelchair people, ambulatory people and their accompanying people.

As we discussed above, there are two cases for Or-interchange scheme, the backward relocation where we relocate the path $/j, \dots, i/$ earlier (see Figure 3.4) and forward relocation where we relocate the path later (see Figure 3.5). Based on these two cases, our intra-route optimization heuristics is divided into two parts, backward search and forward search.

3.2.1 Or-opt Backward Search

For backward search, we use backward relocation method to do the Or-interchange. This is because we start at the destination depot N_1 (see Figure 3.4), choosing stops along the route backward to the origin depot 0. Based on the definition of Or-interchange, we try to relocate a path $/j, \dots, i/$ backward between $k, k+1$. The number of stops within the path is a constant number (one, two or three). Then we will check the feasibility constraints and the objective functions to see if the new route is feasible and being improved.

After the Or-interchange, we can find that the segment $\{k+1, \dots, N_1\}$ might be infeasible because of the insertion of $\{j, \dots, i\}$. Since we do not want to use a straightforward way to check the feasibility constraints of this segment, a local search method is being used here so that we only need $O(1)$ to do the feasibility checking. Before we introduce this local search method, several parameters are defined as follows:

PROFIT: the gain when going directly from $j-1$ to $i+1$

$$\text{PROFIT} = A_{i+1} - (D_{j-1} + t_{j-1,i+1})$$

LOSS: the loss if we use $\{j, \dots, i\}$ as intermediate when going from k to $k+1$

$$\begin{aligned} \text{LOSS} &= A_{k+1}^{\text{new}} - A_{k+1}^{\text{old}}, \text{ where } A_j^{\text{new}} = D_k + t_{k,j} \\ A_{j+1}^{\text{new}} &= \max(A_j^{\text{new}}, e_j) + S_j + t_{j,j+1} \\ A_i^{\text{new}} &= \max(A_{j+1}^{\text{new}}, e_{j+1}) + S_{j+1} + t_{j+1,i} \\ A_{k+1}^{\text{new}} &= \max(A_i^{\text{new}}, e_i) + S_i + t_{i,k+1} \quad (\text{if } i=j+2) \end{aligned}$$

Since the maximum number of stops within the path $\{j, \dots, i\}$ are three, so we need $O(1)$ to calculate LOSS.

TWT $^{(k+1,j-1)}$: the total waiting time on the segment $\{k+1, \dots, j-1\}$

$$\text{TWT}^{(k+1,j-1)} = \sum_{x=k+1}^{j-1} w_x$$

When we move backward along the route from $j-1$ to search for stop $k+1$ to do the Or-interchange, we sum the waiting time at each stop at the same time until stop $k+1$ is found. In this way, we still only need $O(1)$ to get the value of $\text{TWT}^{(k+1,j-1)}$.

PFS $^{(k+1,j-1)}$: possible forward shift in time of the departure time at $k+1$ causing no violation of the time window constraints on the segment $\{k+1, \dots, j-1\}$. We use the following recursive function to calculate its value:

initialize: $PFS(j-2,j-1) = W_{j-1} + l_{j-1} - D_{j-1}$

recurse: $PFS(k+1,j-1) = W_{k+1} + \text{Min}\{l_{k+1} - D_{k+1}, PFS(k+2,j-1)\}$

Using the same argument as for the calculation of the value $TWT(k+1,j-1)$ we can get $PFS(k+1,j-1)$ in $O(1)$.

PBS(i+1,N1): possible backward shift in time of the departure time at $i+1$ causing no extra waiting time on the segment $\{i+1, \dots, N_1\}$. We use the following recursive function to calculate its value:

initialize: $PBS(N_1-1,N_1) = D_{N_1-1} - S_{N_1-1} - e_{N_1-1}$

recurse: $PBS(i+1,N_1) = \text{Min}\{D_{i+1} - S_{i+1} - e_{i+1}, PBS(i+2,N_1)\}$

When we start from the destination depot N_1 backward to search for $i+1$, we calculate PBS at the same time. So it takes $O(1)$ time to calculate.

The values of the parameters above can all be obtained in $O(1)$ time when we do the backward search. With these parameters, we can check the feasibility constraints easily in constant time. Here we embed our time window checking into the objective function's checking to simplify the problem. Based on Figure 3.4, let us consider the three main objective functions for each vehicle:

(1) Minimize the actual total travel time

Since we use $t_{i,j}$ to represent the direct travel time from i to j , the following condition must hold after Or-interchange:

$$t_{k,j} + t_{i,k+1} + t_{j-1,i+1} < t_{i,i+1} + t_{j-1,j} + t_{k,k+1}$$

For the time window constraint, obviously we can find that the segment $\{k+1, \dots, j-1\}$ is feasible only if $\text{LOSS} \leq PFS(k+1,j-1)$ holds. If $\text{PROFIT} < \text{LOSS} - TWT(k+1,j-1)$, that means we arrive at $i+1$ late, so the segment $\{i+1, \dots, N_1\}$ maybe infeasible. In this case, we have to confirm that $\text{LOSS} - TWT(k+1,j-1) - \text{PROFIT} \leq PBS(i+1,N_1)$. Getting them together, we

get the conditions which have to be satisfied for this objective function and related time window constraints:

$$(a) \ t_{k,j} + t_{i,k+1} + t_{j-1,i+1} < t_{i,i+1} + t_{j-1,j} + t_{k,k+1}$$

$$(b) \ \text{LOSS} \leq \text{PFS}(k+1,j-1)$$

$$(c) \ \text{If } \text{PROFIT} < \text{LOSS} - \text{TWT}^{(k+1,j-1)} \text{ then } \text{LOSS} - \text{TWT}^{(k+1,j-1)} - \text{PROFIT} \leq \text{PFS}(i+1,N_1)$$

(2) Minimize the total completion time

The vehicle total completion time includes the total travel time and waiting time. After the Or-interchange, it is not necessary to check the condition

$$t_{k,j} + t_{i,k+1} + t_{j-1,i+1} < t_{i,i+1} + t_{j-1,j} + t_{k,k+1} \ ,$$

but we prefer that this condition holds because our interchange reduces the travel time. Now we want to arrive at the destination depot earlier, so $\text{PROFIT} > \text{LOSS} - \text{TWT}^{(k+1,j-1)}$ must hold. Then we check if the highest index stop k with $D_k = e_k + S_k$ is in the segment $\{i+1, \dots, N_1-1\}$ or not. If "yes", it means the time savings of our Or-interchange cannot be carried through to the destination depot N_1 . In this case, we cannot reduce the total completion time. Otherwise, we can save at most $\text{PBS}(i+1,N_1)$ units of completion time. We also need to check if $\text{LOSS} \leq \text{PFS}(k+1,j-1)$ holds or not for the time feasibility of segment $\{k+1, \dots, j-1\}$. So the conditions which have to be satisfied for this objective function and time window constraints to do Or-interchange are as follows:

$$(a) \ t_{k,j} + t_{i,k+1} + t_{j-1,i+1} < t_{i,i+1} + t_{j-1,j} + t_{k,k+1} \text{ (may or may not)}$$

$$(b) \ \text{PROFIT} > \text{LOSS} - \text{TWT}^{(k+1,j-1)}$$

$$(c) \ \text{LOSS} \leq \text{PFS}(k+1,j-1)$$

(d) The highest index stop k with $D_k = e_k + S_k$ is not in the segment $\{i+1, \dots, N_1-1\}$ that is, $\text{PBS}(i+1,N_1) \neq ()$.

(3) Minimize the partial completion time and actual total travel time

In this case, we loosen the conditions of (2) to do the Or-interchange. Obviously the condition

$$t_{k,j} + t_{i,k+1} + t_{j-1,i+1} < t_{i,i+1} + t_{j-1,j} + t_{k,k+1}$$

must hold. To reduce the completion time, we have to arrive at stop $i+1$ earlier, so $\text{PROFIT} > \text{LOSS} - \text{TWT}^{(k+1,j-1)}$ holds. Here we do not care if the time savings caused by our Or-interchange can be carried through to the destination depot N_1 or not, because we only need to reduce the partial completion time (but actual total travel time has to be reduced). Same as (1), $\text{LOSS} \leq \text{PFS}^{(k+1,j-1)}$ must hold for the time feasibility of segment $\{k+1, \dots, j-1\}$. So the conditions which have to be satisfied for this objective function and related time window constraints are as follows:

$$(a) \ t_{k,j} + t_{i,k+1} + t_{j-1,i+1} < t_{i,i+1} + t_{j-1,j} + t_{k,k+1}$$

$$(b) \ \text{PROFIT} > \text{LOSS} - \text{TWT}^{(k+1,j-1)}$$

$$(c) \ \text{LOSS} \leq \text{PFS}^{(k+1,j-1)}$$

Now let us consider the precedence constraint. Each request corresponds to two stops, pickup and dropoff stops, and pickup stop must precede dropoff stop. Within each $\text{ROUTE}(r)$, an array $\text{PRECED}(r,u)$ is being defined to represent the precedence relationship between any two stops u and v in the route:

$$\text{PRECED}(r,u) = \begin{cases} -v & \text{if } u \text{ precedes } v \\ v & \text{if } v \text{ precedes } u \\ 0 & \text{if } u \text{ has no precedence relation with } v \end{cases}$$

When backward search is used, we can find that the precedence constraint will be violated only when there exists some requests for which the pickup stops are on the segment $\{k+1, \dots, j-1\}$, but dropoff stops are on the segment $\{j, \dots, i\}$. So we can use the following loop to check the precedence constraint for a given $\text{ROUTE}(r)$:

precedence_check()

```

{
    for( u=j; u≤i; u++ )
        if( PRECED(r,u) ≥ k+1 && PRECED(r,u) ≤ j-1 )
            output ("Precedence constraint violated");
}

```

Obviously $O(1)$ is needed because the segment $\{j, \dots, i\}$ contains a constant number of stops.

