Solving Real-World Vehicle Routing Problems Using MILP and PGreedy Heuristics

Diplomarbeit

by
Zhi Yuan

Supervised by
Dr. Armin Fügenschuh

Technische Universität Darmstadt
Department of Mathematics
Research Group of Discrete Optimization
Prof. Dr. Alexander Martin

October 2007

Dedicated to YANG Yun,
the light of my life.

Table of Contents

1 INTRODUCTION 8

2 MATHEMATICAL BACKGROUND 11

2.1 INTEGER LINEAR PROGRAMMING 11
2.2 NETWORK FLOWS 11
2.2.1 Directed Graphs and Networks 12
2.2.2 Minimum Cost Flows and Single Commodity Flows 12
2.2.3 Multicommodity Minimum Cost Flows 12

3 VEHICLE ROUTING PROBLEM (VRP) 14

3.1 INTRODUCTION 14
3.2 PROBLEM CLASSIFICATION 17
3.2.1 Capacitated Vehicle Routing Problem (CVRP) 17
3.2.2 Heterogeneous Vehicle Routing Problem with Time Windows (HVRPTW) 20

4 RICH VEHICLE ROUTING PROBLEMS – THE INDUSTRIAL APPLICATIONS 24

4.1 SCHEDULING NURSES FOR HOME HEALTH CARE SERVICES 24
4.1.1 Project Description for Mobile Nurse Scheduling 24
4.1.2 Mathematical Model for Mobile Nurse Scheduling 27
4.1.2.1 The graphical model and sets 28
4.1.2.2 Parameters 29
4.1.2.3 Variables, Objective and Constraints 30
4.2 SCHOOL TAXI ROUTING FOR HANDICAPPED PUPILS 33
4.2.1 Project Description for School Taxi Routing 33
4.2.2 Mathematical Model for School Taxi Routing 35
4.2.2.1 The graphical model and sets 36
4.2.2.2 Parameters 38
4.2.2.3 Variables, Objective and Constraints 39
4.3 CYCLIC LOCOMOTIVE SCHEDULING 41
4.4 MULTI-DEPOT LOCOMOTIVE SCHEDULING 43
4.5 PREPROCESSING – REDUCING BIG-M CONSTRAINTS 44
4.6 TIGHTER BIG-M FORMULATION 48

5 PRIMAL HEURISTICS 49

5.1 A GREEDY CONSTRUCTIVE SEARCH PARADIGM FOR HVRPTW 49
5.1.1 The Big Picture of the Greedy Construction Search 49
5.1.2 Initialize the Search Graph 51
5.1.3 Select the Best Depot to Start 52
5.1.4 Select the Best First Customer 54
5.1.5 Select the Best Next Customer 56
5.1.6 Computational Complexity of the Greedy Construction search 60
5.2 HOW TO IMPROVE THE GREEDY CONSTRUCTION SEARCH 61

5.2.1 Intensity versus Diversity *62*
5.2.2 Semi-greedy Algorithm -- The Utilization of Restricted Candidate List *63*
5.2.3 Random selection at local step by a perturbation multiplier *64*
5.2.4 Parameterize the greedy scoring function *65*
5.3 PARAMETERIZE THE GREEDY ALGORITHM (PGREEDY) 65
5.3.1 How to parameterize a greedy algorithm *66*
5.3.2 How to parameterize a greedy algorithm for HVRPTW *71*
5.3.3 Automated parameter tuning for PGreedy *72*
5.3.3.1 Parameter tuning with Grid Search 73
5.3.3.2 Parameter tuning with Uninformed Random Picking 74
5.3.3.3 Parameter tuning with Improving Hit and Run 75
5.4 RANDOMIZED GREEDY SEARCH (RGS) 78
5.5 THE HYBRID OF PGREEDY AND RGS: PARGS 82
5.5.1 Combine PGreedy and RGS subsequently *82*
5.5.2 Embed RGS in PGreedy at local steps *83*
5.6 STARTING TIME PROPAGATION 83
5.6.1 The simplified monotone IP2 and its graphical model *84*
5.7 LOCAL SEARCH 88
5.8 APPLICATION EXAMPLES 89
5.8.1 Home Health Care Services *89*
5.8.2 School Taxi Routing *90*
5.8.3 Cyclic Locomotive Scheduling *90*
5.8.4 Multi-Depot Locomotive Scheduling -- Handling Coupling Trips *92*

6 COMPUTATIONAL RESULTS 94

6.1 SOLUTION PROCESS OF OUR SOFTWARE SYSTEM FOR RICH VEHICLE ROUTING PROBLEMS 94
6.2 SOLVING THE MILP MODEL WITH ZIMPL, CPLEX AND OPL 96
6.3 TEST ENVIRONMENT 96
6.4 SOLVING MOBILE NURSE SCHEDULING FOR HOME HEALTH CARE SERVICES 96
6.4.1 The real-world input data Diakonie *97*
6.4.2 The smaller instances extracted from the real-world instance *98*
6.5 SCHOOL TAXI ROUTING FOR HANDICAPPED PUPILS 102
6.5.1 The real-world input data *102*
6.5.2 The extracted instances *105*
6.6 CYCLIC LOCOMOTIVE SCHEDULING 108
6.7 MULTI-DEPOT LOCOMOTIVE SCHEDULING 109

BIBLIOGRAPHY 110

Table of Figures

Figure 3.1: Typical input for a Vehicle Routing Problem.....14
Figure 3.2: An output for the VRP instance above.....15
Figure 3.3: A multi-commodity flow network model for Heterogeneous VRP.....21
Figure 4.1: Graphic model for Mobile Nurse Scheduling Problem. 28
Figure 4.2: Solving the afternoon peak problem by reflection of the morning peak process..35
Figure 4.3: Graphic model for school taxi routing problem.....36
Figure 4.4: Transformed graphical model into VRPPD formulation for the school taxi routing problem.. 36
Figure 4.5: The classification of deadhead trips.....46
Figure 5.1: Greedy selection of the next customer node..57
Figure 5.2: Reduce the two-dimensional parameter space to its hyperplane. 70
Figure 5.3: Grid Search for the two-dimensional parameter cuboid. 73
Figure 5.4: Uninformed Random Picking for the parameter cuboid.....75
Figure 5.5: Improving Hit and Run with hyperspherical direction generator and full range step size generator.. 77
Figure 5.6: probability density function of uniform distribution in [800, 1000].....79
Figure 5.7: probability density function of normal distribution N(1000, 50). 80
Figure 5.8: An acyclic relation graph.....85
Figure 5.9: A cyclic relation graph.....85
Figure 5.10: node insertion neighborhood between two paths.....88
Figure 5.11: node exchange neighborhood between two paths.....89
Figure 6.1: The solution process. 95

1 Introduction

In the last decades, with the development of techniques from Operations Research and Mathematical Programming, the field of transportations and logistics has attracted many researchers and practitioners, and achieved noticeable results. The large number of real-world applications have widely shown that the use of computerized procedures for the transportation planning results in substantial savings, generally ranging from 5% to 20% in the global cost, as reported in Toth & Vigo (2002).

This work presented here is based on my last two years' student job in the research group of discrete optimization of our department, from which I was honorably involved in a number of practical projects in this field. Four real-world projects will be presented here, scheduling nurses for home health care services, school taxi routing for handicapped pupils, and two different locomotive scheduling for freight transportations.

All the four practical projects and their test instances stem from industry and real-life, and all of them can be classified into the problem family of Vehicle Routing Problems (VRP). The VRP is firstly formulized in literature by Dantzig and Ramser (1959). It is a class of problems to assign a fleet of vehicles originating and terminating from depot, to delivering a set of geographically dispersed customers with known demands in such a way that the total transportation cost is minimized. Furthermore, all our four industrial applications fall into a special class of VRP, which we called Heterogeneous VRP with Time Windows (HVRPTW). To be more explicit, the fleet of vehicles is mixed with several distinguishable vehicle types, and each customer must be visited within a given time interval. But still, besides the general HVRPTW structure, each of the four problems has individual constraints depending on the type of application.

In our work of all the four industrial applications, we first formulize the respective problem into a mathematical model using Mixed Integer Linear Programming (MILP). The MILP formulation is based on the model of multi-commodity flows, introduced in Ahuja et. al. (1993). The resulting MILP problem is then solved by a commercial solver ILOG CPLEX.

Due to the high complexity of these practical problems, especially with the time window constraints, normally the real-world instances cannot be solved to optimality by CPLEX, and most of them cannot even yield a feasible solution within a reasonable computation time (6 hours). For such hard combinatorial problems, heuristic implementation plays an important role in the solution process.

We have developed a heuristic approach based on the PGreedy (for Parameterized Greedy Heuristic) algorithm framework for solving this class of problems. The PGreedy algorithm was developed by Fuegenschuh (2005) as a new type of

meta-heuristic. It extends the classic greedy algorithm strategy by introducing more than one greedy criterion, and parameterizing them with individual weights into a linear scoring function. A parameter tuning technique is performed to find the best parameter weight setting. To this end, methods form Global Optimization, such as improving hit-and-run (see Zabinsky et. al. 1993), can be applied.

As is also observed and described in Yuan & Fuegenschuh (2007), during our experiments, it turns out that the general framework introduced by Fuegenschuh could be further improved by including a higher degree of randomization. For instance, not only the best local step that was identified by the parameterized scoring function is selected immediately. Instead, we select randomly from the list of all scores, where the random function is biased by the score itself, so that lower scores have a higher probability for being selected. And non-linear parameterized scoring functions can also come into the scene.

We have also combined the heuristic approach with the MILP solver CPLEX, by providing the heuristic solution as a feasible starting value for the MILP problem, in the hope of gaining a better feasible solution from the solver as well as cutting off some unnecessary sub-branches in an earlier stage to improve the dual bound.

Experiment results show that our PGreedy heuristic algorithms are able to achieve large savings in various application contexts, usually 10 – 20% comparing with the current manual schedule. When the optimal solution is available, our heuristic solution is also observed to have a solution gap around 5% to 10% within optimality. The idea of providing the heuristic solution as a starting value for MILP solver also considerably narrows the global gap of the each problem.

The remainder of the article is organized as follows. In Chapter 2, we give an introduction to some mathematical background. Chapter 3 provides detailed descriptions of the problem class of Vehicle Routing Problem and its variants, especially the Heterogeneous VRP with Time Windows. Chapter 4 presents the four industrial applications, or known as Rich Vehicle Routing Problems in details and how we formulize them into mathematical MILP model, and some preprocessing techniques and tighter formulation will be also discussed. Chapter 5 gives a gentle introduction to our PGreedy heuristic algorithms, and presents how our heuristic algorithms can be applied to each individual practical project. We discuss in Chapter 6 how our test instances are built up for each project, and list all our computational results.

2 Mathematical Background

This chapter presents some mathematical prerequisites that might be needed for understanding the contents of the work. For terminology we stick mainly to Ahuja et. al. (1993), Schrijver (1986), Nemhauser and Wolsey (1999). The rest of the chapter is organized as follows, Section 2.1 gives a brief introduction to the mathematical model of integer linear programming; Section 2.2 gives a quick guide to some basic models of network flows.

2.1 Integer Linear Programming

Modeling and solving integer linear programming problem lies at the heart of discrete optimization. Various problems in science, engineering, economics and society can be modeled into integer linear programming (ILP) problems.

Given an $m \times n$ matrix A, a m-vector b, and a n-vector c. An ILP is a system of the following form:

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & \underline{x} \leq x \leq \overline{x} \\ & x \in \mathbb{Z}^n \end{aligned} \tag{2.1}$$

Solving ILP is in general NP-hard. The exact algorithm for solving this problem in general follows the branch-and-bound paradigm. The branch-and-bound algorithm searches the solution space in a systematical way, branching the problem into subproblems very much as the divide-and-conquer strategy, while keeping a bounding procedure to evaluate the dual bound (lower bound in the case of minimization problems) of the subproblems, and try to cut off as many search branches as possible when its dual bound exceeds the primal bound provided by a known feasible solution. This algorithm is known to have exponential computational complexity with respect to the problem size. For more details about the Branch-and-Bound procedure and integer linear programming, we refer to Schrijver (1986) or Nemhauser and Wolsey (1999).

2.2 Network Flows

The terminologies of this section follow Ahuja et. al. (1993) to a large extent.

2.2.1 Directed Graphs and Networks

A *directed graph* (or *digraph*) $G = (N, A)$ consists of a finite nonempty node set N and a finite arc set A whose elements are ordered pairs of distinct nodes. A *directed network* is a directed graph whose nodes and/or arcs have associated numerical values (typically costs with vector *c*, capacities with vector *u*, and/or supplies (or demands) with vector *b*).

2.2.2 Minimum Cost Flows and Single Commodity Flows

The minimum cost flow model is the most fundamental of all network flow problems. We first present an ILP formulation of the minimum cost flow problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b_i \quad \forall i \in N \\ & 0 \le x_{ij} \le u_{ij} \quad \forall (i,j) \in A \\ & x \in \mathbb{Z}^{|A|} \end{aligned} \tag{2.2}$$

Note that the well-known shortest path problem and maximum flow problem are both special cases of minimum cost flow problem. For shortest path problem, one can simply set the source node to have of one unit supply and the sink node one unit demand, with all other nodes as transshipment nodes, i.e. nodes with 0 supply or demand; for maximum flow problem, one can set the supply/demand on all nodes to be 0 and the costs on all arcs to be 0, and add a virtual arc from sink to source with cost -1.

Minimum cost flow problem can be solved in polynomial time using e.g. successive shortest path algorithm. But in practice, a pseudo polynomial-time network simplex algorithm solves the problem quite efficiently, see Ahuja et. al. (1993), or Löbel (1997) for more details.

2.2.3 Multicommodity Minimum Cost Flows

In many application contexts, including many rich vehicle routing problems that are to be presented in this work, there are more than one commodities sharing the same network, and especially sharing common facilities. In this case the network will have several copies, and each layer belongs to a commodity. A bundle constraint will tie the different commodities together.

Let x_{ij}^k denote the flow of commodity *k* on arc (*i*, *j*), and let x^k and c^k denote the flow vector and per unit cost vector for commodity *k*. Using this notation we can formulate the multicommodity flow problem as follows:

$$
\begin{aligned}
&\text{minimize} \quad \sum_{k \in K} c^k x^k \\
&\text{subject to} \quad \sum_{\{j:(i,j)\in A\}} x_{ij}^k - \sum_{\{j:(j,i)\in A\}} x_{ji}^k = b_i^k \quad \forall i \in N, k \in K \\
&\qquad 0 \le x_{ij}^k \le u_{ij}^k \quad \forall (i,j) \in A, k \in K \qquad (2.3) \\
&\qquad \sum_{k \in K} x_{ij}^k \le u_{ij} \quad \forall (i,j) \in A \\
&\qquad x \in \mathbb{Z}^{|A|\times|K|}
\end{aligned}
$$

Solving multicommodity flow problem is in general NP-hard, some state-of-the-art computation efforts have been presented in Löbel (1997).

3 Vehicle Routing Problem (VRP)

VRP is a generic name to a class of hard combinatorial problem, which arises naturally (Dantzig & Ramser 1959) as a central problem in the fields of transportation, distribution and logistics. In this section we take a journey from the very basic version of VRP and its variants to some practical applications in the industry, and present mathematical formulations accordingly using Mixed Integer Linear Programming.