As to the capacity constraint checking, we use the modified method mentioned in [Savelsbergh 88]. Normally we consider two kinds of handicapped people, wheelchair and ambulatory, and we also need to consider their accompanying people. Since the special vehicle may only take certain amount of wheelchair and ambulatory handicapped people on board, the capacity constraint seems very important in real life. Based on the current market survey done by the author, service agencies use three types of special vehicles, a vehicle which can only take either wheelchair people or ambulatory people but not both and a vehicle which can take both kinds of people. For the third type of vehicle, the seats on board can be folded (see Figure 3.6a, 3.6b and 3.6c) if a wheelchair person is on board. In this case, we use a conversion factor λ to get the number of seats that the wheelchair person will occupy, e.g., $\lambda=2$ means one wheelchair person takes two seats which is the double space of what one ambulatory person will take.

In order to give the checking conditions for the capacity constraint, we have to consider the vehicle type. For the first and second types of vehicle, it will be easier. Suppose the total capacity of vehicle Q is the number of seats on board (for wheelchair-only vehicle, we think one wheelchair person takes one "seat"). Given a feasible route $\{0, 1, \dots, N_1\}$, let a and b be two stops on the route such that stop a precedes stop b . Let $PICK(a,b)$ and $DROP(a,b)$ be the set of pickup and dropoff stops on the subroute $\{a, \dots, b\}$, respectively. Let q_x be the number of people to be picked up or dropped off at stop x , where x is any stop on the subroute $\{a, \dots, b\}$. Let $P(a,b)$ be the set of handicapped

Figure 3.6a: Vehicle with only ambulatory handicapped people

Figure 3.6b: Vehicle with only wheelchair handicapped people

Figure 3.6c: Vehicle with both types of handicapped people

people (wheelchair or ambulatory) which the vehicle has to pick up and $D(a,b)$ be the set of handicapped people which the vehicle has to drop off on the same subroute. Then we get

$$P(a,b) = \sum_{x \in \text{PICK}(a,b)} q_x \quad \text{and} \quad D(a,b) = \sum_{x \in \text{DROP}(a,b)} q_x .$$

The capacity constraint will hold if and only if

$$0 \leq P(0,x) - D(0,x) \leq Q \quad \text{for } x = 0, \dots, N_1.$$

For backward Or-interchange, where the $\{j, \dots, i\}$ is relocated between k and $k+1$ (see Figure 3.4), we find that

$$\begin{aligned} P^{\text{new}}(0,x) &= P(0,x) & \text{for } 0 \leq x \leq k, i+1 \leq x \leq N_1 \\ D^{\text{new}}(0,x) &= D(0,x) & \text{for } 0 \leq x \leq k, i+1 \leq x \leq N_1 \\ P^{\text{new}}(0,x) &= P(0,x) + P(j,i) & \text{for } k+1 \leq x \leq j-1 \\ D^{\text{new}}(0,x) &= D(0,x) + D(j,i) & \text{for } k+1 \leq x \leq j-1 \\ P^{\text{new}}(0,x) &= P(0,x) - P(k+1,j-1) & \text{for } j \leq x \leq i \\ D^{\text{new}}(0,x) &= D(0,x) - D(k+1,j-1) & \text{for } j \leq x \leq i \end{aligned}$$

Based on the above conditions, we can get the following equivalent conditions after we make the proper substitutions:

$$\begin{aligned} \text{Min}_{k+1 \leq x \leq j-1} \{ P^{\text{new}}(0,x) - D^{\text{new}}(0,x) \} &\geq 0 \\ \text{Max}_{k+1 \leq x \leq j-1} \{ P^{\text{new}}(0,x) - D^{\text{new}}(0,x) \} &\leq Q \\ \text{Min}_{j \leq x \leq i} \{ P^{\text{new}}(0,x) - D^{\text{new}}(0,x) \} &\geq 0 \\ \text{Max}_{j \leq x \leq i} \{ P^{\text{new}}(0,x) - D^{\text{new}}(0,x) \} &\leq Q \end{aligned}$$

We rewrite the conditions above as follows:

$$\begin{aligned} \text{Min}_{k+1 \leq x \leq j-1} \{ P(0,x) - D(0,x) \} + \{ P(j,i) - D(j,i) \} &\geq 0 \\ \text{Max}_{k+1 \leq x \leq j-1} \{ P(0,x) - D(0,x) \} + \{ P(j,i) - D(j,i) \} &\leq Q \\ \text{Min}_{j \leq x \leq i} \{ P(0,x) - D(0,x) \} - \{ P(k+1,j-1) - D(k+1,j-1) \} &\geq 0 \\ \text{Max}_{j \leq x \leq i} \{ P(0,x) - D(0,x) \} - \{ P(k+1,j-1) - D(k+1,j-1) \} &\leq Q \end{aligned}$$

Obviously we only need $O(1)$ to calculate $\text{Min (or Max)}_{k+1 \leq x \leq j-1} \{P(0,x)-D(0,x)\}$ when starting from N_1 in the backward search. Specifically, for each $\text{STOP}(x)$, $k+1 \leq x \leq j-1$, we calculate $P(0,x) - D(0,x)$ first. Then we only keep the minimum (or maximum) value. $O(1)$ time is enough to calculate $P(k+1,j-1) - D(k+1,j-1)$. Since the length of segment $\{j, \dots, i\}$ is fixed, we need only constant time to calculate $P(j,i)-D(j,i)$.

Consider now the third type of vehicle which can take both wheelchair and ambulatory people. We can use the above conditions for wheelchair and ambulatory people separately where Q is the total number of seats in the vehicle. Obviously we need another condition, that is, at any time,

$$\lambda * Q_w + Q_d \leq Q$$

where λ is the conversion factor, Q_w and Q_a is the number of wheelchair people and ambulatory people on board, respectively. By using the methods above, we check successfully in time $O(1)$ the vehicle capacity constraint for two types of handicapped people. For a given route r , our backward search heuristic algorithm is shown as follows:

backward_search(int r)

{

```
control = TRUE;
```

```
while( control )
```

1

```
control = FALSE;
```

```
for( i=N1-1; i>2; i-- )          ...O(N)
```

```
for( j=i; j<=i-2; j-- )          ...O(1)
```

```
for( k=j-2; k<=0; k++ )          ...O(N)
```

1

break three edges $[i,i+1]$, $[j-1,j]$ and $[k, k+1]$;

add three new edges $[j-1, i+1]$, $[k+1, i]$ and $[k, j]$;

calculate related parameters' values, i.e., PROFIT, LOSS, etc.

check objective function and time windows constraints; ---(X!)

```
check precedence constraint; ---O(1)
```

```

        check vehicle capacity constraint;                                ---O(1)
        if( this is a favorable interchange )
        {
            do the interchange;
            update all data structures;                                ---O(N)
            control = TRUE;
        }
    }
}

```

The overall computational complexity in the worst case for our backward search algorithm is $O(N^3)$ per route.

3.2.2 Or-opt Forward Search

For forward search, we use forward relocation method to do Or-interchange. Starting from the origin depot 0, we choose stops along the route forward to the destination depot N_1 . Based on the same definition of Or-interchange we try to relocate the path $\{i, \dots, j\}$ forward between k and $k+1$. The number of stops within the path is still a constant number (one, two or three). Referring to Figure 3.5, we use the same objective functions and use the same way to check the feasibility constraints to get feasible and improved routes.

Theoretically, we can use either backward or forward search to get Or-opt. The drawback of using forward search is that we have to use $O(N)$ instead of $O(1)$ to check the feasibility constraint. As an academic research exercise, we try both methods here.

After we relocate $\{i, \dots, j\}$ forward between k and $k+1$, we might push backward some units of time for segment $\{j+1, \dots, k\}$ and push forward some units of time for segment $\{k+1, \dots, N_1\}$. So we have to check the feasibility constraints on these two segments. Similarly to the analysis of backward search, we embed our time window checking into the objective function's checking. The same notation is used here to get conditions which have to be satisfied for three objective functions as follows:

PROFIT: the gain when going directly from $i-1$ to $j+1$

$$\text{PROFIT} = A_{j+1} - (D_{i-1} + t_{i-1,j+1})$$

LOSS: the loss if we use $\{i, \dots, j\}$ as intermediate when going from k to $k+1$

$$\begin{aligned} \text{LOSS} &= A_{k+1}^{\text{new}} - A_{k+1}^{\text{old}}, \quad \text{where } A_i^{\text{new}} = D_k + t_{k,i} \\ A_{i+1}^{\text{new}} &= \max(A_i^{\text{new}}, e_i) + S_i + t_{i,i+1} \\ A_j^{\text{new}} &= \max(A_{i+1}^{\text{new}}, e_{i+1}) + S_{i+1} + t_{i+1,j} \\ A_{k+1}^{\text{new}} &= \max(A_j^{\text{new}}, e_j) + S_j + t_{j,k+1} \quad (\text{if } j=i+2) \end{aligned}$$

PFS($k+1, N_1$): possible forward shift in time of the departure time at $k+1$ causing no violation of the time window constraints on the segment $\{k+1, \dots, N_1\}$.

We use the following recursive function to calculate its value:

$$\begin{aligned} \text{initialize:} \quad & \text{PFS}(N_1-1, N_1) = W_{N_1-1} + I_{N_1-1} - D_{N_1-1} \\ \text{recurse:} \quad & \text{PFS}(k+1, N_1) = W_{k+1} + \text{Min} \{ I_{k+1} - D_{k+1}, \text{PFS}(k+2, N_1) \} \end{aligned}$$

PBS($j+1, k$): possible backward shift in time of the departure time at $j+1$ causing no extra waiting time on the segment $\{j+1, \dots, k\}$. We also use a recursive function to calculate its value $\text{PBS}(j+1, k) = \text{Min}_{j+1 \leq x \leq k} \{ D_x - S_x - e_x \}$ as follows:

$$\begin{aligned} \text{initialize:} \quad & \text{PBS}(j+1, j+2) = D_{j+1} - S_{j+1} - e_{j+1} \\ \text{recurse:} \quad & \text{PBS}(j+1, k) = \text{Min} \{ D_k - S_k - e_k, \text{PBS}(j+1, k-1) \} \end{aligned}$$

Here we have to use $O(N)$ steps to get the value $\text{PFS}(k+1, N_1)$. All the other parameters' values can be obtained in $O(1)$ time because we can get them when we travel along the route forward. Three objective functions are considered as in the backward search mentioned before. They are briefly described as follows:

(1) Minimize the actual total travel time

$$(a) \quad t_{i-1,j+1} + t_{k,i} + t_{j,k+1} < t_{i-1,i} + t_{j,i+1} + t_{k,k+1}$$

(b) If $\text{Min} \{ \text{PROFIT}, \text{PBS}(j+1,k) \} \leq \text{LOSS}$ then $\text{LOSS} - \text{Min} \{ \text{PROFIT}, \text{PBS}(j+1,k) \} \leq \text{PFS}(k+1,N_1)$

(c) Check the feasibility of segment $\{ i, \dots, j \}$

(d) We may incur more waiting time if $\text{PROFIT} > \text{PBS}(j+1,k)$

(2) Minimize the total completion time

(a) $t_{i-1,j+1} + t_{k,i} + t_{j,k+1} < t_{i-1,i} + t_{j,j+1} + t_{k,k+1}$ (may or may not)

(b) Check the feasibility of segment $\{ i, \dots, j \}$

(c) If $\text{Min} \{ \text{PROFIT}, \text{PBS}(j+1,k) \} \leq \text{LOSS}$ then $\text{LOSS} - \text{Min} \{ \text{PROFIT}, \text{PBS}(j+1,k) \} \leq \text{PFS}(k+1,N_1)$, or

(d) If $\text{Min} \{ \text{PROFIT}, \text{PBS}(j+1,k) \} \geq \text{LOSS}$ then $\text{Min} \{ \text{PROFIT}, \text{PBS}(j+1,k) \} - \text{LOSS} \leq \text{PBS}(k+1,N_1)$

(3) Minimize the partial completion time and actual total travel time

(a) $t_{i-1,j+1} + t_{k,i} + t_{j,k+1} < t_{i-1,i} + t_{j,j+1} + t_{k,k+1}$

(b) Check the feasibility of segment $\{ i, \dots, j \}$

(c) If $\text{Min} \{ \text{PROFIT}, \text{PBS}(j+1,k) \} \leq \text{LOSS}$ then $\text{LOSS} - \text{Min} \{ \text{PROFIT}, \text{PBS}(j+1,k) \} \leq \text{PFS}(k+1,N_1)$

As for the precedence constraint, we can find that it will be violated only when there exist some requests for which the pickup stops are on the segment $\{ i, \dots, j \}$, but dropoff stops are on the segment $\{ j+1, \dots, k \}$. So we can use the same data structure and we can loop as in the backward search to check the precedence constraint for a given ROUTE(r):

precedence_check()

```
{
    for( u=j+1; u≤k; u++ )
        if( PRECED(r,u) ≥ i && PRECED(r,u) ≤ j )
            output ("Precedence constraint violated");
}
```

Obviously $O(1)$ is enough because we can do the loop at the same time as the forward search.

As to the capacity constraint checking, we use the same method and notation as in the backward search to solve this problem. The only difference is the stop segments. For the type of vehicle which can only take either wheelchair or ambulatory people, we get:

$$\begin{aligned}
 P^{new}(0,x) &= P(0,x) & \text{for } 0 \leq x \leq i-1, k+1 \leq x \leq N_1 \\
 D^{new}(0,x) &= D(0,x) & \text{for } 0 \leq x \leq i-1, k+1 \leq x \leq N_1 \\
 P^{new}(0,x) &= P(0,x) - P(i,j) & \text{for } j+1 \leq x \leq k \\
 D^{new}(0,x) &= D(0,x) - D(i,j) & \text{for } j+1 \leq x \leq k \\
 P^{new}(0,x) &= P(0,x) + P(j+1,k) & \text{for } i \leq x \leq j \\
 D^{new}(0,x) &= D(0,x) + D(j+1,k) & \text{for } i \leq x \leq j
 \end{aligned}$$

After we make the proper substitutions, we obtain the following conditions:

$$\begin{aligned}
 \text{Min}_{j+1 \leq x \leq k} \{ P^{new}(0,x) - D^{new}(0,x) \} &\geq 0 \\
 \text{Max}_{j+1 \leq x \leq k} \{ P^{new}(0,x) - D^{new}(0,x) \} &\leq Q \\
 \text{Min}_{i \leq x \leq j} \{ P^{new}(0,x) - D^{new}(0,x) \} &\geq 0 \\
 \text{Max}_{i \leq x \leq j} \{ P^{new}(0,x) - D^{new}(0,x) \} &\leq Q
 \end{aligned}$$

We rewrite the conditions above as follows:

$$\begin{aligned}
 \text{Min}_{j+1 \leq x \leq k} \{ P(0,x) - D(0,x) \} - \{ P(i,j) - D(i,j) \} &\geq 0 \\
 \text{Max}_{j+1 \leq x \leq k} \{ P(0,x) - D(0,x) \} - \{ P(i,j) - D(i,j) \} &\leq Q \\
 \text{Min}_{i \leq x \leq j} \{ P(0,x) - D(0,x) \} + \{ P(j+1,k) - D(j+1,k) \} &\geq 0 \\
 \text{Max}_{i \leq x \leq j} \{ P(0,x) - D(0,x) \} + \{ P(j+1,k) - D(j+1,k) \} &\leq Q
 \end{aligned}$$

For the third type of vehicle which can take both wheelchair and ambulatory people, we add another condition:

$$\lambda * Q_w + Q_a \leq Q.$$

Here again, λ is the conversion factor, Q_w and Q_a is the number of wheelchair people and ambulatory people on board, respectively. As in the backward search, we can easily prove that the parameters' values can be obtained in constant time. So we only need $O(1)$ to check the vehicle capacity constraint. For a given route r , the following algorithm is our forward search heuristics:

```

forward_search( int r )
{
    control = TRUE;
    while( control )
    {
        control = FALSE;
        for( i=1; i≤N1-2; i++ )           ---O(N)
            for( j=i; j≤i+2; j++ )         ---O(1)
                for( k=j+1; k≤N1-1; k++ ) ---O(N)
                {
                    break three edges [i-1,i], [j,j+1] and [k, k+1];
                    add three new edges [i-1, j+1], [k,i] and [j,k+1];
                    calculate related parameters' value;
                    check objective function and time windows;           ---O(N)
                    check precedence constraint;                          ---O(1)
                    check vehicle capacity constraint;                    ---O(1)
                    if( this is a favorable interchange )
                    {
                        do the interchange;
                        update all data structures;                       ---O(N)
                        control = TRUE;
                    }
                }
    }
}

```

The overall computational complexity in the worst case for our forward search is thus $O(N^4)$ per route.

3.3 Inter-route Optimization Heuristics

Inter-route optimization is the second stage of our post-optimization procedure for DARPTW. After we finish the first stage, intra-route optimization, each separate route has been improved to Or-opt according to different objective functions. Since the initial routes are highly dependent on the way the requests are clustered and the type of vehicle chosen, it is possible that there exist some requests on different routes which can be swapped to further improve the routes which are already Or-opt. [Solomon 88b] mentions that techniques similar to those of intra-route optimization can be easily embedded within an overall improvement procedure for the VRSPTW where exchanges between routes are also examined. But he does not show how to apply the "similar techniques" to the inter-route optimization (actually, there are no papers published so far mentioning how to do it). Since each customer has to be picked up and dropped off by the same vehicle, our inter-route optimization heuristics is based on the interchanges of two requests, instead of two stops, on different routes. Usually at this stage, customer inconvenience, which is evaluated as a function of the deviation from the desired service time and the excess travel time, is attempted to be reduced, while the value of primary objective functions cannot deteriorate for each route involved at the same time.