3.1 Introduction

The Vehicle Routing Problem (VRP) is a generic name given to a whole class of problems in which a set of routes for a fleet of vehicles based at one or several depots must be determined for a number of geographically dispersed cities or customers. The objective of the VRP is to deliver a set of customers with known demands on minimum-cost vehicle routes originating and terminating at a depot (see Toth & Vigo 2002 or VRP Web). In the two figures below we can see a picture of a typical input for a VRP problem and one of its possible outputs:

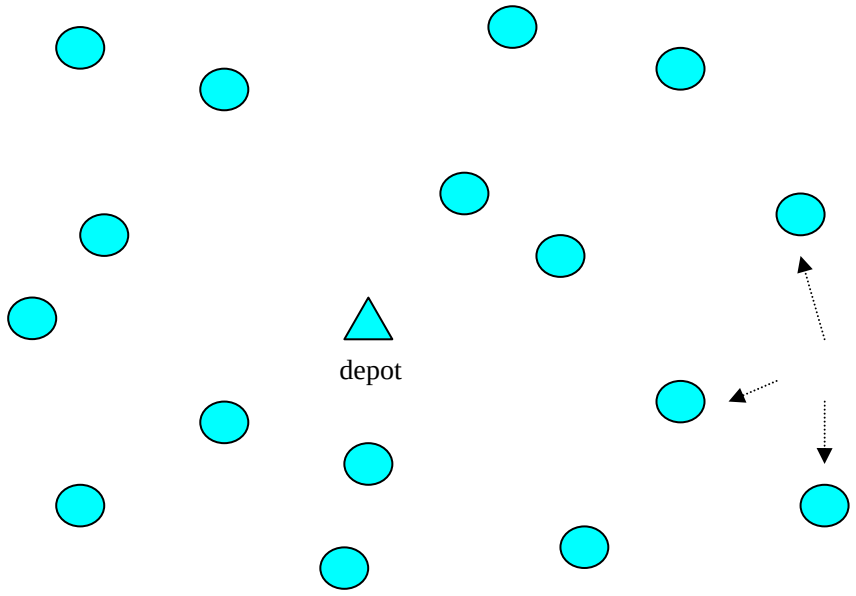


Figure 3.1: Typical input for a Vehicle Routing Problem

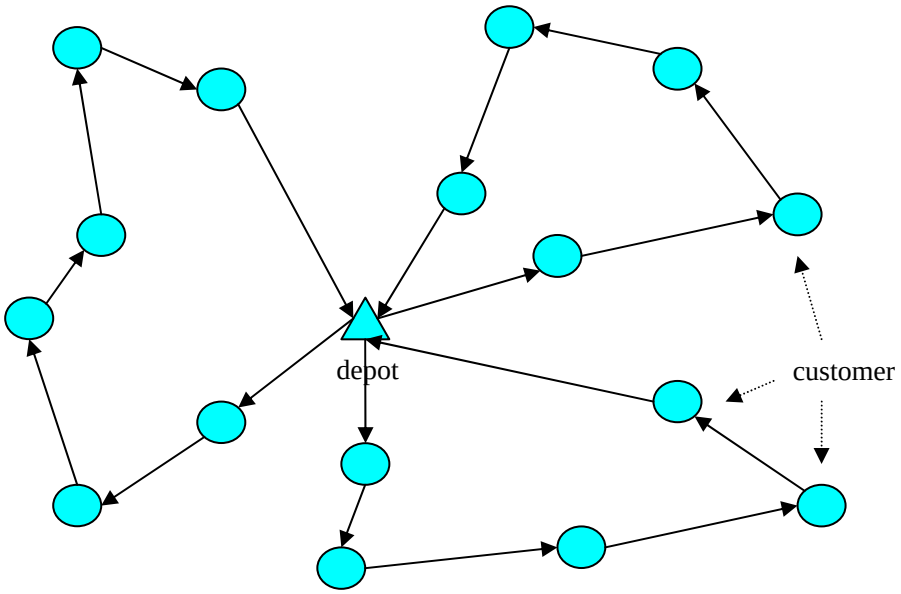


Figure 3.2: An output for the VRP instance above.

In the VRP graph example above, the single depot is drawn with triangle, while other nodes drawn with circle stand for the customers.

A road network, used for the transportation of goods, is generally given as a graph, where the nodes are possibly depots or customers, and an arc is given if the two nodes have a corresponding connection.

In the VRP each vehicle must start and end at its depot. It is natural to think of a depot as a garage where a fleet of vehicles are parked and serviced, but it does not have to be the case. There might be one garage location, like the headquarter in the Mobile Nurse Scheduling project, where the nurses start and end their daily trips; or there might be several fixed garage locations, like in the Multi-Depot Locomotive Scheduling project; or even no geographic garage need to be considered at all, e.g. the School Taxi Routing problem does not take into account from where the taxi company deploy their cars, while the Cyclic Locomotive Scheduling problem supposes each station to be a garage for their locomotives.

What could be called “customers” in the VRP can be of various existence types. The original VRP views each customer as an object at a location where a certain amount of goods should be delivered to or collected from, at a certain time or within a time interval. However, this term customer can be further generalized, for instance the home where a person should be picked up and their school where they should be dropped off as in the school taxi routing project, or a service to be performed, e.g. visiting a client at home and providing health care services as in the mobile nurse scheduling project, or a complete transportation task to be carried out from one location to another. Typical characteristics of customers are:

- The location of the customer, note that for some customer the start location and the end location might be different.
- Amount of goods (demand), possibly of different types, that has to be delivered or collected at the customer.
- The starting time interval (time windows), during which the customer should be reached. In all our application projects, each customer has specified a feasible time window, within which the service should be started, as we will see, the introduction of time windows comparing with a fixed starting time for each customer brings a significant saving in practice, but requires much harder computational efforts.
- The service time, which is required to perform the service at each customer. For pupil transportation the service time is the time that the pupils need to get on or get off the bus, while for mobile nurses the service time is the time needed to carry out a certain home health care service, and for locomotive scheduling the service time is the transportation time of a certain task.
- Subset of available vehicles that can be used to serve the customer, due to e.g. vehicle capacity or pulling power.

Customers are transported by a fleet of vehicles, which might be homogeneous, i.e. all vehicles are of the same type, or heterogeneous on the contrary, when the fleet consists of different types of vehicles. Some most important characteristics of the vehicles are size, pulling power, loading capacity, costs, maximal or average speed, and so on.

- Home depot, where the vehicle starts and ends.
- Loading capacity of the vehicle, with respect to different types of goods, passengers, baggage, etc.
- Maximal or average speed of the vehicle.
- Subset of available customers or available arcs in the road network that can be visited by the vehicle.
- Maximum driving time or distance, because of the fuel or some legal maximum working time limitation.
- Costs, associated with the utilization of the vehicle, usually include the starting cost of a vehicle, and the cost per unit distance.

The objective is usually to reduce operational costs. As we experience from our practical applications, the total costs can be considered hierarchically, the cost of starting a vehicle in service usually dominates the driving costs, therefore the minimization of the number of vehicles are usually of the foremost importance. At a subsidiary level, the driving cost should be minimized, which is usually measured by the total driving time or distance. Some other objectives may also come into the scene, which might be even in conflict with saving operational costs, for instance to balance the load of each routes, or minimize some other service quality related penalties.

In the sequel we will have a look at some different types and variant of the VRP

family.

3.2 Problem Classification

In this section we take a short journey among different members of the VRP family. We start our journey from the very basic Capacitated Vehicle Routing Problem, to different variants including Heterogeneous VRP and VRP with Time Windows.

3.2.1 Capacitated Vehicle Routing Problem (CVRP)

We first describe the simplest and most studied member of the VRP family, the Capacitated VRP (CVRP). In the CVRP, all the customers' demands and locations are known in advance, and may not be split. Each customer is visited by exactly once. The vehicles are identical and based at a single depot, and each route starts and ends at the depot. Only the capacity restrictions for the vehicles are imposed, i.e. the sum of the demands of the customers visited by a route does not exceed the vehicle capacity.

This difficult combinatorial problem conceptually lies at the intersection of these two well-studied problems:

- The Traveling Salesman Problem (TSP): If the capacity of the vehicles C is infinite, we can get an instance of the Multiple Traveling Salesman Problem (MTSP), in which m salesman starting from one origin are trying to build m round trips with minimum cost. CVRP is a generalization of the MTSP and is therefore NP-hard. An MTSP instance can be transformed into an equivalent TSP instance by adjoining to the graph k-1 (being k the number of routes) additional copies of node 0 and its incident edges (there are no edges among the k depot nodes).

- The Bin Packing Problem (BPP): The question of whether there exists a feasible solution for a given instance of the VRP is an instance of the BPP. BPP is a combinatorial NP-hard problem, in which objects of different volumes must be packed into a finite number of bins of certain capacity in a way that minimizes the number of bins used. The decision version of this problem is conceptually equivalent to a CVRP model in which all edge costs are taken to be zero (so that all feasible solutions have the same cost).

Hence, we can think of the first transformation as relaxing the underlying packing (BPP) structure and the second transformation as relaxing the underlying routing (TSP) structure. A feasible solution to the full problem is a TSP tour (in the expanded graph) that also satisfies the packing constraints (i.e., the total demand along each of the k segments joining successive copies of the depot does not exceed C). Because of

the interplay between the two underlying models (both of them are NP-hard problems), the Capacitated Vehicle Routing Problem can be very difficult to solve in practice.

A mathematical description of CVRP can be given as follows: Given is a graph $G(N, A)$ with

- $N = \{0,1,...,n\}$ is a node set with 0 denoting the depot node, and other *n* nodes $N \setminus \{0\}$ denoting the customer nodes.
- $A = \{(i, j) \mid i, j \in N, \text{ and } i \neq j\}$ is a directed arc set, note that here we consider the asymmetric version of CVRP (or ACVRP), since the symmetric case can be easily transformed into an asymmetric form.

Furthermore, given $c_{i,j}$ as the cost parameter attached to each arc $(i, j) \in A$, d is a vector of customer demand, C is the capacity of each vehicle, and K the total number of vehicles at depot.

The objective is to minimize the total cost, i.e. a weighted cost function of the number of routes and route length or travel time, to serve all the customers. Usually the cost function is defined as the sum of the costs of the arcs belonging to the routes, and in practice, starting a new vehicle in service is usually much more expensive than the driving costs that are normally related to the driving time or distance. Usually a large value of the vehicle starting cost is imposed on the arcs that connect from a depot to customers, or pull-out arcs.

The mixed integer linear programming (MILP) model for CVRP can be formulated as follows:

$$\min \quad \sum_{i \in N} \sum_{j \in N} c_{i,j} \cdot x_{i,j} \tag{3.1}$$

subject to

$$\sum_{i \in N} x_{i,j} = 1 \quad \forall j \in N \setminus \{0\} \tag{3.2}$$

$$\sum_{j \in N} x_{i,j} = 1 \quad \forall i \in N \setminus \{0\} \tag{3.3}$$

$$\sum_{i \in N} x_{0,i} \leq K \tag{3.4}$$

$$\sum_{j \in N} x_{0,j} = \sum_{i \in N} x_{i,0} \tag{3.5}$$

$$\sum_{i \in S} \sum_{j \notin S} x_{i,j} \geq r(S) \quad \forall S \subseteq N \setminus \{0\}, S \neq \varnothing \tag{3.6}$$

$$x_{i,j} \in \{0,1\} \quad \forall i, j \in N \tag{3.7}$$

The MILP model shown above is also introduced in Toth and Vigo 2002 as the standard two-index vehicle flow model. The constraints (3.2) – (3.5) describe a single commodity flow network. The constraints (3.2) and (3.3) impose that exactly one arc enters and leaves each customer node, respectively. The constraint (3.4) imposes the requirement that the flow leaving the depot should not exceed the maximum size of the fleet at the depot, and (3.5) further imposes the amount of flow that leaves the depot should be identical with the flow returns to the depot.

The so-called capacity-cut constraint (3.6) imposes both the connectivity of the solution and the vehicle capacity requirements. In fact, they stipulate that each cut $(S, V \setminus S)$ defined by a customer set *S* is crossed by a number of arcs not smaller than $r(S)$, which is the minimum number of vehicles needed to serve set *S*. Often, $r(S)$ is replaced by the trivial BPP lower bound as follows:

$$r(S) = \left\lceil \frac{d(S)}{C} \right\rceil$$

where $d(S)$ denotes the total demands of customers in *S*, and *C* the capacity of the vehicle.

Note that the family of constraints (3.6) has a cardinality growing exponentially with *n*. this means that it is practically impossible to solve directly the linear programming relaxation of problem (3.2) – (3.6). A possible way to partially overcome this drawback is to consider only a limited subset of these constraints and to add the remaining ones only if needed. Two different approaches using Lagrangian relaxation or Branch-and-Cut are introduced in Toth and Vigo Chapter 2 and 3, respectively.

Alternatively, a family of constraints equivalent to (3.6) and having a polynomial cardinality may be obtained by considering the subtour elimination constraints proposed for the TSP by Miller, Tucker and Zemlin 1960, and extending them to asymmetric CVRP as follows:

$$l_i - l_j + C \cdot x_{i,j} \leq C - d_j \quad \forall i, j \in N \setminus \{0\}, i \neq j \tag{3.8}$$

$$d_i \leq l_i \leq C \quad \forall i \in N \setminus \{0\} \tag{3.9}$$

where l_i, for $i \in N \setminus \{0\}$, is an additional continuous variable representing the load of the vehicle after visiting customer *i*. It is easy to see that constraints (3.8) and (3.9) impose both capacity and the connectivity requirements of ACVRP. Indeed, when $x_{i,j} = 0$,constraint (3.8) is not binding since $l_i \le C$ and $l_j \ge d_j$, whereas when $x_{i,j} = 1$, they impose that $l_j \ge l_i + d_j$.

It is worth noting that, as in the TSP formulation, the linear relaxation of the formulation with Miller-Tucker-Zemlin constraints (3.2) — (3.5), (3.7)—(3.9) generally is much weaker than that of formulation with exponentially many constraints as (3.2)—(3.7). Tightening constraints were proposed by Desrochers and Laporte 1991.

Despite the weakness in LP relaxation of Miller-Tucker-Zemlin formulation, there is an advantage of it, since it can be easily extended to the case of time windows.

Some solution approaches based on mathematical programming can be found in Toth & Vigo (2002), Naddef, Rinaldi (2002) and Bramel & Simchi-Levi (2002), some heuristic and metaheuristic approaches are described in Laporte & Semet (2002) and Gendreau et. al. (2002).

3.2.2 Heterogeneous Vehicle Routing Problem with Time Windows (HVRPTW)

In this subsection, a special extension of the classic CVRP is introduced, called Heterogeneous Vehicle Routing Problem with Time Windows (HVRPTW). It combines two VRP variants together, namely, heterogeneous fleet and time windows. All our practical applications described in the next sections fall in the class of Heterogeneous VRP with time windows.

The first variant Heterogeneous VRP (HVRP) is the utilization of heterogeneous fleet (or mix fleet) of vehicles. These vehicles differ in depot location, size, pulling power, loading capacity, costs, and maximal or average speed, maximum driving time, for instance.

The heterogeneous VRP is usually modeled as a multi-commodity flow problem (see Chapter 2). In order to tackle the variant of heterogeneous fleet, we first generalize the term “depot”. In contrast to regarding a depot as a garage where vehicles are parked and serviced, we consider here each depot as a fleet of homogeneous vehicles, locating at the same spot and having the same characteristics. Each depot represents a specific commodity, and each commodity builds up a layer of the multi-commodity

network, together with the copies of the customer nodes which is feasible to be visited by vehicles from this depot. It is more general to view each single vehicle as a depot or commodity, and indeed in many cases it has to be so, if some single route specific constraints come into the scene. For example, two customer nodes have to be served by the same vehicle route, or as another example, if some specific information has to be collected with respect to each single route, such as the total driving time of one vehicle on one day. However, our experiments have shown that, the more commodities there are, the more computational efforts it will take to solve the corresponding ILP model. A graphical model of the multi-commodity flow problem can be given as follows:

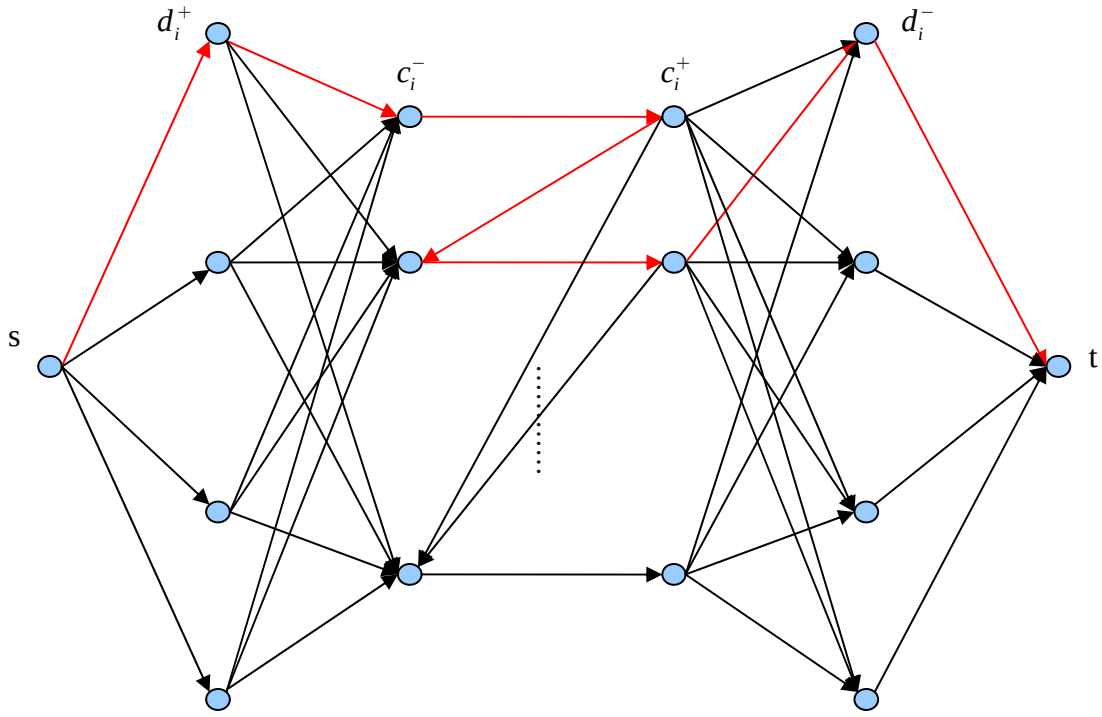


Figure 3.3: A multi-commodity flow network model for Heterogeneous VRP. The leaving depots are denoted as d_i^+, and the entering depots are denoted as d_i^-. Similarly, we decompose a customer node into two, the entering customer node is denoted as c_i^+, and the leaving customer node is denoted as c_i^-. A possible route is marked as red.

The second variant VRP with Time Windows (VRPTW) is the introduction of a time window for each customer node. Each customer expects exactly one of the vehicles to arrive at a certain time or within a certain time window. If the vehicle arrives early then waiting is permitted, but late arrival is strictly forbidden. Therefore, our time windows imposed here are hard, no early or late arrival is allowed.

As mentioned in the last section, the Miller-Tucker-Zemlin constraints can be easily adapted in the time window case. In order to simplify the problem in this section, we

drop the vehicle capacity constraints in our discussion here. However, that does not mean our vehicles will become uncapacitated and degenerate to a TSP problem, since the time windows itself has imposed a very strict time capacity constraints on the VRP problem.

A mathematical description of the constraints and objective as a mixed-integer programming model based on the multi-commodity flow problem (see chapter 2) can be stated as follows:

$$\min \quad \sum_{i\in N}\sum_{j\in N}\sum_{k\in K} c_{i,j}^{k}\cdot x_{i,j}^{k} \tag{3.10}$$

subject to

$$\sum_{i\in N}\sum_{k\in K} x_{i,j}^{k} = 1 \quad \forall j \in N\setminus\{0\} \tag{3.11}$$

$$\sum_{h\in N} x_{h,i}^{k} - \sum_{j\in N} x_{i,j}^{k} = 0 \quad \forall i \in N_{srv}, \forall k \in K \tag{3.12}$$

$$\sum_{i\in N\setminus\{0\}} x_{0,i}^{k} \le m_k \quad \forall k \in K \tag{3.13}$$

$$\sum_{i\in N} x_{0,i}^{k} - \sum_{i\in N} x_{i,0}^{k} = 0 \quad \forall k \in K \tag{3.14}$$

$$t_i + \delta_{i,j}^{k} + M\bullet(x_{i,j}^{k} - 1) \le t_j \quad \forall i, j \in N\setminus\{0\}, \forall k \in K \tag{3.15}$$

$$\underline{t_i} \le t_i \le \bar{t_i} \quad \forall i \in N\setminus\{0\} \tag{3.16}$$

$$x_{i,j} \in \{0,1\} \quad \forall i, j \in N \tag{3.17}$$

The objective function (3.10) is to minimize the operating cost, as the sum of costs on each arc that is contained in a route. Constraints (3.11) – (3.14), (3,17) describe a multi-commodity flow model, (3.11) imposes the bundle constraint, each customer can be visited only once by one vehicle. (3.12) imposes the flow conservation, the total inflow should equal the outflow. (3.13) imposes the flow capacity for depot nodes, the total flow starting from the depot should not exceed the maximum number of vehicles m_k available in the depot $k \in K$. (3.14) imposes that the flow leaving the depot and returning the depot should be identical.

The Miller-Tucker-Zemlin constraints (3.15) – (3.16) handle the time window variant, with the new variable t_i specifying the starting time of a customer *i*. (3.16) imposes that the lower bound and upper bound of the starting time t_i as $\underline{t_i}$ and $\bar{t_i}$, respectively. In (3.15), $\delta_{i,j}^{k}$ is a fixed value showing the time difference between the

start of customer *i* to the start of customer *j* by vehicle type *k*, if customer *i* and *j* are taken consecutively by vehicle type *k*. Usually $\delta_{i,j}^{k} = \delta_{i}^{service} + \delta_{i,j}^{deadhead}$, while $\delta_{i}^{service}$ denotes the service time at customer *i*, and $\delta_{i,j}^{deadhead}$ denotes the deadhead time from customer *i* to *j*. *M* is chosen as a sufficiently large value. We show that constraint (3.15) imposes both time compatibility and the connectivity requirements. As we can infer, when $x_{i,j}^{k} = 0$,constraints (3. 15) are not binding because of the sufficiently large *M*, whereas when $x_{i,j}^{k} = 1$, they impose that $t_i + \delta_{i,j}^{k} \le t_j$, so that the time compatibility holds.

We summarize the ILP model as follows:

$$\begin{aligned} &\min \quad (3.10) \\ &\text{s.t.} \quad (3.11 - 1.17) \\ &x \in \{0,1\}^{|A|\times|K|}, t \in \mathbb{Z}_{+}^{|N|} \end{aligned} \tag{3.18}$$

The ILP models of all our practical applications will be based on the formulation above for heterogeneous VRP with time windows.

4 Rich Vehicle Routing Problems – the Industrial Applications

In this chapter, four different practical applications of the VRP will be introduced. All these problems are based on real-world industrial projects. The term Rich Vehicle Routing Problems are usually referred to as the family of the extended problems of VRP, which include aspects that are essential to the routing of vehicles in real-life world. The four applications, namely, the Mobile Nurse Scheduling, School Taxi Routing, and the two Locomotive Scheduling problems all shares the common feature: heterogeneous fleets and time windows. Each project brings in some application-specific constraints, which we will discuss in details.

4.1 Scheduling Nurses for Home Health Care Services

An overview and some terminologies used in the mobile nurse scheduling project will be presented in section 4.1.1, and followed by the mathematical model description in the later sections.

4.1.1 Project Description for Mobile Nurse Scheduling

The health care service system in Germany and many other countries is facing increasing costs due to the ageing population. In our project Mobile Nurse Scheduling in cooperation with a local health care service provider, we focus on the specific field of home health care, i.e. visiting and providing medical services to clients at home. These medical services range from washing, cleaning to personal hygiene to some medical treatments, including blood pressure measuring, medication prescription, injections and so on. For notational simplicity we will refer to the employees of the service as nurses, the clients as patients, the service activities that the customers require as treatments.

- Schedule: our task is to develop computer software in order to assist the health care provider to generate a nurse schedule on a weekly basis. The task is to assign each patient or treatment to a nurse with competent qualification on an appropriate day at a time within a patient-specified time window. Multiple optimization objectives, some of which might even conflict with each other, are expected to be resolved in the three-phase process.
  - Construction: during the construction phase we will try to build up a nurse visitation schedule and find feasible daily routes for each nurse from scratch, while satisfying all side constraints. Our objective in this phase is in general

to minimize the operating cost. The primary goal of saving cost is to reduce the staff size, use as few nurses as possible, while our goal on a secondary level is to reduce the total working time of all nurses. We measure the working time of each nurse each day by the time difference between one's work starting time and ending time of the day.

- Balance: after a complete nurse schedule is constructed, we would like to improve the schedule quality by balancing the workload among the staff, so that it would be fair for all nurses to have similar number of patients (or treatments) to take care of.
- Reallocate: The scheduling tool is also supposed to be robust, if any changes happen, the reallocation of a new schedule should be made in a short time with as few modifications to the original schedule as possible. These changes might be due to various causes, for instances new clients joining in, clients changing their preferred visitation dates or time windows, and nurse staff's availability might also affect the existing schedule to some extent. Some of these changes are known one week before, but in some urgent cases, which is not rare, patients may call in a short time to cancel or change the visit time, or nurses may call in sick just before the plan is to start. In such cases, especially the last minute cases, a new schedule should be generated in a very short time, without changing the original schedule too much.

- Treatments and Patients:
  - Treatment natures: a treatment or a nurse visitation is described as a medical activity to be performed at a proper time on an appropriate day. Different treatments include washing, cleaning, medication prescription, injections and so on. The duration of each treatment is fixed and given in advance, but the start time of each visit can be varied within a patient-specified time window.
  - Day preference and time windows: each patient might have multiple treatments within a week, and patients can indicate on which day(s) they are expecting a certain treatment, e.g. no treatments on Thursday. Each patient can also specify a time interval for a certain treatment, within which he prefers being visited, e.g. cleaning is to start between 10:00 and 11:00. Note that the time windows imposed here are hard constraints, which means, if the nurse arrives earlier than the given lower bound of the time window, she has to wait.
  - Inter-visit day difference: for some patients who need several visits per week, a minimum inter-visit day difference between some pair of visits should be retained, e.g. some injections must be given twice a week with at least 3 days' break in between. In the literature, most of scheduling processes are performed on a daily basis and the treatment dates are fixed in advance. For example in BegMilWea97, a two-visit-per-week patient will be assigned to a Monday and Thursday or a Tuesday and Friday where possible before the optimization process, but as we notice such fixed assignments of treatment days simplified the problem, but lost scheduling flexibility and efficiency. In our project we allow the patients to define their preferred date for each

treatment and an inter-visit day difference between pairs of treatments, schedule on a weekly basis, and assign each treatment to an appropriate date dynamically adapting both the day preference and the inter-visit day difference into consideration.

- Always the same nurse: each patient should, if possible, be visited by the same nurse. This is a crucial point for the service quality. The reason for this is to provide a reasonable continuity for the elderly, where they should not have to see a new face at each visit.

- Nurses:
  - Qualification: the medical services are provided by basically two types of nurses differentiated by their qualifications, namely professional nurses who own a medical certificate, and home care aides who receive only a short-term vocational training. We refer to the latter as partial nurses. It is assumed that a professional nurse is able to perform all treatments of the clients, while certain treatments cannot be carried out by partial nurses, for example injection of insulin.
  - Working date and time: Nurses' everyday work starts at 7 am in the headquarter of the organization with a car, and the car must return to the headquarter by the end of the day's work. Each nurse has the right to choose on which day (normally working days from Monday to Friday) and at most how many hours on this day she's willing to work, as well as the weekly maximum working hours she prefers. However, according to the organization's legal regulation, the daily working hour of each nurse cannot exceed 6 hours, while the total weekly working hours for each nurse cannot be over 30 hours. By working time not only the treatment time a nurse spends at a patient is counted, all the deadhead time, i.e. the driving time that she spends in the vehicle from one place to another, and the idle time if the next task cannot be taken immediately, these are all counted as a nurse's working time. Note that the maximum working time restriction imposed in our project is "hard" constraint, which means no overtime is allowed for the nurses.
- Goals and Objectives:
  - Our goal of primary importance is to reduce the operating cost, especially the number of the staff members required to carry out the tasks, while on a subsidiary level, minimize the total working time of all nurses. A secondary goal is to improve the schedule quality by balancing the workload among staff.

Rönnqvist et al. 2005 introduces a decision support software developed to aid the staff planner for daily schedules at a home health care organization in Sweden. Various practical constraints like staff competence, time windows for visits, break for meals etc. A mixed integer programming formulation is given using set partitioning model. For a solution method, they make use of repeated matching algorithm, which

approximates the staff scheduling problem into a matching problem, and then solves the resulting matching problem with exact method or heuristic method called repeated assignment. Numerical results are reported on 4 different instances up to 123 daily visits and 20 nurses, and the solution time is within a couple of minutes.

In Cheng & Rich 1998, they formulate the home health care service scheduling problem into a multiple depot vehicle routing problem with time windows, both fulltime and parttime nurses are considered. The multiple depot is because each nurse starts and ends their daily service trip from their home. The lunch break problem is considered by adding an additional lunch node into the scheduling graph. Two mixed integer programming models based on double-indexed and triple-indexed formulations are presented and solved using commercial solver CPLEX. It is reported that the largest instance that CPLEX is able to solve contained 2 regular nurses and 2 parttime nurses and 10 patients, using a 143 MHz machine. It is also interesting to note that the triple-indexed model is more than 10 times faster than the double-indexed model. A two-phase heuristic is also developed, using randomized greedy algorithm to construct a not necessarily feasible solution in the first phase and improve it by a problem-specific local search in the second phase.

In Begur, Miller and Weaver 1997, a spatial decision support system was developed, to schedule and route home health care nurses in Birmingham, Alabama, US. The system integrates geographic information system with scheduling heuristics and databases, and it is reported an over 20,000 US Dollars saving annually for travel expenses and scheduling preparation, and it helps improve the balance of work among nurses. A saving-type route-building heuristic is proposed and described, and a route improvement is done manually through a visual interactive system.

In Fahle 2001, it is stated that the challenge of finding good working plans for home health care is the combination of vehicle routing and staff rostering aspects. Although the MIP model is not explicitly given, a set partitioning model is mentioned, and for solution approach the author has proposed constrained programming based column generation. But no computational results can be seen.

As far as we are aware of, none of the presented models are completely applicable to our problem, mainly for some or all of the following reasons. Our schedule is on a weekly basis, and the inter-visit day different must be integrated into the optimization process. Besides, not only the daily maximum working time but also the total weekly maximum working time can be imposed for each nurse, and each Nurse specifies their individual preference on working days in advance, and so do the patients too.

4.1.2 Mathematical Model for Mobile Nurse Scheduling

The mathematical model will be introduced step by step as it follows.