The easiest way to do the inter-route interchange is that for each REQUEST(*i*) on ROUTE(*r*₁), we try to find any other REQUEST(*j*) on ROUTE(*r*₂) so that they can be interchanged, while satisfying all the feasibility constraints and retaining the primary objective functions established in stage 1, intra-route optimization. Referring to Figure 3.2, suppose we have *m* routes, each route containing maximum of *n* stops (*n*/2 requests). At the beginning, all the requests are unmarked. Our heuristics for the inter-route optimization is as follows:

```

inter_route_opt1()
{
    for( r1=1; r1≤m-1; r1++ )

```



```

{
    for( i=1; i≤nr1/2; i++ )
    {
        get unmarked REQUEST(i);
        Obj_improved = 0;
        for( r2=r1+1; r2≤m; r2++ )
        {
            for( j=1; j≤nr2/2; j++ )
            {
                get unmarked REQUEST(j);
                if( REQUEST(i) can be inserted into {ROUTE(r2) - REQUEST(j)} feasibly )
                {
                    if( REQUEST(j) can be inserted into {ROUTE(r1) - REQUEST(i)}
                        feasibly )
                    {
                        if( primary objective functions improved && value improved
                            > Obj_improved )
                        {
                            update Obj_improved;
                            store the route number r2 and request number j;
                            store related stop positions;
                        }
                    }
                }
            }
        }
    }
    if( Obj_improved > 0 )
    {
        interchange two requests ( swap REQUEST(i) and REQUEST(j) );
        update all related data structures;
        mark REQUEST(i) and REQUEST(j);
    }
}

```

---O(m)
---O(n)
---O(m)
---O(n)
---O(n³)
---O(n³)
---O(n)

where n_{r1} and n_{r2} is the number of stops of $ROUTE(r_1)$ and $ROUTE(r_2)$ ($n_{r1}, n_{r2} \leq n$), respectively. $Obj_improved$ is the latest maximum improved value of chosen objective function (we initialize $Obj_improved$ to 0). Obviously the computational complexity for the heuristics above is $O(m * n * m * n * n^3)$, that is, $O(m^2n^5)$.

We can see that the heuristics above is very time consuming. So this is not a practical optimization algorithm. In fact, when we deal with the reduction of customer inconvenience, excess travel time and deviation from desired service time are the most important factors. For any $REQUEST(i)$, we use the following formula to obtain these two values:

$$\begin{aligned} \text{Excess_Travel_Time}(i) &= A_{\text{dropoff}} - D_{\text{pickup}} - t_{\text{pickup,dropoff}} \\ \text{Deviation}(i) &= \begin{cases} A_{\text{pickup}} - \text{DST}(i) & \text{if request_type}(i) = 'D' \\ \text{DST}(i) - S_{\text{dropoff}} - A_{\text{dropoff}} & \text{if request_type}(i) = 'A' \end{cases} \end{aligned}$$

where $\text{DST}(i)$ is the customer's desired service time, S_{dropoff} is the service time at $\text{STOP}(\text{dropoff})$, A_{pickup} and A_{dropoff} are the actual arrival time at $\text{STOP}(\text{pickup})$ and $\text{STOP}(\text{dropoff})$, respectively. The $\text{Deviation}(i)$ must be greater than or equal to zero based on our definitions, or the service agency will be penalized. It is because we cannot pick up a customer earlier than his/her $\text{DST}(i)$, if the request type is "Departure" required, and we cannot drop off a customer later than his/her $\text{DST}(i)$, if the request type is "Arrival" required. For any $REQUEST(i)$, the function giving the customer inconvenience can be defined as follows:

$$F_REQUEST(i) = \alpha * \text{Excess_Travel_Time}(i) + \beta * \text{Deviation}(i), \quad \alpha + \beta = 1$$

In a normal situation, we might think that reducing excess travel time is preferred to reducing the deviation from the desired service time. In this case, we can set $\beta = 0$. Based on the definitions above, it is easy to get the value of inconvenience function of the whole route as follows:

$$F_ROUTE(r) = \sum_i F_REQUEST(i),$$

where the sum is taken over all requests on ROUTE(r).

Our objective at this stage is to reduce a customer's inconvenience and the route corresponding inconvenience function value. We can proceed by sorting the requests of a specific route according to their values of inconvenience function and then attempting to swap the request with the highest value of inconvenience function with a request on another route.

A new inter-route optimization heuristics is proposed below. The horizon concept is applied to get related requests for swapping because it makes sense that we swap two requests within the same horizon. We will use term "*request_horizon*" to make it different from the concept of horizon defined in Chapter 2. In order to choose the *request_horizon*, we need a time factor ϵ . The exact value can be chosen after some experimentation. Here we choose half an hour ($\epsilon = 30$ minutes) as the time factor initially. Let REQUEST(g) be the request with greatest value of $F(i)$ and time window of pickup STOP(P_g) equal to $[e_{P_g}, l_{P_g}]$. The corresponding *request_horizon* is $[e_{P_g}-30, l_{P_g}+30]$. If we cannot get any other requests where their pickup stops are within this *request_horizon*, or we find out that the number of requests within this *request_horizon* are not enough to make a decision, then we adjust the time factor ϵ , i.e., increase it to an hour (60 minutes) to get more requests for possible consideration.

For each REQUEST(k) within the same *request_horizon* of REQUEST(g), we will calculate "how close" the pair of requests is. Only the closest request will be kept for swapping with REQUEST(g) in the future. The measure of "how close" should be compatible with the primary objective function (dealing with time or distance). Here we use time in minutes as our measure. Referring to Figure 3.7, suppose we want to calculate "how close" the pair of REQUEST(g) on ROUTE(g) and REQUEST(k), on ROUTE(k) is. P_g , P_k and D_g , D_k are the pickup stops and dropoff stops of REQUEST(g) and

REQUEST(k), respectively. Meanwhile P_{g-1} , P_{g+1} and D_{g-1} , D_{g+1} are the previous and following stops of P_g and D_g . P_{k-1} , P_{k+1} and D_{k-1} , D_{k+1} are the previous and following stops of P_k and D_k , respectively. We use the following formula to measure the closeness of these two requests:

$$\begin{aligned} \text{Close} = & [t_{P_g, P_{k-1}} + t_{P_g, P_{k+1}} - t_{P_{k-1}, P_{k+1}}] + [t_{D_g, D_{k-1}} + t_{D_g, D_{k+1}} - t_{D_{k-1}, D_{k+1}}] \\ & + [t_{P_k, P_{g-1}} + t_{P_k, P_{g+1}} - t_{P_{g-1}, P_{g+1}}] + [t_{D_k, D_{g-1}} + t_{D_k, D_{g+1}} - t_{D_{g-1}, D_{g+1}}] \end{aligned}$$

In this way, we can find a REQUEST(c) with minimum value of Close, in other words, for REQUEST(g), we can find its closest request REQUEST(c).

Figure 3.7: Method to calculate "how close" the pair of requests is

Now we will use our insertion heuristics to swap REQUEST(g) and its closest request, REQUEST(c). Referring to Figure 3.8a, we first set a time factor δ for REQUEST(g), obtaining thus two time periods $[e_{P_g} - \delta, l_{P_g} + \delta]$ and $[l_{P_g} + \delta, l_{D_g} + \delta]$. Since our objective now is to swap the two requests, we try to insert P_c of REQUEST(c) into the segment of ROUTE(g) within the time period $[e_{P_g} - \delta, l_{P_g} + \delta]$ and at the same time to exclude P_g , then insert D_c to the segment of ROUTE(g) within the time period $[l_{P_g} + \delta, l_{D_g} + \delta]$ and at the same time to exclude D_g , while satisfying all the feasibility constraints of

Figure 3.8: Insertion heuristics for swapping REQUEST(g) with REQUEST(c)

ROUTE(g), and guaranteeing that the primary objective function will not deteriorate. Referring to Figure 3.8b, we use the same time factor δ used for insertion of REQUEST(c) and insert in the same way P_g and D_g of ROUTE(g) into ROUTE(c), for which REQUEST(c) is excluded. The value of time factor δ will be set depending on the number of stops within each time period, e.g., $\delta = 15$ minutes. After the swap, the decrease of

route inconvenience function value for $ROUTE(g)$ must be greater than the increase of the value for $ROUTE(c)$ so that we really decrease the total inconvenience function value for all routes.

We are now in a position to describe more formally our new inter-route optimization heuristics based on reducing the total customer inconvenience. For each step we indicate its complexity.