4.1.2.1 The graphical model and sets

The graphic model is shown below:

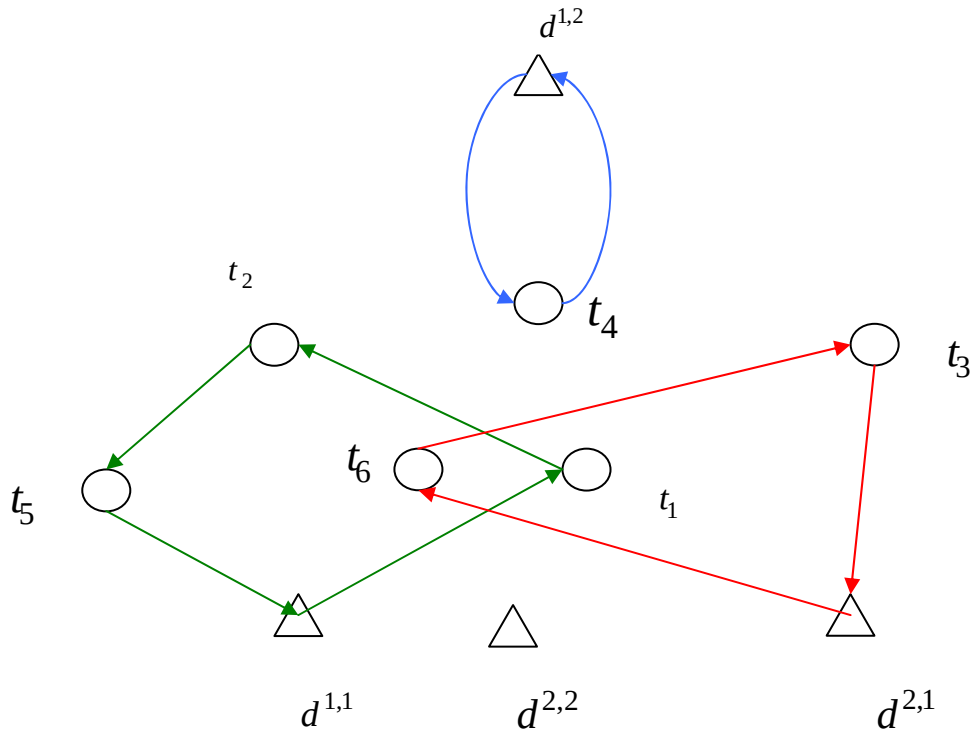


Figure 4.1: Graphic model for Mobile Nurse Scheduling Problem. Triangles stand for depot nodes, where node $d^{i,j}$ stands for a nurse index i and on day j, note that some nurse may be totally free on some day. Circles stand for treatment nodes with its index. Each tour must start and end at a depot node.

We denote the set of all nodes with N. There are two types of nodes in our graphical model, namely depot nodes and treatment nodes.

- N_{dp} - a depot node represents the depot where a nurse should start and end at.
- N_{trm} - a treatment node represents a treatment that should be taken by one of the nurses.

Similarly we denote the set of all arcs with A, and distinguish three types of arcs in our graphical model, pull-out arcs, deadhead arcs and pull-in arcs.

- $A_{pullout}$ - a pull-out arc starts from a depot node and ends at a treatment node.

- $A_{deadhead}$ - a deadhead arc starts and ends both at a treatment node, denotes the trip from one patient to another patient.
- A_{pullin} - a pull-in arc starts from a treatment node and ends at a depot node.

Apart from the sets that are visible from the graph, there are some other sets outside the graph. Like

- the set of nurses, which we denote as K,
- and we denote the set of days as $D := \{1,...,5\}$, standing for Monday to Friday. Note that we consider per nurse per day as a depot in our model.
- $R \subseteq N_{trm} \times N_{trm}$ is used to denote the set of pairs of related treatments, $(i, j) \in R$ means that treatments *i* and *j* are related and must have some days' difference between each other.

4.1.2.2 Parameters

The parameters that will be used in our MIP model are listed below:

- c_{start}^{k} - in our mobile nurse scheduling project, the cost parameter is not attached to each arc as normal. Here we will have a starting cost for each nurse $k \in K$.
- $u_{i,j}^{k,d}$ - the upper bound parameter, or the flow capacity parameter for each arc $(i, j) \in A$, under a nurse $k \in K$ and a day $d \in D$. For each arc $(i, j) \in A$, $u_{i,j}^{k,d}$ is always binary valued, since each treatment node must be visited once and only once, and each depot can only deploy one unit of flow. We also use the parameter $u_{i,j}^{k,d}$ to control the feasibility of some arcs in preprocessing, e.g.

  i. If a treatment node $i \in N_{trm}$ cannot be taken by a nurse $k \in K$, then we set all the arcs that are incident to node *i* to be infeasible by assigning their upper bounds to 0:

$$u_{i,j}^{k,d} := 0 \quad \forall j \in N, \forall d \in D$$

$$u_{j,i}^{k,d} := 0 \quad \forall j \in N, \forall d \in D$$

  ii. If a treatment node $i \in N_{trm}$ cannot be taken on a day $d \in D$, similarly we

set

$$u_{j,i}^{k,d} := 0 \quad \forall j \in N, \forall k \in K$$

$$u_{i,j}^{k,d} := 0 \quad \forall j \in N, \forall k \in K$$

iii. If a nurse $k \in K$ is not possible to work on day $d \in D$, then all arcs on this layer will be set to infeasible:

$$u_{i,j}^{k,d} := 0 \quad \forall (i, j) \in A$$

- $\underline{t_i}$ and $\overline{t_i}$ - the lower bound and the upper bound of the time window for a treatment node $i \in N_{trm}$.
- $\delta_{i,j}$ - the duration from the start of treatment *i* until the start of treatment *j*, if treatment *j* is taken by the same nurse on the same day subsequently after treatment *i*. It consists of two parts, $\delta_{i,j} = \delta_i^{service} + \delta_{i,j}^{deadhead}$, where
  - $\delta_i^{service}$ - the duration of the treatment node $i \in N_{trm}$.
  - $\delta_{i,j}^{deadhead}$ - the duration of the deadhead trip from node *i* to node *j*.
- $MAX_{daily}^{k,d}$ - the daily maximum working time of nurse $k \in K$ on day $d \in D$.
- MAX_{weekly}^{k} - the weekly maximum working time of nurse $k \in K$.
- $\sigma_{i,j}$ - is used to denote the necessary day difference between two related treatments *i* and *j*, when $(i, j) \in R$. The value of $\sigma_{i,j}$ is normally restricted as $1 \le \sigma_{i,j} \le 3$.

4.1.2.3 Variables, Objective and Constraints

The decision Variables are listed below:

- $x_{i,j}^{k,d} \in \{0,1\}$ - the flow variable over an arc (i, j) with $(i, j) \in A$, $k \in K$ for a nurse, $d \in D$ for a day.
- $t_i \in \mathbb{Z}^+$ - the starting time of all treatment nodes $i \in N_{trm}$.
- $s^k \in \{0,1\}$ - the binary indicator to determine whether nurse $k \in K$ has been

deployed.

- $\tau^{k,d} \in \mathbb{N}$ - denotes the working time of the nurse $k \in K$ on day $d \in D$. Its value equals the ending time of the tour of nurse $k \in K$ on day $d \in D$ minus the starting time 420. If the nurse on that day does not start any tour, the parameter $\tau^{k,d}$ should equal to 0.

Objective function is given as follows:

We have basically two goals to achieve,

- Firstly, imprecisely speaking, we want to use as few nurses as possible to take care of all the patients. Each nurse deployment is given a large value of starting cost, and we want to minimize the total starting cost.
- Secondly at a subsidiary level, we wish to shorten the nurse's daily working hours. We assume each nurse starts their work at 7 am, and a nurse's daily working hour is defined by the difference between the starting time and the ending time of the day's work.

$$\min \quad \sum_{k \in K} c_{start}^{k} \bullet s^{k} + \sum_{\substack{k \in K \\ d \in D}} \tau^{k,d} \tag{4.1}$$

Subject to the constraints as below:

- Bundle constraint, every treatment will be taken exactly once,

$$\sum_{\substack{k \in K \\ d \in D \\ i \in N}} x_{i,j}^{k,d} = 1 \quad \forall j \in N_{trm} \tag{4.2}$$

- Flow capacity, so the flow cannot exceed the upper bound on each arc,

$$x_{i,j}^{k,d} \le u_{i,j}^{k,d} \quad \forall (i,j) \in A, \forall k \in K, \forall d \in D \tag{4.3}$$

- Flow conservation, the inflow of each patient node equals its outflow,

$$\sum_{j \in N} x_{j,i}^{k,d} = \sum_{j \in N} x_{i,j}^{k,d} \quad \forall i \in N_{trm}, \forall k \in K, \forall d \in D \tag{4.4}$$

- Depot capacity constraint, in our case each depot can deploy at most one unit of flow, and we should guarantee each deployed nurse returns to the depot at the end of the work,

$$\sum_{j \in N_{trm}} x_{i,j}^{k,d} \le 1 \quad \forall i \in N_{dp}, \forall k \in K, \forall d \in D \tag{4.5}$$

$$\sum_{j \in N_{trm}} x_{i,j}^{k,d} - \sum_{h \in N_{trm}} x_{h,i}^{k,d} = 0 \quad \forall i \in N_{dp}, \forall k \in K, \forall d \in D \tag{4.6}$$

- Time window range, which the starting time of each treatment should obey,

$$\underline{t_i} \le t_i \le \overline{t_i} \quad \forall i \in N_{trm} \tag{4.7}$$

- Starting time compatibility for deadhead trips, two consecutive treatments should

satisfy the time compatibility constraint:

$$t_i + \delta_{i,j} + M \bullet (x_{i,j}^{k,d} - 1) \le t_j \quad \forall (i,j) \in A_{deadhead}, \forall k \in K, \forall d \in D \tag{4.8}$$

with a sufficiently large M.

- Starting time compatibility for pull-out trips, it is supposed that every nurse starts their daily work at 7 am (420 minutes after 0:00), so the first treatment of the tour should satisfy:

$$420 + \delta_{i,j} + M \bullet (x_{i,j}^{k,d} - 1) \le t_j \quad \forall i \in N_{dp}, j \in N_{trm}, \forall k \in K, \forall d \in D \tag{4.9}$$

with a sufficiently large M.

- Starting time compatibility for pull-in trips, a day's work ends in the depot where the nurse started. The total working time of a nurse on this day is measured by this ending time minus their starting time 7am (420). In such a way with the constraint described below, we can properly define the total daily working time $\tau^{k,d}$. Note that in the following inequality we have only determined the lower bound for $\tau^{k,d}$, but we guarantee that $\tau^{k,d}$ always arrive at its lower bound by placing it in the minimization objective function.

$$t_i + \delta_{i,j} + M \bullet (x_{i,j}^{k,d} - 1) - 420 \le \tau^{k,d} \quad \forall i \in N_{trm}, j \in N_{dp}, \forall k \in K, \forall d \in D \tag{4.10}$$

with a sufficiently large M.

- Maximum daily working time should not be exceeded,

$$\tau^{k,d} \le MAX_{daily}^{k,d}, \quad \forall k \in K, \forall d \in D \tag{4.11}$$

- Maximum weekly working time should not be exceeded,

$$\sum_{d \in D} \tau^{k,d} \le MAX_{weekly}^{k}, \quad \forall k \in K \tag{4.12}$$

- Determine the starting indicator s^k, whether nurse $k \in K$ has been started. Therefore we check each pull-out trip as follows:

$$s^k \ge \sum_{(i,j) \in A_{pullout}} x_{i,j}^{k,d}, \quad \forall k \in K, \forall d \in D \tag{4.13}$$

- Day difference between a pair of related treatment, we model this constraint in two steps:
  i. The former treatment *i* should not be started too late, so that the latter treatment *j* can be taken during the week, i.e. the former treatment *i* should be started latest on $(5 - \sigma_{i,j})$:

$$\sum_{\substack{d\in\{1,\ldots,5-\sigma_{i,j}\}\\ h\in N\\ k\in K}} x_{h,i}^{k,d} = 1, \quad \forall (i,j)\in R \text{ with } \sigma_{i,j} \tag{4.14}$$

ii. If the former treatment i is started on day $d \in (1,\ldots,5-\sigma_{i,j})$, we should guarantee that the latter treatment j can be started only earliest on the day $d' \geq (d+\sigma_{i,j})$, and no earlier.

$$x_{h,i}^{k,d} + \sum_{\substack{d'\in\{1,\ldots,d+\sigma_{i,j}-1\}\\ h'\in N\\ k'\in K}} x_{h',j}^{k',d'} \leq 1, \quad \forall (i,j)\in R \text{ with } \sigma_{i,j}, \forall h\in N, \forall k\in K, \forall d\in D \tag{4.15}$$

To sum up we have the following model for mobile nurse scheduling problem:

$$\begin{aligned} &\min \quad (4.1)\\ &\text{s.t.} \quad (4.2 - 4.15)\\ &x\in\{0,1\}^{|A|\times|K|\times|D|}, t\in\mathbb{Z}_+^{|N_{trm}|}, s\in\{0,1\}^{|K|}, \tau\in\mathbb{N}^{|K|\times|D|} \end{aligned} \tag{4.16}$$

4.2 School Taxi Routing for Handicapped Pupils

An overview and some terminologies used in the mobile nurse scheduling project will be presented in section 4.1.1, and followed by the mathematical model description in the later sections.

4.2.1 Project Description for School Taxi Routing

In Germany, especially in the rural counties, routing and scheduling public transportation for pupils has aroused more and more interest. As mentioned in Fuegenschuh 2005, about half to two third of pupils in rural areas take a bus to get to school. Most of them are integrated into the public bus system. A minority is transferred by special purpose school buses. In either way the county administration is responsible for their transfer. In order to reduce cost in both transportation and organization, and to save tax money, developing a computerized optimization system for school bus routing and scheduling is currently in more and more demand. In this work, we focus on the specific field of transporting handicapped pupils with taxis.

As is also noticed by Braca et al. 1997, the routing of Special Education (or handicapped, or Special Ed for short) pupils is fundamentally different from routing of General Education (General Ed) pupils. The General Ed pupils usually walk and gather to their neighborhood bus stop in the morning and wait for a bus to take them to school. In the afternoon a bus takes them from school back to their neighborhood bus stop. However, Special Ed pupils are usually picked up and dropped off directly at

their homes. Besides, in many cases Special Ed pupils must traverse a very long distance to get to one of the few schools that provide for their special needs. Furthermore, vehicles with special designed spaces for different types of wheelchairs will be needed for some Special Ed pupils.

In our project, various types of taxis owned by different taxi companies are available. There are in total about a hundred different types of cars, including some cars specially designed for wheelchairs. However, most of cars differ only in minor aspects and therefore can be grouped into 5 to 11 classes. The main differences between the cars are the capacity and costs. For instance, the passenger capacity of the cars differs from 2 to 28 persons. Wheelchairs are not admissible on some cars at all, and some cars can take from 1 up to 4 wheelchairs. Note that there are two types of wheelchairs, namely normal wheelchair and electronic wheelchair. The difference between them is that the normal wheelchair can be folded so that it takes much less spaces, while the electronic wheelchair cannot, so it takes about twice the space as a normal one. In practice it is safe to assume that the space for one electronic wheelchair can be replaced by 2 normal wheelchairs, and vise versa. Cars with different capacities mean different costs. The costs for using a taxi are in general composed of two parts, the basic price when a car is called to start, and the driving cost per kilometer. In practice some pupil might need a companion to go together with them in the car to school, the cost for a companion person is usually a fixed value independent of different vehicles and driving distance, so the cost for companion can be ignored in our optimization process. Further characteristics such as speed are not distinguished here for the cars.

In the morning, the pupils are picked up directly at their homes and transferred to school at a reasonable time. When their classes finish in the afternoon, a fleet of taxis are sent to bring them to their home. Considering the morning situation, when a pupil is picked up, the car does not have to go immediately to school but it can try picking up some more pupils until some limit is reached before heading for school. Note that we don’t restrict pupils from different schools to be on the same car at the same time, and we have found in our experiment that allowing pupils from different schools on board makes the routing more complicated but also more flexible, which results in further potential savings.