Assume that there are N requests in total and M routes, each route contains n requests ($n \leq N$). ϵ and δ are two time factors which are used to set up `request_horizon` and time period for swapping requests as described above. At the beginning, all the requests are unmarked. The heuristics is given as follows:

Inter_route_opt2()

```
{
    sort N requests by earliest pickup time;                ---O(NlogN)
    sort N requests by their values of F_REQUEST() in decreasing order ---O(NlogN)
    i = 1;
    while( i ≤ N/2 )
    {
        ---O(N)
        get an unmarked REQUEST(g) with the i-th value of F_REQUEST();
        get related request_horizon for REQUEST(g) based on value ε ;
        for( each REQUEST(k) NOT on ROUTE(g) within the request_horizon )
        {
            ---O(N)
            calculate the value "how close" for the pair of REQUEST(g) and
                REQUEST(k);
            keep the closest request as REQUEST(c) which is unmarked;
        }
        get the value of time factor δ ;
        if( REQUEST(c) can be inserted into {ROUTE(g) - REQUEST(g)} feasibly )
        {
            ---O(N3)
            if( REQUEST(g) can be inserted into {ROUTE(c) - REQUEST(c)}
                feasibly )
            {
                ---O(N3)
                if( ( the decrease of F_REQUEST(g) on ROUTE(g) >
```

```

        the increase of F_REQUEST(c) on ROUTE(c) ) &&
        objective function's value will not deteriorate )
    {
        do the swap;
        update related data structures;      ---O(N)
        mark REQUEST(g) and REQUEST(c);

    }
}
else
    i = i + 1;
}
else
    i = i + 1;
}
output every route (maybe updated)
statistics calculation
}

```

To insert a request into an existing route, we can use the insertion heuristics presented in Chapter 2 in which the complexity is $O(N^3)$. Therefore, the computational complexity of our heuristics, in the worst case, is $O(N * N^3) = O(N^4)$, which is better than the previous algorithm, **Inter_route_opt1()**, presented at the beginning of this section. Normally, the number of favorable swaps is expected to be very small, most of the requests in $ROUTE(c)$ cannot be inserted into $\{ ROUTE(g) - REQUEST(g) \}$.

The primary objective function can be any one we discussed before. When we check the value of customer's inconvenience function before doing the swap, the value of objective function can be checked as well. In this case we do not need extra time to check whether the values of primary objective functions have deteriorated or not.

The heuristics provided above is one of the methods to reduce customer inconvenience so that the quality of service is improved, while retaining the values of primary objective functions. As far as we know this is the first algorithm for inter-route post-optimization. It is expected to be efficient because the algorithm is based on the

insertion heuristics and its main goal is to reduce customer inconvenience, which is the combination of excess travel time and the deviation from the desired service time. Even though this algorithm still needs to be improved and tested, we hope that it can be helpful in the future for research of inter-route optimization.

Chapter 4

Computational Results

In order to test the computational performance of our Or-opt post-optimization algorithms described in Chapter 3, and to compare their performance with the results obtained from 2-opt and 3-opt, we need a set of test problems for the DARPTW in paratransit. At present, most test results published are for the TSPTW or VRSPTW. Little work has been done on the application in paratransit, especially how to optimize the initial routes, while satisfying the time window constraints, precedence constraints and vehicle capacity constraints.

In this chapter, we present computational results obtained from 6 test cases chosen from a city in Ontario, Canada. The results include the initial routes based on our insertion heuristics, the 2-opt routes, the 3-opt routes and Or-opt routes when we apply 2-opt, 3-opt, Or-opt backward search and forward search algorithms to the initial routes. Three objective functions are chosen as the measurement of the quality of the routes. Eight tables are given for comparison of the results obtained by using the different post-optimization procedures. Finally we give conclusions based on our test cases.

All test cases were run on IBM PC-486 SCO UNIX operating system. The test code was written in the C language.

4.1 Test Problem Description

The test problem we use comes from a specific city in Ontario, Canada. One of the service agencies in the city has one depot and hundreds of requests every day. Each REQUEST(*i*) has its pickup address, dropoff address and desired service time (desired pickup time or dropoff time, but not both). The method used to set up a time window for each STOP(*pd*) (pickup or dropoff stop) is based on the algorithm described in Chapter 2 where

$$\text{MAX_PWT}(i) = 30 \text{ mins}$$

$$\text{MAX_ET}(i) = 100\% * t(\text{pickup}(i), \text{dropoff}(i))$$

$$\text{MAX_VOYAGE} = 60 \text{ mins}$$

$$\text{HT}(\text{pd}) = \begin{cases} 3 \text{ mins} & \text{if a wheelchair person} \\ 2 \text{ mins} & \text{if an ambulatory person} \end{cases}$$

Three types of vehicles, Van, Car and Orion, are used. Van and Orion can take both wheelchair and ambulatory people, but Car can only take ambulatory people. Each Van can take 4 wheelchair people, and a total of 4 ambulatory and accompanying people (as for the capacity constraint, we treat ambulatory and accompanying people the same). But the seats on board of Van cannot be folded. Each Car can take 4 ambulatory and accompanying people in total, and each Orion has 16 seats in which 8 seats can be folded for 4 wheelchair people (suppose each wheelchair person needs two seat spaces).

The address of each stop is given by its longitude and latitude, so we can calculate the distance between any two stops by using the formula mentioned in Section 2.1.3. If we use a map software package, the distance between two stops will be calculated based on the shortest path on the map. To simplify our tests, we use a vehicle average speed of 50km/hr, as provided by the city. Three objective functions are used for Or-opt algorithms to improve the initial routes (for 2-opt and 3-opt algorithms, only the first one is used):

- (1) Minimize the vehicle total travel time
- (2) Minimize the vehicle total completion time
- (3) Minimize the vehicle partial completion time & total travel time

In order to judge the quality of the routes, we also evaluate the values of Vehicle Productivity Rate (VPR) of initial and improved routes which give the average number of people transported per hour. We expect that, after post-optimization, the values of VPR of the improved routes will not deteriorate.

4.2 Test Results

4.2.1 Initial Routes

We have chosen six test cases here. Each case involves 50 requests. Figure 4.1 shows the distribution of these service stops ("." represents a stop, "*" means that two or more stops overlap). The distance between two stops in Figure 4.1 does not represent the actual distance of these two stops because the surface of the earth is not flat.

Figure 4.1: The distribution of the given stops

The working hours of the given depot are 07:00 - 19:00. Any request with the `earliest_arrival_time` of its pickup stop earlier than the depot start time or later than the depot finish time is considered as a special request. In a real situation, the service company can offer 24 hour service every day. But in our case, we do not handle these special requests.

A set of routes is generated for each test case using the insertion heuristic algorithm presented in Chapter 2. The input data includes available vehicle list and driver shift list. Since we use only time to cluster the requests, the cluster time interval may be adjusted depending on the maximum waiting time allowed at each stop. In a real situation, the service company gives the driver some time for break and lunch. So we should consider this factor when we design the routes. Another consideration is to make drivers work

evenly. In some service agencies, a fixed number of drivers work during the day time, no matter how many requests need to be serviced. In this situation, we try to allocate duties evenly by limiting the maximum number of requests per route. This consideration may not be as critical as the first one. We simulate both situations within our test cases by choosing different values of the parameters, *Max_Waiting_Time_Per_Stop* and *Max_Num_Of_Request_Per_Route*.

The results of our six test cases are presented in Table 4.1 and Table 4.2. Each case successfully serves 45 requests. 24 wheelchair people, 21 ambulatory people and 4 accompanying people are transported. Each case also shows the number of routes

Table 4.1: Initial routes

	No. of Routes	Travel Time	Waiting Time	Handling Time	Total Time	Loaded Waiting Time	Completion Time	No. of Van	No. of Car	No. of Orion
Case1:	6	456	706	228	1390	45	1162	3	2	1
Case2:	7	513	190	228	931	15	703	4	2	1
Case3:	7	507	176	228	911	15	683	4	2	1
Case4:	10	546	210	228	984	22	756	6	3	1
Case5:	10	514	181	228	932	36	695	6	3	1
Case6:	12	550	164	228	942	15	714	7	4	1

Table 4.2: Initial routes (continued)

	VPR	Max. Waiting Time Per Stop	Max. Number of Requests Per Route	Actual Max. Number of Requests Per Route
Case1:	4.298	180	50	12
Case2:	3.968	90	50	19
Case3:	4.000	60	15	15
Case4:	3.798	60	10	10
Case5:	3.962	30	10	10
Case6:	3.779	30	7	7

generated, vehicle travel time of all routes, waiting time of all routes and handling time of all routes (time is in minutes). Here loaded vehicle waiting time (vehicle waiting time when any handicapped person is on board) and completion time of all routes are calculated in order to analyze results and compare them with the results obtained from other cases later. Table 4.1 and Table 4.2 also show the number of different types of vehicles used. The VPR for all cases is calculated and is considered to be a measurement of service quality. We use the initial routes as input to run 2-opt and 3-opt and Or-opt (backward and forward search) algorithms to get improved routes.