Considering the time constraints during routing, several aspects come into the scene. The driving time and distances between each pupils and schools are given, and is assumed to be invariant under different taxis. When a car stops at a pupil’s home, a pickup time is specified for each pupil to get in the car. A pupil without wheelchairs usually takes one minute to get on, while a pupil with wheelchair must take longer time, about 3 to 5 minutes. The drop-off time at school also depends on how many wheelchairs on board. For the task of transporting pupils to school, a maximum driving time from home to school is specified for each pupil and must be complied with. There is no explicit time window for each pupil, however pupils should arrive at school within a given time window, usually 5 to 15 minutes before the school starts,

coupled with the maximum driving time, the time interval during which they should be picked up is implied.

Note that in our work so far we have only considered the morning peak, i.e. the transportation from home to school. We are doing so for two reasons, firstly, the optimization process for the afternoon peak, i.e. picking up a set of pupils at school and delivering them back home one by one, is essentially the same, since we can first assume we are shipping the pupils from home to school before the school finish time, reverse the route then we will have a route from school back to pupils' home, and the arrival time of the pupil at home is exactly the reflection of the time that the pupil is assumed to be picked up at home with respect to the school finish time. The figure below shows the time reflection of the pupil arrival time with respect to the school finish time. The second reason that we didn't consider the afternoon peak at the beginning is that, the afternoon peak requires usually less vehicles as it does in the morning peak. As is also noticed by Fuegenschuh 2005 that, this is caused by the fact that nearly all schools start at more or less the same time, in a narrow interval around 8 o'clock. The afternoon peak is much lower since the schools release their pupils at different times.

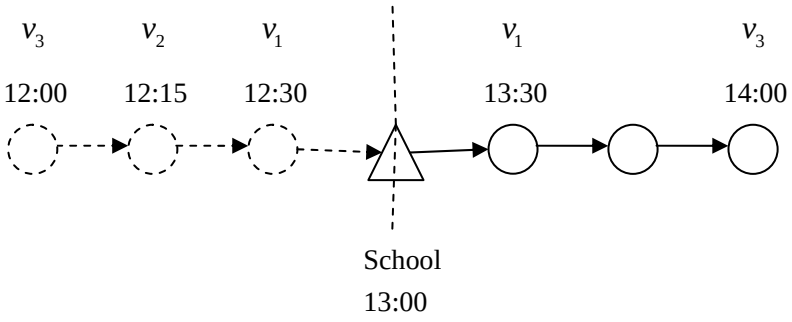


Figure 4.2: Solving the afternoon peak problem by reflection of the morning peak process.

The objective is to save the practical operational costs, mainly to reduce the number of the vehicles needed (due to the high basic cost of each taxi), and the total driving distance of the all vehicles. Note that the driving distance from the taxi depot to customers or from customers to the depot is not considered.

4.2.2 Mathematical Model for School Taxi Routing

The mathematical model will be introduced step by step as it follows.

4.2.2.1 The graphical model and sets

The graphic model is shown below:

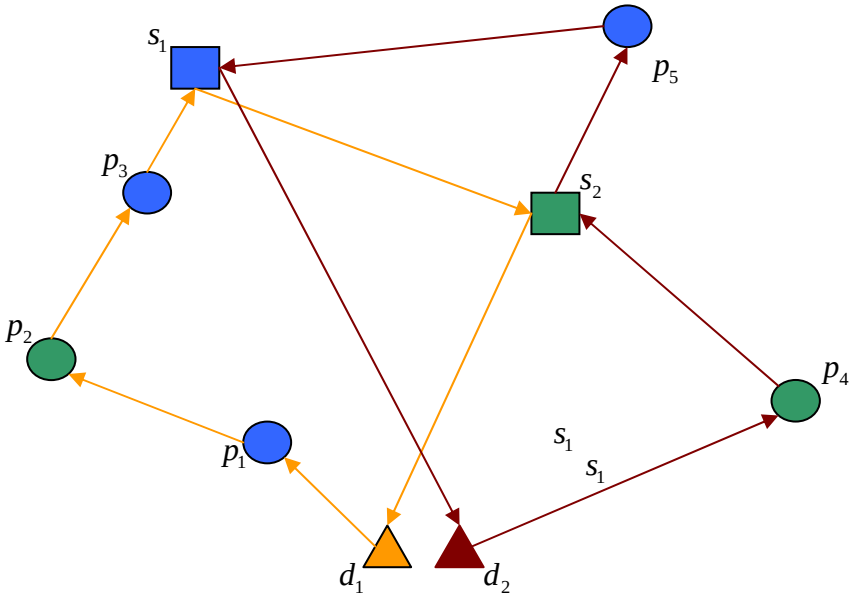


Figure 4.3: Graphic model for school taxi routing problem. Triangles stand for depots, circles for pupils, squares for schools. Pupils of the same school have the same color of as their associated school. Two tours are shown.

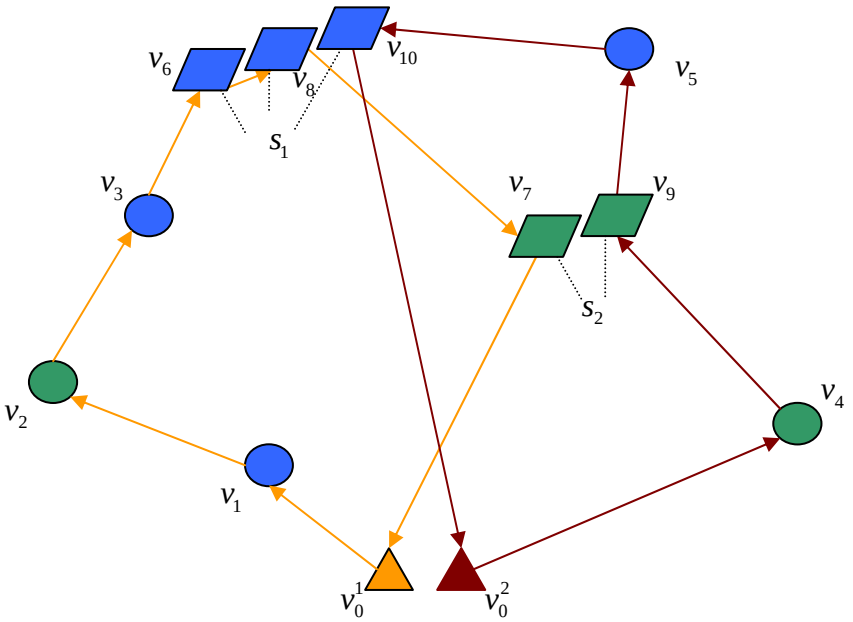


Figure 4.4: Transformed graphical model into VRPPD formulation for the school taxi routing problem. Each school is decomposed so that each pupil v_i has his

unique corresponding delivery node v_{n+i}. E.g. pupil v_1 is dropped off at school v_6.

As is shown in the graph above, We name each pupil from the graph as the *Pickup Node*, and the set of all pickup nodes are denoted $N_{pk} := \{v_i \mid i = 1,...,n\}$. In order to adapt our school taxi routing problem into the framework of VRPPD, each school is decomposed into several nodes at the same geographic location, and each node for exactly one of its associated pupil. The decomposed nodes from schools are named *Delivery Nodes*, and the set of all delivery nodes are denoted as $N_{dlv} := \{v_i \mid i = n+1,...,2n\}$. Under the decomposition, it is possible to impose a one-to-one correspondence is between each pickup node v_i and the delivery node v_{n+i}, so that each corresponding pair should be visited exactly once by the same vehicle. A corresponding pair is called a request. Furthermore, the vehicle should visit each pickup node before its corresponding delivery node, and the delivery time should not exceed a certain maximum duration.

For notation convenience we introduce the set of *service nodes* $N_{srv} := N_{pk} \cup N_{dlv}$ to be a collection, distinguishing from the set of *depot nodes* N_{dp}. Depot nodes (like v_0 in the graph) represent where the vehicles start and end. Each vehicle possesses its own depot, and defines its own layer in our multicommodity flow formulation. The set of all nodes is denoted as $N := N_{dp} \cup N_{srv}$.

The set of all arcs A consists of 3 types of arcs as usual multicommodity models,

- $A_{pullout}$ - a pull-out arc starts from depot and ends at a pickup node, note that delivery nodes are always visited after pickup nodes;
- $A_{deadhead}$ - a deadhead arc connects between two service nodes;
- A_{pullin} - a pull-in arc starts from a delivery node and ends at a depot, note that a pickup node must be followed by its corresponding delivery node within the same tour.

We further denote K the set of all available vehicles.

4.2.2.2 Parameters

The parameters that will be used in our MIP model are listed below:

- $c_{i,j}^{k}$ - the cost parameter for arc $(i, j) \in A$ with vehicle $k \in K$. The cost on the deadhead arc $(i, j) \in A_{deadhead}$ is given as the driving cost from node *i* to *j* with vehicle *k*; the cost on the pull-out arc equals the starting cost of vehicle *k*; the pull-in arcs have zero cost on them, so do the "virtual" deadhead arcs between two delivery nodes of the same school.
- $u_{i,j}^{k}$ - the flow upper bound on the arc $(i, j) \in A$ with vehicle $k \in K$. It is binary valued, and inadmissible arcs, are eliminated by setting this value to 0. For example, some nodes with wheelchair cannot be taken by some vehicles, in this case the upper bound of their incident arcs are set to 0 in preprocessing phase.
- $\underline{t_i}$ and $\overline{t_i}$ - the lower bound and the upper bound of the time window for a delivery node $i \in N_{delivery}$.
- $\delta_{i,j}$ - the time duration required between the service node *i* and the service node *j*, if *j* is taken directly after *i*. It consists of two parts, $\delta_{i,j} = \delta_i^{service} + \delta_{i,j}^{deadhead}$, where
  - $\delta_i^{service}$ - the service duration of a pickup node $i \in N_{pk}$ means the time that a pupil needs to get in the car. It takes 5 minutes for pupils with wheelchair, and 1 minute for pupils without wheelchair. For a delivery node $i \in N_{dlv}$, the service duration is set to 0.
  - $\delta_{i,j}^{deadhead}$ - the duration of the deadhead trip from node *i* to node *j*. Note that arcs between two delivery nodes representing the same school has zero deadhead duration, since the two nodes are at the same location and the pupils get off the car at the same school simultaneously; for all other arcs that end at a delivery node, the deadhead duration is their driving duration plus the get-off time δ_j^{getoff} at school *j*, so that the get-off time at school *j* is imposed on its inflow arcs.

$$\delta_{i,j}^{deadhead} = \begin{cases} 0 \quad \text{if } i, j \in N_{dlv}, \text{ and } i, j \text{ from the same school} \\ \delta_{i,j}^{deadhead} + \delta_j^{getoff}, \text{ elsewise when } j \in N_{dlv} \end{cases}$$

  The δ_j^{getoff} is fixed to be 5 minutes for all schools.

- δ^i_{dlvmax} - the maximum delivery duration for a pickup request $i \in N_{pk}$, i.e. the maximum driving time to school for pupil *i*.
- C^k_{psg} - the passenger capacity of vehicle $k \in K$.
- C^k_{wch} - the maximum wheelchair load of vehicle $k \in K$.
- p_i - the passenger load at service node $i \in N_{srv}$. It is positive at each pickup node, and negative at each delivery node. Normally each pickup node has only one pupil, but some pupil might have one company coming along together in the car.
- w_i - the wheelchair load at the service node $i \in N_{srv}$. Each normal wheelchair is counted as 1 unit, while each electronic wheelchair is counted as 2 units of load.

4.2.2.3 Variables, Objective and Constraints

The decision Variables are listed below:

- $x^k_{i,j} \in \{0,1\}$ - the binary flow variable similar as before, equal to 1 if arc $(i, j) \in A$ is taken by a vehicle $k \in K$, and 0 otherwise;
- $t_i \in \mathbb{Z}^+$ - the starting time for each service node $i \in N_{srv}$;
- $\pi^k_i \in \mathbb{N}$ - the passenger load of the vehicle after the node $i \in N$ is serviced, and we set the initial passenger load at the depot to 0;
- $\omega^k_i \in \mathbb{N}$ - the wheelchair load of the vehicle after the node $i \in N$ is serviced, similarly the initial wheelchair load at depot is set to 0. Note that for simplification reason, each normal wheelchair is counted as 1 unit load, while each electronic wheelchair is counted as 2 unit load.

The objective function is formulated as below. Note that we not only minimize the operating cost, but also consider the service quality by sending the pupils to school with as little time as possible:

$$\min \quad \sum_{\substack{(i,j)\in A \\ k\in K}} c^k_{i,j} \bullet x^k_{i,j} + \varepsilon \cdot \sum_{i \in N_{pk}} (\overline{t_{i+n}} - t_i) \tag{4.17}$$

Subject to the constraints:

- Bundle constraint, each node, i.e. each pupil or school node, is to be visited exactly once by one vehicle:

$$\sum_{\substack{i\in N\\ k\in K}} x_{i,j}^{k} = 1 \quad \forall j \in N_{srv} \tag{4.18}$$

- Flow capacity, so the flow cannot exceed the upper bound on each arc,

$$x_{i,j}^{k} \le u_{i,j}^{k} \quad \forall (i,j) \in A, \forall k \in K \tag{4.19}$$

- Flow conservation, the inflow of each service node, i.e. pupil node or school node, equals its outflow,

$$\sum_{h\in N} x_{h,i}^{k} - \sum_{j\in N} x_{i,j}^{k} = 0 \quad \forall i \in N_{srv}, \forall k \in K \tag{4.20}$$

- Depot capacity constraint, in our case each depot contains only one vehicle, therefore can deploy at most one unit of flow, and we should guarantee each deployed vehicle returns to the depot at the end of the schedule,

$$\sum_{j\in N_{srv}} x_{i,j}^{k} \le 1 \quad \forall i \in N_{dp}, \forall k \in K \tag{4.21}$$

$$\sum_{j\in N_{srv}} x_{i,j}^{k} - \sum_{h\in N_{srv}} x_{h,i}^{k} = 0 \quad \forall i \in N_{dp}, \forall k \in K \tag{4.22}$$

- Each pupil and the corresponding school should be served by the same vehicle,

$$\sum_{i\in N} x_{i,j}^{k} - \sum_{i\in N} x_{i,n+j}^{k} = 0 \quad \forall j \in N_{pk}, \forall k \in K \tag{4.23}$$

- Time window range, for each delivery node, i.e. school node, since there are no explicit time interval for pickup nodes,

$$\underline{t_i} \le t_i \le \overline{t_i} \quad \forall i \in N_{dlv} \tag{4.24}$$

- Starting time compatibility for deadhead trips, two consecutive service nodes should satisfy the time compatibility constraint:

$$t_i + \delta_{i,j} + M \bullet (x_{i,j}^{k} - 1) \le t_j \quad \forall (i,j) \in A_{deadhead}, \forall k \in K \tag{4.25}$$

  with a sufficiently large M .

- Time restriction for corresponding pickup and delivery nodes, the delivery node should be visited after the pickup node, but within the maximum delivery duration imposed by each pupil,

$$t_i \le t_{n+i} \quad \forall i \in N_{pk} \tag{4.26}$$

$$t_i + \delta_{\mathrm{dlvmax}}^{i} \ge t_{n+i} \quad \forall i \in N_{pk} \tag{4.27}$$

- Vehicle load constraints, two types of loads have to be taken into account, namely passenger load and wheelchair load:

a) Passenger load compatibility

$$\pi_i^k + p_j + M_p \bullet (x_{i,j}^k - 1) \le \pi_j^k \quad \forall (i,j) \in A_{deadhead} \cup A_{pullout}, \forall k \in K \tag{4.28}$$

with sufficiently large M_p, and the initial passenger load is set to:

$$\pi_i^k = 0 \quad \forall i \in N_{dp}, \forall k \in K \tag{4.29}$$

And passenger load at each node is restricted according to the vehicle capacity:

$$\pi_i^k \le C_{psg}^k \quad \forall i \in N_{srv}, \forall k \in K \tag{4.30}$$

b) Wheelchair load compatibility

$$\omega_i^k + w_j + M_w \bullet (x_{i,j}^k - 1) \le \omega_j^k \quad \forall (i,j) \in A_{deadhead} \cup A_{pullout}, \forall k \in K \tag{4.31}$$

with sufficiently large M_w, and the initial wheelchair load is set to:

$$\omega_i^k = 0 \quad \forall i \in N_{dp}, \forall k \in K \tag{4.32}$$

And wheelchair load at each node is restricted according to the vehicle wheelchair capacity:

$$\omega_i^k \le C_{wch}^k \quad \forall i \in N_{srv}, \forall k \in K \tag{4.33}$$

All in all the mathematical model for the school taxi routing problem is given as below,

$$\begin{aligned} &\min \quad (4.17) \\ &\text{s.t.} \quad (4{,}18 - 4.33) \\ &x \in \{0,1\}^{|A|\times|K|}, t \in \mathbb{Z}_+^{|N_{srv}|}, \pi \in \mathbb{N}^{|N|\times|K|}, \omega \in \mathbb{N}^{|N|\times|K|} \end{aligned} \tag{4.34}$$

4.3 Cyclic Locomotive Scheduling

This project is carried out together with our industrial Deutsche Bahn. Deutsche Bahn AG (DB) is the largest German railway company with 216,000 employees and a turnover of 25 billion Euros in 2005. In this section we give details of the integrated scheduling problem and introduce the terminology used at DB.

- **Wagon:** A *wagon* is a rolling stock for freight transport. The wagons have to be delivered between a source and a destination point (goods station) within the network. Large customers produce and/or consume so much goods that they order whole trains. In these cases, the route of the wagons equals the route of the train. Smaller customers order individual wagons. Then the wagons of different customers are assembled to trains and pulled as a whole to an intermediate destination (a shunting yard), where the trains are disaggregated and reassembled

to new trains. The trains and the yards where the wagon transfer between trains are known in advance. When changing the starting time of the trains, one has to take care that these transfers still remain feasible.

- **Trains:** A *freight train* (also called *production trip*) consists of several *wagons*. Each train has a *start* and a *destination*, which are goods stations or railroad shunting yards. Also given are *starting times* and *arrival times*. These can be either fixed times or intervals, in which the start or the arrival has to take place. We assume that the trains start cyclical every 24 hours. The *trip duration* is the time difference between start and arrival. The average travel speed of freight trains is not as high as in passenger transport, especially at daytime, when passenger trains always have priority, such that some trips can last up to 3 days. The trains have different length and weight and thus require locomotives with sufficient driving power. A train is always pulled by a single locomotive. At the start a locomotive is attached to the train, and at the destination station it is detached (uncoupled). For both *coupling* processes, a certain train-dependent amount of time has to be taken into account (15 to 30 minutes). At this stage, technical checks and refueling of diesel locomotives are carried out.

- **Locomotives:** DB uses up to 30 different locomotives of several manufacturers. However, the differences between them are often minor, so they are grouped into 3 to 6 *classes* of similar locomotives. The main differences among the classes are the driving power of the engines, and the *traction* (i.e., the motor type, diesel or electrical). Electrical locomotives can only be used on electrical tracks, whereas diesel locomotives in principle can drive everywhere. However, diesel soots the electrical wires, so one wants to avoid their deployment on such tracks. Hence, it is only possible to assign such locomotives to trains that have a sufficient power and the right traction for the track.

- **Deadheads:** A locomotive is either active, i.e., pulling atrain, or deadheading, i.e., driving under its own power without pulling a train from the destination station of one train to the start of another train. The duration for a deadhead trip depends on the distance between these two points, and on the class of the locomotive (diesel and electrical might have to use different routes), but not so much on the network load (i.e., independent of daytime or nighttime), because it is assumed that a single locomotive can always be pushed through.

- **Goals and Objectives:** The main goal is to compute feasible starting and arrival times of the trains such that the wagons are transported as fast as possible from their start to the destination within the trains. At intermediate shunting stations the stopover of wagons should not exceed certain limits. The main objective is to reduce operating expenses, that is, to use as few locomotives as possible to pull all trains and, on a subordinate level, to schedule the locomotives in such a way that the deadhead trips are as short as possible.

We summarize the new features that will bring to our mathematical model and heuristic implementation as follows:

- **Cyclic departures of the trains:** Instead of starting and terminating at a fixed depot, the locomotives are scheduled in a cyclic fashion every 24 hours.
- **Network-load dependent travel times:** The driving duration of each trip is variable depending on its starting time and the network traffic load in that hour. The time window is imposed on the arrival time of each train, therefore the variable trip duration separates the train starting time interval into a multiple time window.
- **Wagon transfers between trains:** There are wagons that need to be transferred from one train to another at a shunting yard, which restricts the starting time difference between the two related trains within some interval. It is also our objective of first priority is to minimize the missed wagon transfers. The larger the time window that we allow for the trains, the more missed wagon transfers can be caught up with. More discussion of dealing with the wagon transfers and other features will be presented in section 5.8.3.

A detailed description of this project and its mathematical model can be found in Fuegenschuh, Homfeld et. al. 2006 and Fuegenschuh, Yuan et al. 2006.

4.4 Multi-Depot Locomotive Scheduling

This work presented here is based on my internship during the summer 2005 by Siemens AG in Munich. Our task was to develop a software to help our customer, a middle-large size logistics company, to schedule their locomotives to their weekly timetabled freight trains. Before we start, we will first have a close look over some details of the problem setting:

- The shipments (or trains): what to be transported are mainly industrial bulk goods. And the companies who require or supply them usually order a whole freight train, consisting of several wagons of different total weight. Every week the logistic company will come up with a complete list of trains for the coming week, it is our task to assign a fleet of locomotives to pull all these trains. For each train, a fixed start and a specific destination station is given, and the weight of each train is also known in advance. Note that the weight of the train is important since that decides which locomotive combinations will be possible for it.

- Timetable of the trains: also given is a timetable, with the arrival deadlines of all trains on it. Some of the shipments are strict about punctuality, i.e. they have to arrive exactly at the time given by the timetable; while the other shipments are more flexible, their arrival time can vary within a range of time, usually several hours, around the given deadline. The duration of each shipment is also contained in the timetable, depending on different types of locomotives, the duration may

also be different.

- Locomotives: the logistic company owns about 50 locomotives of several manufacturers. However, the differences between them are minor, so they are grouped into 2 classes of similar locomotives. The main differences among the classes are the driving power of the engine (the service weight), and the speed. It is clear that a locomotive can be assigned to a train if it has sufficient driving power; if it does not, it is also possible to combine 2 locomotives of the same class to pull the train, if that suffices. It is called a coupling trip or twin combination for the latter case. The locomotives owned by the company are located in different garages of different cities. In order to keep the workload of each garage stable from week to week, it is required that, the number of locomotives of the same class should remain unchanged in each garage, after a week's service plan. Note that the locomotives are deployed from their garages at the beginning of the week, after serving all the shipments, they do not have to return to their own garages, the garages with locomotives of the same class can exchange with each other. But sometimes in the busy weeks, the company may need to lease some locomotives, in such case the leased locomotives will have to return to the exact depot where they start. In this article, both cases will be considered in detail.

- Objectives: the main objective is to reduce operating expenses, that is, to use as few locomotives as possible to pull all trains and, on a subordinate level, to schedule the locomotives in such a ways that the total trip length, including delivery trips and unloaded trips, is as short as possible.

Unlike the Cyclic Locomotive Scheduling introduced in the last section, one of the special feature of this project is the coupling trip, i.e. some trips are required to be taken by more than one locomotives.

The mathematical model differs from (3.18), the MILP model for Heterogeneous VRP with Time Windows, by introducing a new binary decision variable e as an flow indicator for each arc, and the binary variable x in (3.18) becomes integer, since the flow on each arc might be more than one due to the coupling trip feature. If e indicates there is flow on an arc, the flow x one this arc will be determine by a constant parameter which depends on how many locomotives will be needed for taking this arc. For the detailed mathematical model for this problem we refer the interested readers to Yuan 2007.

4.5 Preprocessing – Reducing Big-M Constraints

Before actually solving the model by branch-and-bound techniques, it is desirable to have a compact formulation with as few constraints and variables as possible. We are

also aware that the linear programming based branch-and- bound approach usually suffers from constraints having big-M-terms, i.e. the terms using the Miller-Tucker-Zemlin (MTZ) formulation, such as (4.8) in the mobile nurse scheduling project, or (4.25) in the school taxi routing project. In the following we seek possibilities to reduce redundant big-M constraints and possibly decrease the value of M.

We observe that the big-M constraints, e.g. (4.25), are attached with every deadhead trip, whereas many of them are redundant. Let $(i, j) \in A_{deadhead}$ be a deadhead arc, and $a^k_{(i,j)}$ denotes the deadhead trip from customer *i* to *j* with vehicle type *k*. We can partition the set of deadhead arcs $A^k_{deadhead}$ into 3 cases, i.e.

$$A^k_{deadhead} = A^k_{always} \dot{\bigcup} A^k_{dilemma} \dot{\bigcup} A^k_{never}$$

Where

- $A^k_{always} := \{a^k_{(i,j)} \mid \overline{t_i} + \delta^k_{(i,j)} \le \underline{t_j}\}$ -- denotes those deadhead arcs that are always feasible despite the choice of t_i and t_j.
- $A^k_{never} := \{a^k_{(i,j)} \mid \underline{t_i} + \delta^k_{(i,j)} > \overline{t_j}\}$ -- denotes those deadhead arcs that are always infeasible despite the choice of t_i and t_j.
- $A^k_{dilemma} := \{a^k_{(i,j)} \mid \overline{t_i} + \delta^k_{(i,j)} > \underline{t_j} \wedge \underline{t_i} + \delta^k_{(i,j)} \le \overline{t_j}\}$ -- denotes those deadhead arcs whose feasibility depends on the choice of t_i and t_j.

A graphical interpretation of the three classes of deadhead trips is presented in Figure 4.5 as follows:

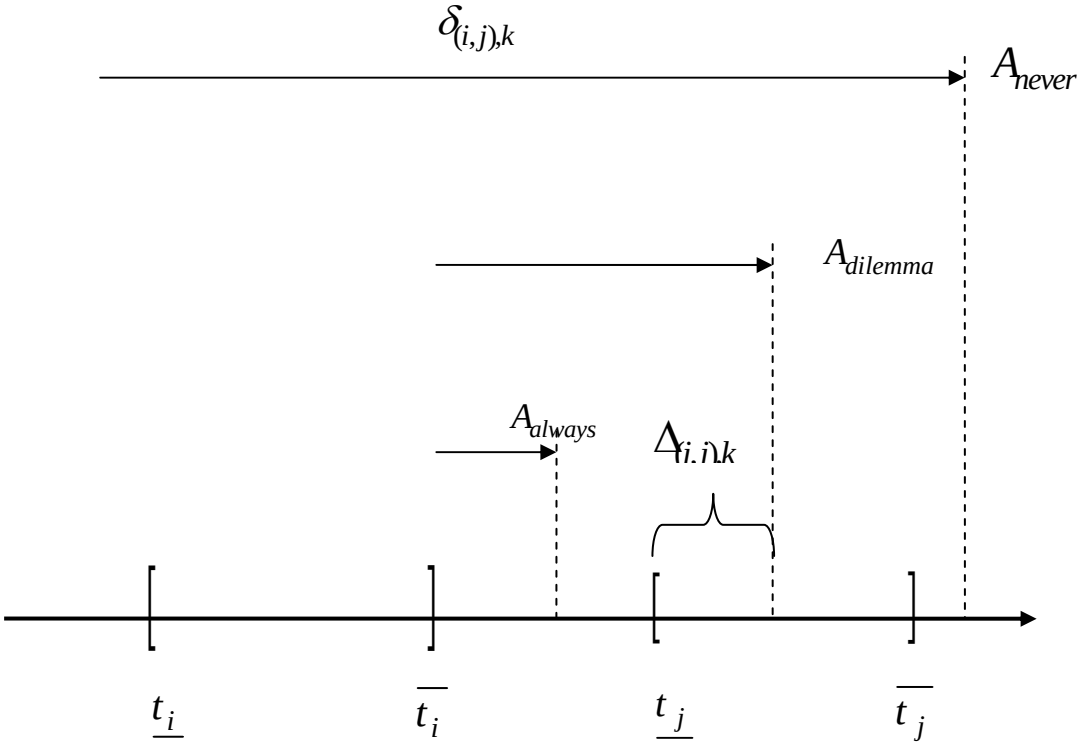


Figure 4.5: The classification of deadhead trips.

For the deadhead arcs in A^k_{always} and A^k_{never}, it is clear that the constraint (4.25) is redundant, since for $(i, j) \in A^k_{always}$ the constraint (4.25) always holds, and for $(i, j) \in A^k_{never}$, the left side of (4.25) is negative, the inequality holds if and only if $x^k_{i,j} = 0$, i.e. this arc can never be taken. So we can even set all the parameters $u^k_{(i,j)}$ to 0 for $(i, j) \in A^k_{never}$, so that variables $e^k_{(i,j)}$ and $x^k_{(i,j)}$ will be fixed to 0 at the beginning.

Now only the arcs in $A^k_{dilemma}$ remain to appear in constraint (4.25). This is already a great reduction in the number of big-M constraints. And we also notice an interesting fact that the larger the sizes of our time windows are, the more deadhead arcs will join the set of $A^k_{dilemma}$. If we restrict the time window to a fixed value, the set $A^k_{dilemma}$ will decrease to an empty set.

Can we do it even better? The answer is positive. We define a parameter representing the time closeness between adjacent customers as follows:

$$\Delta_{(i,j)}^{k} = \bar{t}_i + \delta_{(i,j)}^{k} - \underline{t}_j, \ \forall a_{(i,j)}^{k} \in A_{dilemma}^{k}, \ \text{ and } \ \forall k \in T$$

where the parameter $\Delta_{(i,j)}^{k}$ represents the *interval closeness* of the two customers i and j with vehicle type k. In the sequel we will estimate the bound of $\Delta_{(i,j)}^{k}$.

Let $\Delta_{(i,j)}^{k}$ be the interval closeness for a dilemma arc $a_{(i,j)}^{k} \in A_{dilemma}$, it follows from definition of dilemma arcs $A_{dilemma}$ that,

$$\bar{t}_i + \delta_{(i,j)}^{k} > \underline{t}_j \tag{4.35}$$

$$\underline{t}_i + \delta_{(i,j)}^{k} \le \bar{t}_j \tag{4.36}$$

which implies $\Delta_{(i,j)}^{k} = \bar{t}_i + \delta_{(i,j)}^{k} - \underline{t}_j \overset{(4.35)}{>} 0$,

and $\Delta_{(i,j)}^{k} = \bar{t}_i + \delta_{(i,j)}^{k} - \underline{t}_j \overset{(4.36)}{\le} \bar{t}_i - \underline{t}_i + \bar{t}_j - \underline{t}_j$.

To sum up, $\Delta_{(i,j)}^{k}$ has a bound:

$$0 < \Delta_{(i,j)}^{k} \le \bar{t}_i - \underline{t}_i + \bar{t}_j - \underline{t}_j \tag{4.37}$$

In other word, The interval closeness $\Delta_{(i,j)}^{k}$ for $a_{(i,j)}^{k} \in A_{dilemma}$ between two customers i and j with vehicle type $k \in T$, is non-negative and bounded upwards by the sum of time window sizes of the associated customer i and j.