4.2.2 2-opt and 3-opt Solutions

2-opt and 3-opt algorithms for the TSP problem have been studied and tested completely before. In order to get a comparison with the results obtained from Or-opt, we modified the 2-opt and 3-opt algorithms to our DARP-TW in paratransit. For each of our 6 test cases, we use 2-opt and 3-opt algorithms to improve our initial routes. Due to the computational complexity of 3-opt, we only consider the interchanges preserving the direction of traversal but exchanging the order in which the segments are visited (see Figure 3.3a). Our version is based on [Baker 86], where he observes that less than ten percent of the improvements found involved the reversal of the orientation of a sequence of two or more nodes for a given route. The objective function chosen here is to minimize the total travel time. Tables 4.3 - 4.8 show the results of 2-opt and 3-opt for our initial routes.

4.2.3 Or-opt Solutions

The Or-opt algorithm includes backward search and forward search. Theoretically any of the two can be used to improve the given routes. As we discussed in Chapter 3, forward search needs extra $O(N)$ time to do the feasibility check, whereas backward search needs only $O(1)$ time for the check. In both algorithms, we try to relocate a string of one, two or three currently adjacent stops in the route either backward or forward. Three objective functions are used to improve the quality of the initial routes. All the feasibility constraints,

including the service time windows at each stop, the customer precedence and vehicle capacity, are checked during the optimization. In order to compare the results of Or-opt with the results of 2-opt and 3-opt, we show the computational results also in Tables 4.3 - 4.8.

Table 4.3: Results of improvement heuristics (Case1)

	Travel Time	Waiting Time	Handling Time	Total Time	Loaded Waiting Time	Completion Time	VPR
Case1: Initial Routes	456	706	228	1390	45	1162	4.298
2-opt (Min. travel time)	431	717	228	1376	53	1148	4.461
3-opt (Min. travel time)	456	706	228	1390	45	1162	4.298
Or-opt (backward, Min. travel time)	426	721	228	1375	52	1147	4.495
Or-opt (backward, Min. completion time)	455	706	228	1389	45	1161	4.305
Or-opt (backward, Min. partial time)	430	724	228	1382	52	1154	4.468
Or-opt (forward, Min. travel time)	432	729	228	1389	56	1161	4.455
Or-opt (forward, Min. completion time)	446	708	228	1382	54	1154	4.362
Or-opt (forward, Min. partial time)	432	729	228	1389	56	1161	4.455

Table 4.4: Results of improvement heuristics (Case2)

	Travel Time	Waiting Time	Handling Time	Total Time	Loaded Waiting Time	Completion Time	VPR
Case2: Initial Routes	513	190	228	931	15	703	3.968
2-opt (Min. travel time)	500	180	228	908	1	680	4.038
3-opt (Min. travel time)	512	191	228	931	15	703	3.973
Or-opt (backward, Min. travel time)	485	187	228	900	0	672	4.123
Or-opt (backward, Min. completion time)	505	190	228	923	15	695	4.011
Or-opt (backward, Min. partial time)	488	207	228	923	15	695	4.106
Or-opt (forward, Min. travel time)	499	180	228	907	4	679	4.044
Or-opt (forward, Min. completion time)	507	172	228	907	4	679	4.000
Or-opt (forward, Min. partial time)	499	180	228	907	4	679	4.044

Table 4.5: Results of improvement heuristics (Case3)

	Travel Time	Waiting Time	Handling Time	Total Time	Loaded Waiting Time	Completion Time	VPR
Case3: Initial Routes	507	176	228	911	15	683	4.000
2-opt (Min. travel time)	493	162	228	883	1	655	4.078
3-opt (Min. travel time)	505	177	228	910	15	682	4.011
Or-opt (backward, Min. travel time)	477	176	228	881	1	653	4.170
Or-opt (backward, Min. completion time)	500	176	228	904	15	676	4.038
Or-opt (backward, Min. partial time)	481	195	228	904	16	676	4.147
Or-opt (forward, Min. travel time)	491	162	228	881	4	653	4.089
Or-opt (forward, Min. completion time)	496	157	228	881	4	653	4.061
Or-opt (forward, Min. partial time)	491	162	228	881	4	653	4.089

Table 4.6: Results of improvement heuristics (Case4)

	Travel Time	Waiting Time	Handling Time	Total Time	Loaded Waiting Time	Completion Time	VPR
Case4: Initial Routes	546	210	228	984	22	756	3.798
2-opt (Min. travel time)	537	196	228	961	8	733	3.843
3-opt (Min. travel time)	533	223	228	984	30	756	3.863
Or-opt (backward, Min. travel time)	521	195	228	944	6	716	3.925
Or-opt (backward, Min. completion time)	539	210	228	977	22	749	3.833
Or-opt (backward, Min. partial time)	525	214	228	967	15	739	3.904
Or-opt (forward, Min. travel time)	537	196	228	961	11	733	3.843
Or-opt (forward, Min. completion time)	542	191	228	961	11	733	3.818
Or-opt (forward, Min. partial time)	537	196	228	961	11	733	3.843

Table 4.7: Results of improvement heuristics (Case5)

	Travel Time	Waiting Time	Handling Time	Total Time	Loaded Waiting Time	Completion Time	VPR
Case5: Initial Routes	514	181	228	923	36	695	3.962
2-opt (Min. travel time)	488	182	228	898	33	670	4.106
3-opt (Min. travel time)	514	180	228	922	43	694	3.962
Or-opt (backward, Min. travel time)	485	199	228	912	32	684	4.123
Or-opt (backward, Min. completion time)	514	181	228	923	36	695	3.962
Or-opt (backward, Min. partial time)	494	194	228	916	36	688	4.072
Or-opt (forward, Min. travel time)	495	189	228	912	36	684	4.066
Or-opt (forward, Min. completion time)	510	162	228	900	25	672	3.984
Or-opt (forward, Min. partial time)	495	189	228	912	36	684	4.066

Table 4.8: Results of improvement heuristics (Case6)

	Travel Time	Waiting Time	Handling Time	Total Time	Loaded Waiting Time	Completion Time	VPR
Case6: Initial Routes	550	164	228	942	15	714	3.779
2-opt (Min. travel time)	528	154	228	910	1	682	3.889
3-opt (Min. travel time)	549	164	228	941	15	713	3.784
Or-opt (backward, Min. travel time)	522	167	228	917	4	689	3.920
Or-opt (backward, Min. completion time)	548	164	228	940	15	712	3.789
Or-opt (backward, Min. partial time)	525	186	228	939	19	711	3.904
Or-opt (forward, Min. travel time)	534	156	228	918	4	690	3.858
Or-opt (forward, Min. completion time)	542	147	228	917	4	689	3.818
Or-opt (forward, Min. partial time)	534	156	228	918	4	690	3.858

4.3 Conclusions

There are many factors which can be considered in generating a new test case. In addition to limiting the maximum waiting time allowed at each stop and the maximum number of requests allowed per route, we can choose, for example, the type of service vehicles or the manner in which to generate a route which best fits the driver shift. So it is not easy to obtain typical test cases.

From the results of the six test cases used here, we can conclude that the total number of routes generated for each test case relates to our two parameters, the maximum waiting time allowed per stop and the maximum number of requests allowed per route. Obviously, the longer the waiting time allowed per stop, the smaller the number of vehicles needed. Usually, if the service agency wants to use the minimum number of vehicles to serve the customers, the objective can be reached by enlarging the cluster time interval, or by setting up the maximum waiting time allowed per stop to a proper large number. The difficulty is how to get a balance between these two parameters. On the other hand, loaded vehicle waiting time is another factor which needs to be considered. From Table 4.1 and Table 4.2, we can find out that the total waiting time is very high for some test cases, but the loaded vehicle waiting time is acceptable. So the quality of the routes can be maintained because the driver can use most of the waiting time as lunch time or break time. Furthermore, the customer inconvenience function, as we discussed in Chapter 3, depends mainly on the customer excess travel time. An increase of loaded vehicle waiting time will definitely increase the customer excess travel time. Therefore, when we do the post-optimization, we expect that the total loaded vehicle waiting time can be reduced as well. This is shown in our results, but not consistently.

Since our post-optimization algorithms are based on the k-opt algorithm of the TSP problem, the more interchanges within a route, the greater the possibility that we can improve the route. It is more difficult to get a feasible interchange in paratransit, because the time window constraints, the customer precedence constraints and the vehicle capacity

constraints have to be satisfied. Comparing the results obtained from 2-opt and 3-opt (actually we only consider one situation of the full 3-opt), we can find that the performance of 2-opt is much better than that of our 3-opt. There is only one exception. It happens in Case 4 where the total travel time for our 3-opt is smaller than the corresponding value of 2-opt. A reason could be that we only consider one specific situation of 3-opt, and more stops (6 stops have to be chosen each time) are needed than 2-opt (4 stops) so that the feasibility constraints are easier to be violated, probably due to the tightness of the time windows and the small sample size.

The results obtained from Or-opt backward search and forward search in Tables 4.3 - 4.8 show that a different objective function results in different performance. Even though there should be no difference between choosing the backward or the forward strategies to do the interchanges theoretically, different results are produced by the two approaches. We can see that the backward search leads to a smaller value of total travel time than the forward search, and the forward search leads to a smaller value of total completion time than the backward search for all of the six test cases. A reason for getting a smaller value of total completion time from the forward search seems to be the checking conditions of this objective function. In the backward search, we start from the depot and proceed backward to the beginning of the route. When we get $STOP(i+1)$ for which $PBS^{(i+1,ND)}$ is equal to zero, the whole procedure finishes. Obviously this method will eliminate some potential feasible interchanges. Since this situation does not happen in the forward search, it is the probable cause why the forward search generates better results. As for the objective function of minimizing partial completion time and total travel time, it is difficult to make a conclusion based on the test results.

Another interesting point is that the results obtained from 2-opt, compared with Or-opt, are acceptable. From Tables 4.3 - 4.8, it is easy to see that the total travel time obtained in 2-opt is close to that one obtained in Or-opt forward search, and the total completion time is better than the one obtained in Or-opt backward search. As for the total

A resolution test chart featuring several groups of horizontal and vertical lines of varying thicknesses. Each group is accompanied by a numerical value indicating the resolution level. The values include 1.0, 1.1, 1.25, 1.4, 1.6, 1.8, 2.0, 2.2, 2.5, 2.8, 3.2, 3.6, 4.0, 4.5, 5.0, 5.6, and 6.3. The chart is used to measure the resolving power of an optical system.

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

time (travel time, waiting time and handling time altogether), 2-opt gives better results than Or-opt backward search (minimization of the partial completion time and total travel time). Considering the Vehicle Productivity Rate (VPR), 2-opt also has acceptable results.

With respect to computational complexity, Or-opt backward search is the most efficient algorithm (it needs $O(1)$ time to check feasibility conditions), even though the interchanges considered form only a subset of those considered in 3-opt algorithm. The Or-opt forward search algorithm has the same complexity as the 2-opt algorithm. For our 3-opt algorithm, it seems that we need to consider all interchanges not only those preserving the direction of traversal, but also those reversing the direction of traversal in paratransit.

Finally, we can claim that, in paratransit, Or-opt backward search is the best method for improving the initial routes, if the objective function is the minimization of the vehicle total travel time. To improve the quality of the initial routes as a whole, 2-opt algorithms can also produce similar and acceptable results.

Chapter 5

Summary and Conclusions

The final chapter summarizes the work done in this thesis, presents some conclusions, and indicates possible directions of future work. An overview of vehicle routing and scheduling problem with time windows (VRSPWT) and dial-a-ride problem with time windows (DARPTW) is given in Chapter 1. The insertion heuristic algorithm, which is described in Chapter 2, is used to set up our initial routes. In Chapter 3, we provide a detailed description of the Or-opt backward and forward search algorithms for intra-route post-optimization. We compare them with the 2-opt and 3-opt algorithms. A new inter-route post-optimization algorithm is presented. It is based on the insertion heuristics. Finally, Chapter 4 gives computational results obtained by using the 2-opt, 3-opt, and Or-opt (both backward and forward search) algorithms for six chosen test cases.

5.1 Summary of the Thesis

The problem studied in this thesis is the so called DARPTW in paratransit. Since this combinatorial optimization problem is NP-hard, two basic techniques are used to get an acceptable solution. Heuristic algorithms are used to quickly generate initial routes that are feasible with respect to all of the constraints (e.g., precedence constraints, time windows constraints and vehicle capacity constraints). Then post-optimization algorithms, which are based on the k-interchange scheme and insertion heuristic algorithm, are used to improve the initial routes according to the given objective functions. The post-optimization is applied not only within each route but also between routes.

The main objective of the thesis is to develop post-optimization algorithms, to implement them, and to present some computational results for the DARPTW in paratransit. Normally we can apply the techniques used in TSP optimization, such as 2-opt

and 3-opt algorithms, to our problem to obtain the solutions. However, in TSP optimization, we only consider an improvement of the objective function. In our situation, not only the value of the objective function needs to be improved, but also a straightforward method is needed to check the feasibility constraints to guarantee the modified route is feasible before we do the interchange. To reduce the computational complexity, Or-opt algorithms consider only a subset of all possible 3-interchanges. Based on the ideas of [Savelsbergh 88], we give details of Or-opt backward and forward search for the intra-route post-optimization, and specify the conditions to check the feasibility constraints, that is, time window constraints, customer precedence constraints and vehicle capacity constraints. Algorithms are developed for Or-opt backward search and forward search separately. In the backward search the algorithm uses $O(1)$ time to check the feasibility constraints. In the forward search $O(N)$ time is needed to check the feasibility constraints. For the inter-route post-optimization, we concentrate on minimizing the customer inconvenience which is a function of the excess travel time and the deviation from desired service time. A new algorithm based on the insertion heuristics is presented to reduce the customer inconvenience by swapping two requests from different routes.

The test results obtained from our six test cases strongly support a conclusion that Or-opt algorithm has the best performance for intra-route post-optimization, when compared with the results of 2-opt and 3-opt algorithms. 2-opt algorithm also produces acceptable results and improves the quality of the initial routes as a whole.

From the test results, we can conclude that post-optimization procedures do improve the quality of the initial routes. An interesting point is that, when we reduce the total travel time, the total waiting time gets increased. In some cases, the improvement of the initial routes is not too evident. One possible way to improve the quality of post-optimization procedures is that we run inter-route post-optimization first to get better initial routes as a whole, then run intra-route post-optimization on each separate route to improve its quality. In this way, we can reduce the impact of some factors on route initialization.

As a result, we claim that the quality and efficiency of the post-optimization procedures depend on the quality of initial routes to some extent.

5.2 Directions of Future Work

One area of possible future research is to develop methods that generate good quality initial routes. There are many factors that influence the quality of the initial routes. The way of calculating the time window for each stop, the way of calculating the distance between any two stops, the way of getting proper vehicle speed along different road segments (used to calculate travel time), are just some examples. Since we use "cluster-first route-second" heuristics, the way of clustering the requests is a very important factor. A popular method of clustering is to group the requests according to their geographical proximity, desired service time and travel direction. Grouping of requests is based on the chosen centroid on the map. In a real system, the scheduling software has to combine the street database, the client database, and the vehicle and driver database together to set up the initial routes. More research is needed to integrate properly the use of these databases.

Since the paratransit system has its own special characteristics, the scheduling software should respond to a customer phone call in a very short time to confirm the request. Therefore, an efficient real-time insertion routine is needed to iteratively compute the best feasible insertion point for each request into every existing route and to choose the insertion of the request into a route that minimizes the insertion cost.

Many combinatorial optimization algorithms are based on integer linear programming formulation of the problem at hand. After applying linear programming to a relaxation of the problem (that is, the solution variables are not required to be integral), these techniques usually resort to time-consuming enumeration routines (such as dynamic programming or branch-and-bound) in order to obtain feasible solutions. This is another direction which can be applied to the problem in paratransit to improve the initial routes.

As for the inter-route post-optimization, there are no research papers on this topic published so far. We only presented an algorithm which should be fully tested in the future. Only extensive testing can confirm that the inter-route post optimization leads to better solutions of the problem or show that all the effort is a waste of time. In our opinion, the main research goal in the future is still trying to find practical, efficient algorithms for route initialization and post-optimization.

References

- [Altinkemer 85] K. Altinkemer and B. Gavish, *A Parallel Savings Heuristic for the Delivery Problem with a Log Q Error Guarantee*, Graduate School of Management, University of Rochester, NY, 1985.
- [Bellisle 84] Jean-Pierre Bellisle, F. Soumis and S. Roy, *The Impact on Vehicle Routing of Various Operational Rules of a Transportation System for Handicapped Persons*, Cahier du GERAD #84-14, Ecole des H.E.C., Montreal, Canada H3T 1V6.
- [Bodin 83] L. Bodin, B. Golden, A. Assad and M. Ball, *Routing and Scheduling of Vehicles and Crews: The State of the Art*, Computers and Operations Research 10 (2), 65-211, 1983.
- [Bodin 86] L. Bodin and T. Sexton, *The Multi-Vehicle Subscriber Dial-a-Ride Problem: The Delivery of Urban Services*, TIMS Studies in the Management Sciences 22, 736-743, 1986.
- [Clarke 64] G. Clarke and J.W. Wright, *Scheduling of Vehicles from a Central Depot to a Number of Delivery Points*, Operations Research 12, 568-581, 1964.
- [Desrosiers 84] J. Desrosiers, Y. Dumas and F. Soumis, *The Multiple Vehicle Many to Many Routing Problem With Time Windows*, Cahier du GERAD #84-13, Ecole des H.E.C., Montreal, Canada H3T 1V6.
- [Desrosiers 86] J. Desrosiers, Y. Dumas and F. Soumis, *A Dynamic Programming Solution of the Large-Scale Single-Vehicle Dial-A-Ride Problem With time Windows*, Cahier du GERAD #84-12(Revised), Ecole des H.E.C., Montreal, Canada H3T 1V6.