Then the constraint (4.25) can be reformulated as follows:

$$\bar{t}_i - t_i + t_j - \underline{t}_j - \Delta_{(i,j)}^{k} \ge M \bullet (x_{i,j}^{k} - 1), \ \forall a_{(i,j)}^{k} \in A_{dilemma}, \ \text{ and } \ \forall k \in T$$

When $e_{(i,j)}^{k} = 0$, we should guarantee:

$$M \ge -\bar{t}_i + t_i - t_j + \underline{t}_j + \Delta_{(i,j)}^{k} \quad \forall a_{(i,j)}^{k} \in A_{dilemma}, \ \forall \underline{t}_i \le t_i \le \bar{t}_i, \underline{t}_j \le t_j \le \bar{t}_j$$

Let t_i take its maximum $\bar{t}_i$ and t_j take its minimum $\underline{t}_j$, we conclude:

$$\mathrm{M} \ge \Delta_{(i,j)}^{k}, \quad \forall a_{(i,j)}^{k} \in A_{dilemma}$$

In this way we can reduce the value of M,

$$M \overset{(4.37)}{\geq} \overline{t_i} - \underline{t_i} + \overline{t_j} - \underline{t_j}, \ \forall a^k_{(i,j)} \in A_{dilemma}$$

i.e. the smallest value of M becomes dependent on the sum of the time window sizes of the adjacent customers. It can not only be applied to the MTZ constraints for time windows, e.g. constraints (4.8) – (4.10), (4.25), but also applied to the MTZ constraints for capacities such as (4.28) and (4.31).

4.6 Tighter Big-M Formulation

We again focus on the big-M constraints, and take (4.25) as an example, and modify it as follows:

$$t_i + \delta_{i,j} + M \bullet (\sum_{k \in K} x^k_{i,j} - 1) \leq t_j \quad \forall (i,j) \in A_{dilemma} \tag{4.38}$$

Instead of writing the big-M constraint for each vehicle *k*, we sum the variable *x* up for all vehicles. Its feasibility holds due to the bundle constraint (4.18), i.e. each customer can be taken by one and only one vehicle. It is clear that (4.38) has a tighter LP structure than (4.25), since the polyhedron including (4.38) is a strict subset of the polyhedron including (4.25).

5 Primal Heuristics

In general, NP-hard combinatorial problems, such as Heterogeneous Vehicle Routing Problem with Time Windows, are solved in two approaches, exact algorithms and heuristic algorithms. The exact methods, e.g. Branch-and-Bound strategy for solving general integer linear programming problems, usually search the solution space in a systematic way, such that either a provable optimal solution is found, or we can prove theoretically at most how far our current best solution from the optimal solution is. Such exact methods usually have in worst case exponential time complexity with respect to the problem instance size, which in practice causes a strong rise in computation time when the problem size increases. On the other side, although without guarantee of finding optimal solutions, heuristic methods (or heuristics) usually acquire good or even near-optimal solutions within a relatively low computational cost. Other than that, heuristics can also serve as a preprocessing step for providing a good initial feasible solution for a Branch-and-Bound procedure, which in practice yields a noticeable speed up.

In this chapter, we will introduce two simple but robust construction strategies, Parameterized Greedy (PGreedy) and Randomized Greedy Search (RGS), for solving the Heterogeneous VRP with Time Windows as well as the four practical applications belonging to this class.

5.1 A Greedy Constructive Search Paradigm for HVRPTW

Before we introduce our metaheuristics PGreedy and RGS, which requires running the greedy construction algorithm iteratively, we first start with introducing a greedy algorithm framework for the general problem of Heterogeneous VRP with Time Windows (HVRPTW).

5.1.1 The Big Picture of the Greedy Construction Search

As stated in Hoos & Stützle (2004), basically all computational approaches for solving hard combinatorial problems can be characterized as search algorithms. The fundamental idea behind the search approach is to iteratively generate and evaluate *candidate solutions*. Generally, the evaluation of candidate solutions much depends on the given problem, and is often straightforward to implement so as to have it done in polynomial time. The fundamental differences between search algorithms are in the

way in which candidate solutions are generated. Normally they can be classified into two types, *constructive* or *local search* methods.

Constructive search methods (or *construction heuristics*) build a solution to a combinatorial optimization problem from scratch in an incremental way. Step by step and without backtracking, they add *solution components* until a complete solution is generated. Often, greedy construction heuristics are used which at each construction step add a solution component with maximum myopic benefit as estimated by a heuristic function.

In order to better clarify how a greedy constructive search can be performed on the HVRPTW, we first have a close look at some basic terms:

The *search space* is the set of all *candidate solutions* (or *solution* for short). As we have discussed in Section 3.2.2, in HVRPTW given is a directed graph $G(N, A)$ with node set N and arc set A. The nodes correspond to depots D and customers' sites S with $N = D \dot{\bigcup} S$. The arcs correspond to a road network. A depot represents a set of homogeneous vehicle. Let P denotes the set of (directed) paths. A path $p \in P$ in the graph starts from a depot visits (or covers) a set of customers $\{s_{p_1}, ..., s_{p_m} \mid s_{p_i} \in S\}$ and ends at the same original depot. A *solution* to the HVRPTW is a set of paths, denoted as P_s, that cover all customers and each customer is covered exactly once. A *feasible solution* to HVRPTW is a *solution* that satisfies all capacity and time window constraints (3.11) – (3.17). Each customer or each vehicle from a depot can be considered as a *solution component* to be added to the *solution*. An unfinished solution during the construction phase is called a *partial solution.*

The construction of a feasible solution is composed of two parts, in a macro-scale sense we generate paths one by one to cover the previously uncovered customers until all customers are served; subsidiarily in a micro-scale sense, we build a path by incrementally appending an unserved customer to the end of the current path in the hope of covering as many customers as possible, while in the meantime, minimizing the driving time or distance of unloaded trips (especially the deadhead trips).

Here in the sequel we will outline a greedy construction framework for solving HVRPTW, details are explained in depth thereafter:

greedy(G)

Input: the multicommodity network G

Output: a set of paths P_s

(1) Initialize Depot Set D and Customer Set S from N, $P_s := \varnothing$
(2) **while** (S is not empty)
(3) Select the best depot $d \in D$ to start the path p
(4) Select a best first customer $s_{first} \in S$ and add to p
(5) **while** (path is not full and S is not empty)
(6) Select a best next customer $s_{next} \in S$ and add to p
(7) **end while**
(8) $P_s := P_s \cup \{p\}$
(9) Update
(10) **end while**
(11) **return** P_s

Box 1: A greedy construction search paradigm for MDVSP with coupling trips and time windows.

In the following subsections we will take a closer look at each step of the greedy construction search algorithm shown in the box above.

5.1.2 Initialize the Search Graph

Line (1) of Box 1 initializes the graph that the search is based on. The depot set D and the customer set S is built up. These two sets contain all the solution components for our construction search, and during the search, empty depots and covered customers will be removed from the respective set, so that only the available components will appear in our search phase. The goal of our construction search is to cover all the customers, and the search process ends when S becomes empty. By some algorithms introduced in later sections that iteratively call the greedy construction search, the initialization should also restore some attributes of each components that is modified during the last search phase (for example, the starting time interval of each customer) to its original state, so that we can restart the search immediately without reading the input data again.

5.1.3 Select the Best Depot to Start

In line (3) of Box 1 we will select a best depot to start a path. Recall that a depot is a set of homogeneous vehicles, therefore the difference between the depots is actually the different characteristics between vehicle types and their home depot geographical location.

For simplicity reasons, we put the selection of the depot in front of the actual route construction, therefore the vehicle is not selected based on the existing route information, on the contrary the route to be constructed afterwards will depend on the vehicle that is chosen, for instance, a vehicle with small size or capacity will result in a shorter route or a restricted set of customers that are able to be taken. So here we are more like predicting a best vehicle for the route construction later on.

There are in general various factors to be taken into account while selecting a vehicle type, among which the most important ones are the fixed cost c for using a vehicle of this type, and the capability of the vehicle n_k, that is, how many customers can potentially be served by it. In practice we found that the cost factor can be given less attention at the construction phase, since in the local search phase we can exchange vehicle types based on the route information by replacing an expensive vehicle type by a cheaper but capable type. Subsidiarily, speed and loading capacity sometimes might also come into consideration.

However, which factors should be taken into account and how they should be weighed is quite problem-dependent. As a general outline for greedy algorithm here, we specify only two factors, cost and capability, to be scored for each vehicle types, with a certain function.

$$score_{vehicle}(i) = f(c, n_k), \quad where\ f : \mathbb{Q}^2 \to \mathbb{Q} \tag{5.1}$$

How to define this scoring function f will be discussed in later sections. We select the best vehicle type $t_{best} \in T$ according to the scoring function:

$$t_{best} = \arg\min_{t \in T} \{score_{vehicle}(t)\} \tag{5.2}$$

There is another important factor about selecting the depot, i.e. the depot location, when the set of depots are geographically dispersed. Normally the depot location can be selected randomly. But the selection process can be further optimized by combining the depot location selection together with the selection of the first customer node. Normally we will select the earliest possible customer in S to be the first customer node in the path, let $s_{earliest} \in S$ be it, and we use $l(s^+_{earliest})$ denote the

location of the starting station of the earliest customer node (note that the starting location and the destination location of a customer node can be different, e.g. each customer as a customer in the Multi-Depot Locomotive Scheduling project). Furthermore, let $d(l_1, l_2)$ denote the geographical distance between two locations $l_1, l_2 \in L$. Then we define the scoring function of each depot location with respect to a given vehicle type to be as follows:

$$score_{location}(l) = d(l, l(s^{+}_{earliest})),\ l \in L \tag{5.3}$$

That means, the depot with location nearest to the potential first customer should stand out. Putting it into formula, the best depot location $l_{best} \in L$ with respect to a vehicle type t_{best} is selected according to the given scoring function:

$$l_{best} = \arg\min_{l \in L} \{score_{vt}(l) : l \in L \wedge d^{l}_{t_{best}} \in D\} \tag{5.4}$$

Summing up, after the vehicle type t_{best} and the depot location l_{best} are determined, we have found our best depot $d^{l_{best}}_{t_{best}} \in D$ to start service. The method *selectDepot()* is shown in Box 2.

selectDepot(D)

Input: the depot set D, (optional) the location of the earliest possible customer $l(s^{+}_{earliest})$

Output: the best depot $d_{best} \in D$

(1) **if** (D is empty)
(2) print error message("Sorry, run out of vehicles!")
(3) exit
(4) **end if**
(5) **foreach** ($t \in T$)
(6) score t using (5.1)
(7) **end foreach**
(8) select the best vehicle type t_{best} using (5.2)
(9) **foreach** ($l \in L \wedge d^{l}_{t_{best}} \in D$)
(10) score l using (5.3)
(11) **end foreach**
(12) select the best depot location l_{best} using (5.4)
(13) $d_{best} := d^{l_{best}}_{t_{best}}$
(14) $|d_{best}|--$
(15) **if** ($|d_{best}| == 0$)
(16) $D := D \setminus \{d_{best}\}$
(17) **end if**
(18) **return** d_{best}

Box 2: greedy method to select the best depot.

5.1.4 Select the Best First Customer

Now we are back to the line (4) of Box 1. A starting depot $d \in D$ is selected, and now our vehicle starts from the depot and selects the first customer. The selection procedure will be different from the way we select the next customers thereafter, so we distinguish it out. As already mentioned above, the customer which can start the earliest should be chosen. We formulate the scoring function for selecting the first customer as follows:

$$score_{first}(i) = \underline{t_i},\ i \in S \tag{5.5}$$

And the best first customer stands out when it has the minimum lower bound of time interval:

$$s_{first} = \arg\min_{s \in S} \{score_{first}(s)\} \tag{5.6}$$

The *selectFirstCustomer()* is shown in Box 3. Note that we also keep an global variable $t_{current}$, denoting the current time of the path. After the first customer s_{first} is selected, usually the starting time of the customer is set to the earliest possible time point, and $t_{current}$ is set to the earliest time that the vehicle can leave the first customer.

After a customer is covered by a vehicle route, it is usually removed from the customer set S immediately. In some cases some certain customers require more than one vehicle of a certain type to serve at the same time, also known as a "Coupling Trip" feature (see Yuan 2007). Under such circumstances a selected customer will be removed from the available customer set S if and only if the customer is "fully covered". Otherwise we shrink its starting time window to a fixed time point at its earliest possible starting time, so that another vehicle can visit this customer at the same starting time.

There are in general some other update procedures required when a customer node is selected. The most noticeable step is to perform a constraint propagation procedure after the starting time interval of the selected customer has been changed, or fixed. Some other attributes of the selected customer might also change and require a corresponding propagation, if they will affect other customers. Besides, some global attributes might also need to change, such as the current load of the vehicle, etc.

selectFirstCustomer(S, k)

Input: the customer set S, the vehicle type of the path k

Output: the best first customer $s_{first} \in S$

(1) **foreach** ($s \in S$)
(2) score s using (5.5)
(3) **end foreach**
(4) select the best first customer s_{first} using (5.6)
(5) **if** (s_{first} is fully covered)
(6) $S := S \setminus \{s_{first}\}$
(7) **else**
(8) $\overline{t_{s_{first}}} := \underline{t_{s_{first}}}$
(9) **end if**
(10) $t_{s_{first}} := \underline{t_{s_{first}}}$
(11) $t_{current} := \underline{t_{s_{first}}} + \delta_{s_{first}}^{service}$
(12) update
(13) **return** s_{first}

Box 3: greedy method to select the best first customer.

5.1.5 Select the Best Next Customer

After the first customer is selected, how to select the customers that follow is essential for the solution quality. In the sequel we will look into the line (6) of Box 1, the method *selectNextCustomer()*, in more details.

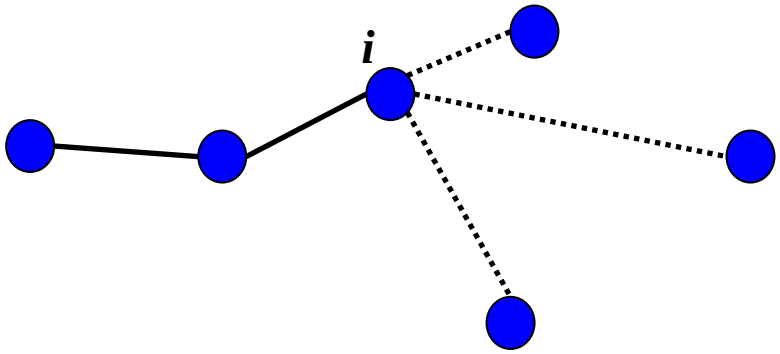


Figure 5.1: Greedy selection of the next customer node. Node *i* is the current node and the distance from node *i* to the next candidate nodes are shown in dashed lines.

We select the customers one by one and add it the current path, so as to
1) cover as many customers as possible, so that the fewest possible customers will be left for later constructions;
2) in the mean time, minimize the driving time of unloaded trips (especially the deadhead trips).

In general, these two goals are not in great conflict with each other, but they will bring in a subtle difference to the construction strategy. The second goal will indicate that a geographical nearest neighborhood criterion should be applied when selecting the next node to be added. But a simple thought will show that this criterion is undesirable. Let's imagine that the geographical nearest neighbor is timetabled at the end of the week, then our vehicle will become very "lazy". It drives to visit its geographic nearest neighbor, and stop there idle for a whole week only to wait for this nearest neighbor to start. The first goal however, will suggest that the temporal nearest neighbor should stand out, i.e. a customer with earliest possible starting time should be better, so that the sum of the deadhead driving time and the idle time of waiting for the next customer to start should be minimized. Nevertheless, the geographical distance between the destination station of the previous customer and the origin station of the next customer can be viewed as a reasonable factor to be considered, when one decide to parameterize the greedy algorithm.