-
- [Desrosiers 88] J. Desrosiers, M. Sauve and F. Soumis, *Lagrangian Relaxation Methods for Solving the Minimum Fleet Size Multiple Traveling Salesman Problem With Time Windows*, Management Science 34, 1005-1022, 1987.
- [Desrochers 89] M. Desrochers and T.W. Verhoog, *A Matching Based Savings Algorithm for the Vehicle Routing Problem*, Cahier du GERAD G-89-04, Ecole des H.E.C., Montreal, Canada H3T 1V6.
- [Desrochers 90a] M. Desrochers, J. Desrosiers and M.M. Solomon, *A New Optimization Algorithm for the Vehicle Routing Problem With Time Windows*, Cahiers du GERAD G-90-30, Ecole des H.E.C., Montreal, Canada H3T 1V6.
- [Desrochers 90b] M. Desrochers, J. Gilbert, M. Sauve and F. Soumis, *CREW-OPT: Subproblems Modeling in a Column Generation Approach to Urban Crew Scheduling*, Cahier du GERAD G-90-30, Ecole des H.E.C., Montreal, Canada H3T 1V6.
- [Dumas 85] Y. Dumas, *Confection d'itineraires de vehicules en vue du transport de plusieurs origines a plusieurs destinations*, Publication #434, Centre de recherche sur les transports, Universite de Montreal, Montreal, Canada H3C 3A7, 1985
- [Dumas 89] Y. Dumas, J. Desrosiers and F. Soumis, *Large Scale Multi-Vehicle Dial-A-Ride Problems*, Cahier du GERAD G-89-30, Ecole des H.E.C., Montreal, Canada H3T 1V6.
- [Timlin 89] M.F. Timlin, *Precedence Constrained Routing and Helicopter Scheduling*, M.Math Thesis, University of Waterloo, 1989.
- [Fisher 81] M.L. Fisher and R. Jaikumar, *A Generalized Assignment Heuristic for Vehicle Routing*, Network 11, 109-124, 1981.

-
- [Gendreau 91] M. Gendreau, A. Hertz, and G. Laporte, *A Tabu Search Heuristic for the Vehicle Routing Problem*, Publication CRT-777, Centre de recherche sur les transports, Montreal, 1991.
- [Gillett 74] B. Gillett and L. Miller, *A Heuristic Algorithm for the Vehicle Dispatch Problem*, Operations Research 22, 340-349, 1974.
- [Glover 77] F. Glover, *Heuristic for Integer Programming Using Surrogate Constraints*, Decision Sciences 8, 156-166, 1977.
- [Golden 88] B.L. Golden and A.A. Assad (eds.) , *Vehicle Routing: Methods and Studies*, North-Holland, Amsterdam, 1988.
- [Haimovich 85] M. Haimovich and A.H.G. Rinnooy Kan, *Bounds and Heuristics for Capacitated Routing Problems*, Mathematics of Operations Research 10, 527-542, 1985.
- [Hung 82] H. Hung, R. Chapman, W. Hall and E. Neigut, *A Heuristic Algorithm for Routing and Scheduling Dial-A-Ride Vehicles*, OSRA/TIMS National Meeting, San Diego, Calif., 1982.
- [Jaw 86] J. Jaw, A. Odoni, H. Psaraftis, and N. Wilson, *A Heuristic Algorithm for the Multi-Vehicle Advanced Request Dial-A-Ride Problem with Time Windows*, Transportation Research B 20, 243-257, 1986.
- [Kikuchi 87] S. Kikuchi, *Vehicle Routing and Scheduling Development for Transportation of Elderly and Handicapped Persons*, Transportation Quarterly 41, 207-227, 1987.
- [Lapalme 92] Guy Lapalme et al., *GEO ROUTE: A Geographic Information System for Transportation Applications*, Communication of the ACM, Vol.35, No.1, 80-88, January 1992.

-
- [Laporte 84] G. Laporte and Y. Nobert, *Exact Algorithms for the Vehicle Routing Problem*, Surveys in Combinatorial Optimization, North-Holland, Amsterdam, 147-184, 1984.
- [Lawler 85] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (eds.) , *The Traveling Salesman Problem*, John Wiley & Sons, Chichester, 1985.
- [Lenstra 81] J.K. Lenstra, A.H.G. Rinnooy Kan, *Complexity of Vehicle Routing and Scheduling Problems*, Networks 11, 221-227, 1981.
- [Lin 73] S. Lin and B.W. Kernighan, *An Effective Heuristic Algorithm for the Traveling Salesman Problem*, Operations Research 21, 498-516, 1973.
- [Mole 76] R.H. Mole and S.R. Jameson, *A Sequential Route-Building Algorithm Employing a Generalised Savings Criterion*, Operational Research Quarterly 27, 503-511, 1976.
- [Noon 91] C.E. Noon, J. Mittenhal and R. Pillai, *A TSSP+1 Decomposition Approach for the Capacity-Constrained Vehicle Routing Problem*, Working Paper, Management Science Program, The University of Tennessee, Knoxville, TN, 1991.
- [Or 76] I. Or, *Traveling Salesman-type Combinatorial Problems and Their Relation to the Logistics of Blood Banking*, Ph.D Thesis, Dept. of Industrial Engineering and Management Sciences, Northwestern University, 1976.
- [Paessens 88] H. Paessens, *The Savings Algorithm for the Vehicle Routing Problem*, European Journal of Operational Research 34, 336-344, 1988.
- [Psaraftis 80] H.N. Psaraftis, *A Dynamic Programming Solution to the Single-Vehicle, Many-to-Many, Immediate Request Dial-A-Ride Problem*, Trans. Sci. 14, 130-154, 1980.

-
- [Psaraftis 83a] H.N. Psaraftis, *An Exact Algorithm for the Single-Vehicle Many-to-Many Dial-A-Ride Problem with Time Windows*, Trans. Sci. 17, 351-357, 1983.
- [Psaraftis 83b] H.N. Psaraftis, *Analysis of an $O(N^2)$ Heuristic for the Single Vehicle Many-to-Many Euclidian Dial-A-Ride Problem*, Transportation Research B 17B (2), 133-145, 1983.
- [Pureza 91] V.M. Pureza and P.M. Franca, *Vehicle Routing Problems via Tabu Search Metaheuristic*, Publication CRT-747, Centre de recherche sur les transports, Montreal, 1991.
- [Roy 84a] S. Roy, J. Rousseau, G. Lapalme and J. Ferland, *Routing and Scheduling of Transportation Services for Disabled Persons: The Algorithm*, TP 5596E, Transport Canada, Transportation Development Center, Montreal, Canada H2Z 1X4, 1984.
- [Roy 84b] S. Roy, J. Rousseau, G. Lapalme and J. Ferland, *Routing and Scheduling of Transportation Services for Disabled Persons: The Tests*, TP 5598E, Transport Canada, Transportation Development Center, Montreal, Canada H2Z 1X4, 1984.
- [Savelsbergh 85] M.W.P. Savelsbergh, *Local Search in Routing Problems With Time Windows*, Annals of Operations Research 4, 285-305, 1985.
- [Savelsbergh 88] M.W.P. Savelsbergh, *Implementing Local Search Algorithms for Constrained Routing Problems*, Center for Mathematics and Computer Science, Amsterdam, Erasmus University, Rotterdam, 1988.
- [Semet 91] F. Semet and E. Taillard, *Solving Real-Life Vehicle Routing Problems Efficiently Using Taboo Search*, Working paper ORWP 91/03. Department de Mathematiques, Ecole Polytechnique Federale de Lausanne, Switzerland, 1991.

-
- [Sexton 85a] T. Sexton and L. Bodin, *Optimizing Single Vehicle Many-to-Many Operations with Desired Delivery Times: I. Scheduling*, Transportation Science 10(4), 378-410, 1985.
- [Sexton 85b] T. Sexton and L. Bodin, *Optimizing Single Vehicle Many-to-Many Operations with Desired Delivery Times: II. Routing*, Transportation Science 10(4), 411-435, 1985.
- [Solomon 86] M.M. Solomon, *On the Worst-Case Performance of Some Heuristics for the Vehicle Routing and Scheduling Problem With Time Window Constraints*, Networks 16, 161-174, 1986.
- [Solomon 87] M.M. Solomon, *Algorithms for the Vehicle Routing and Scheduling Problem With Time Window Constraints*, Operations Research 35, 254-265, 1987.
- [Solomon 88a] M.M. Solomon and J. Desrosiers, *Time Window Constrained Routing and Scheduling Problems: A Survey*, Transportation Science 22, 1-13, 1988.
- [Solomon 88b] M.M. Solomon, E. Baker and J. Schaffer, *Vehicle Routing and Scheduling Problems With Time Window Constraints: Efficient Implementations of Solutions Improvement Procedures*, Vehicle Routing: Methods and Studies, North-Holland, 1988.
- [Willard 89] J.A.G. Willard, *Vehicle Routing Using r-Optimal Tabu Search*, M.Sc. Dissertation, The Management School, Imperial College, London, 1989.
- [Wilson 71] H. Wilson, J. Sussman, H. Wang and B. Higonnet, *Scheduling Algorithms for Dial-A-Ride Systems*, Urban Systems Laboratory Report USL TR-70-13, MIT, 1971.
- [Yellow 70] P. Yellow, *A Computational Modification to the Savings Method of Vehicle Scheduling*, Operational Research Quarterly 21, 281-283, 1970.

END

1 9-0 7-9 4

FIN