Now we will put what we have discussed above into formula. Suppose we have arrived at the destination station of customer $i \in S$, with current time $t_{current}$, and we wish to select the best next customer $j \in S$. Let $k \in T$ denote the vehicle type of the current path.

First we need to determine whether a connection to a candidate customer is feasible. A customer $j \in S$ is feasible to be appended after a customer $i \in S$ in a path of vehicle type $k \in T$ under current time $t_{current}$, if and only if the following inequalities holds:

$$u_{(i,j)}^{k} > 0 \tag{5.7}$$

$$t_{current} + \delta_{(i,j),k}^{dh} \leq \overline{t_j} \tag{5.8}$$

The inequality (5.7) says the deadhead trip connecting customer i and j with vehicle type k is applicable. Recall from Chapter 4 that $\delta_{(i,j),k}^{dh}$ denotes the driving time of the deadhead trip from customer i to customer j with vehicle type *k*. The inequality (5.8) says the vehicle stands now in the end station of customer i at $t_{current}$, if the customer j is to be taken, the vehicle drives $\delta_{(i,j),k}^{dh}$ time to arrive at the starting station of customer j. This connection is feasible if and only if the arrival time is no later than the latest possible starting time of customer j, i.e. $\overline{t_j}$, the upper bound of the starting time interval of customer j.

After the feasibility of customer j is confirmed, how do we score it to measure its goodness? In this section, we will first define the scoring function in a simple way, following the temporal nearest neighborhood criterion. In the next sections we will look into some more complex, parameterize or randomly perturbed, scoring functions. Before we define the scoring function, we first evaluate the earliest possible starting time of customer j following customer i with path type k and current time $t_{current}$:

$$t_j \,|\, (i,k,t_{current}) = \max\{(t_{current} + \delta_{(i,j),k}^{dh}), \underline{t_i}\}, \;\; i \in S,\; k \in T,\; t_{current} \in \mathbb{R}^+ \tag{5.9}$$

The equation above shows, after the vehicle drives along the deadhead trip to the starting station of customer j, if the arrival time falls within the starting time interval of j, i.e. $\left[\underline{t_i},\ \overline{t_i}\right]$, then we can start customer j immediately; if the arrival time falls before the lower bound of starting time interval of j, i.e. $\underline{t_i}$, the vehicle has to wait

until the earliest time that the customer j can start, i.e. $\underline{t_i}$.

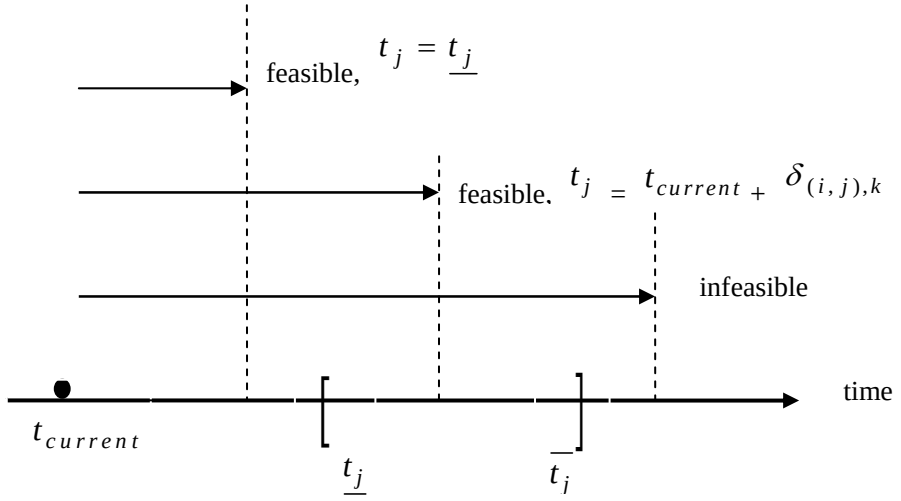


Figure 5.3: an overview of possible starting time of customer j

A scoring function to measure the temporal distance of customer j is given as follows:

$$score_{next}(j \mid (i,k,t_{current})) = t_j \mid (i,k,t_{current}) - t_{current},\ \ i \in S,\ k \in T,\ t_{current} \in \mathbb{R}^+ \qquad (5.10)$$

We are expecting the temporal nearest neighbor to stand out, which means it should have the minimum score:

$$s_{next} = \arg\min_{s \in S} \{score_{next}(s \mid (i,k,t_{current})) \mid s \in S \wedge s \text{ satisfies (5.7), (5.8)}\} \qquad (5.11)$$

The method *selectNextCustomer* is shown in Box 4.

selectNextCustomer(S, i, k, $t_{current}$)

Input: the customer set S, the previous customer $i \in S$, the vehicle type of the path k, current time $t_{current}$

Output: the best next customer $s_{next} \in S$

(1) **foreach** ($s \in S$)
(2) **if** (s satisfies (5.7), (5.8))
(3) score s using (5.10)
(4) **end if**
(5) **end foreach**
(6) select the best next customer s_{next} using (5.11)
(7) **if** (s_{next} is fully covered)
(8) $S := S \setminus \{s_{next}\}$
(9) **else**
(10) $\overline{t_{s_{next}}} := \underline{t_{s_{next}}}$
(11) **end if**
(12) determine $t_{s_{next}}$ using (5.9)
(13) $t_{current} := t_{s_{next}} + \delta^{dlv}_{s_{next}}$
(14) **return** s_{next}

Box 4: greedy method to select the best next customer.

5.1.6 Computational Complexity of the Greedy Construction search

In the following we will make a brief evaluation about the computational complexity of our greedy constructive search introduced in Section 5.2 above. We first need some parameters to describe the instance size.

- $n = |S|$ denotes the number of customers;
- $k = |D|$ denotes the number of depots (number of distinguishable vehicle types);
- m denotes the total number of vehicles, and m can be computed as follows:

$$m = \sum_{i=1}^{k} | d_i |, \quad d_i \in D \tag{5.12}$$

where $| d_i |$ denotes the number vehicles contained in depot d_i.

The outer loop in line (2) of Box 1 is called every time when a path is built. We can bound the number of paths $| P_s |$ as follows:

$$| P_s | \leq \min\{n, m\} \tag{5.13}$$

That is because the number of paths can exceed neither the number of vehicles nor the number of customers. For every path we will select a best depot at the beginning following Box 2, scoring each depot takes constant time, and we will at most score for each depot once, so the depot selections take at most $O(\min\{n, m\} \bullet k)$ time.

Each customer can be selected at most twice by two different paths, and every time we are selecting a customer, no matter as the first customer or next customer, each available customer will be scored in constant time (see Box 3 and Box 4), and at most n customers are to be scored. So the customer selections take at most $O(2n \cdot n) = O(n^2)$ time.

All in all, it requires $O(\min\{n, m\} \bullet k + n^2)$ time for a greedy construction.

The memory space required by the greedy algorithm is exactly the memory space needed for storing the multicommodity flow network, which is mentioned in Chapter 3 to be $O(kn + n^2)$.

5.2 How to improve the greedy construction search

As can be seen from above, a greedy construction search algorithm builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. Some classical greedy algorithms, despite their natural myopic behavior, are able to find optimal solution for some combinatorial optimization problems, like Dijkstra's algorithm for finding a shortest path between two nodes in a graph with non-negative arc weights, and Prim's algorithm for solving minimum spanning tree problem, etc. However, for hard combinatorial problems, i.e. the provable NP-hard problems, like TSP or in our case Heterogeneous VRP with Time Windows, no greedy-type algorithms are known to guarantee the optimal solution.

Nevertheless, greedy constructive algorithms are still an attractive and effective problem solving strategy. Especially for some combinatorial problem types on which local search is not very effective, such as HVRPTW, then the solution returned from the construction phase will be essential for the final solution quality. Therefore, a careful tuning of the greedy constructive algorithm is essential for solving this type of problems heuristically. In the sequel we will introduce a few strategies to improve the greedy construction algorithm.

In this section we will first give the general guideline for tuning a constructive greedy search algorithm in 5.2.1, taking care of the trade-off of intensity and diversity; in the sequel three general diversification strategies are introduced, with utilization of a Restricted Candidate List in 5.2.2, parameterization of the scoring function in 5.2.3, and a highly biased perturbation multiplier in 5.2.4.

5.2.1 Intensity versus Diversity

A common strategy to improve a constructive search algorithm is to follow with a local search phase. But the prerequisite to perform a local search is to define an effective neighborhood relation of a solution space, which is not always easy. We will discuss it in more details in section 5.7.

The algorithms we describe in the following sections 5.4 – 5.6, are based on iteratively running the greedy construction search algorithm without local search. As is shown in Chapter 7, iterative greedy construction search algorithms yield very good solutions for HVRPTW and its practical applications, when they are carefully tuned. One general principle for tuning such kind of construction heuristics is to balance between intensity and diversity of the searchable space of the given problem instance.

- **Searchable Space:** is a subset of the whole solution space that is reachable by the iterative greedy construction search, and every element of it has a hitting probability. Note that the searchable space is usually not complete, the setting of different parameter combinations will affect its structure.
- **Intensity:** the searchable space becomes intensive, when it is in a concentrative manner. The space focuses on relatively less elements, and some good elements are biased with higher hitting probability, by strongly guiding the construction phase via the greedy evaluation function.
- **Diversity:** the term diversity means, to explore the solution space in a reasonably extensive manner, to cover more elements and to prevent the search process from stagnation, i.e. to make sure our search area won’t be restricted to some region without high quality solutions in it.

If the searchable space becomes intensive, it probably will reach a good solution very fast, since in general the average solution quality will be higher. But it lacks variety in

the solution construction, then the solutions it can reach are very limited, hence that might probably exclude many solutions of higher quality, and result being trapped in a so-called stagnation. On the contrary, diversification will broaden the searchable space by bringing in variety and reasonable perturbation. In many cases the diversification mechanism also causes the average solution quality to drop, but the broadened searchable space will increase the probability of hitting a solution of very high quality. However, we must be very careful when we adjust the degree of diversity. If the searchable space becomes too diverse, only very bad solution will be returned, and the probability of hitting a good solution becomes only theoretical, because not enough guidance is given during construction search.

So the principle for tuning the greedy algorithm is to adjust it taking into account the balance of intensity and extensity, i.e. in ideal case, to extend the searchable space as wide as possible, and distribute the hitting probability of solutions according to their qualities, which means, the very good solutions gets as high probability as possible to be hit.

An extreme example of intensity is to perform the greedy construction search in a "static" way, such that in each partial solution construction step there is always a unique best solution component to be selected and added. In such case, every greedy construction will return exactly the same solution. One possible diversification is, at each partial solution construction step, we store the set of the solution components that achieve the equally best evaluated value, and select one element from the set uniformly, i.e. solution components that share the same best score have an equal chance to stand out. In this case we for example can rewrite equation (5.11) as follows:

$$S_{\min} = \arg\min_{s \in S} \{score_{next}(s \mid (i,k,t_{current})) \mid s \in S \wedge s \text{ satisfies (5.7), (5.8)}\} \quad (5.14)$$

And

$$s_{next} \in S_{\min} \text{ uniformly randomly drawn} \quad (5.15)$$

In general, admitting random selection of only best solution components with equally good scoring value still generates only limited number of different candidate solutions. In the following we will introduce some more diversification mechanisms, including Semi-greedy which makes use of a Restricted Candidate List, RGS which makes use of a much better guided diversification comparing with Semi-greedy by a perturbation multiplier, PGreedy which diversifies the greedy construction by parameterize the scoring function.

5.2.2 Semi-greedy Algorithm -- The Utilization of Restricted Candidate List

In order to increase diversity of the searchable space, Hart and Schogan (1987)

proposed *semi-greedy* heuristics, which later become the construction part of the well-known metaheuristic GRASP (Greedy Randomized Adaptive Search Procedures, see Feo and Resend (1989); (1995); Hoos and Stützle (2004)).

In contrast to standard greedy construction search methods, *semi-greedy* does not necessarily select the best scored solution component. Instead, it generates in each construction step a *restricted candidate list* (RCL), and then selects one element out of the RCL randomly according to a uniform distribution. In *semi-greedy* there are two different ways to define RCL:

i. **cardinality restriction**: only the *k* best ranked solution components are included in the RCL;
ii. **value restriction**: let *S* be the set of feasible solution component, and $s \in S$. Let $g(s)$ denote the greedy scoring value of *s*. We further define $g_{\min} = \min\{g(s) \mid s \in S\}$ and $g_{\max} = \max\{g(s) \mid s \in S\}$. Then the solution component *s* is admitted into the RCL, if $g(s) \le g_{\min} + \alpha(g_{\max} - g_{\min})$.

k and α used above are parameters of the algorithm. Clearly, the smaller *k* or α, the more intensive the selection will be.

Semi-greedy brings in considerable diversity to the standard greedy construction search, but as we have seen from our practical experience, this strategy does not yield enough satisfactory solutions, especially when the instance sizes are getting large. The reason is because it does not give bias to the elements in the RCL according to their qualities, which makes the searchable space too extensive, prevent the best solutions from standing out. In section 5.5, we will discuss another diversification strategy Randomized Greedy Search (RGS), which restricts the number of candidate solution components as well as biases the selection according to the quality of the solution component measured by a given scoring function. GRASP improves *semi-greedy* with a subsidiary local search, and all local search issues will be discussed in section 5.7.

5.2.3 Random selection at local step by a perturbation multiplier

The bad performance of Semi-greedy has urged us to develop another diversification strategy, which is also based on bringing in perturbation during the solution component selection. In contrast to keeping a restricted list and thereafter selecting uniformly from it, we generate a perturbation factor from some distribution for each candidate solution component and multiply it to yield a perturbed score. In such a way a solution component with a higher solution quality by its original scoring function will have a higher probability to stand out. And we can control the degree of diversity

easily by adjusting the distribution form.

More details will be discussed in the following section 5.4, and we have named this type of diversification strategy as “Randomized Greedy Search (RGS)”.

5.2.4 Parameterize the greedy scoring function

The two previous introduced diversification strategies Semi-greedy and RGS both focused on giving perturbation during the solution component selection, now we turn our attention to another aspect, the greedy scoring function. In PGreedy algorithm, we introduce more than one greedy criterion into the scoring function, parameterize each criterion with an individual weight and try to find the best parameter setting for the scoring function. This strategy can also be viewed as a diversification strategy based on the scoring function, while the more greedy criteria we introduce and the larger the parameter space we assign, the more diverse the searchable space will be.

We can hybridize the PGreedy with RGS by including random selection at each local step to further extend the searchable space, or in another way, first apply PGreedy to find the best parameter settings for the scoring function and then perform RGS based on the parameterized scoring function. The hybrid of the two heuristic strategy PGreedy and RGS, named as PRGS, will be introduced in section 5.5.

In the following we will first give detailed introduction to PGreedy in section 5.3, RGS will be treated in section 5.4, and leave the hybrid PRGS in section 5.5.

5.3 Parameterize the greedy algorithm (PGreedy)

In this section we give in-depth description on our first metaheuristic approach -- Parameterized Greedy Algorithm (PGreedy) for solving Heterogeneous VRP with Time Windows. The basic guideline for applying PGreedy is as follows: introduce more than one criterion in to the greedy scoring function, parameterize these criteria with individual weight, and find the best parameter setting by an automated parameter tuning technique.

It is reported that PGreedy can be further hybridized with existing local search heuristics, or can be included in a GRASP framework (see Fuegenschuh and Hoefler 2006), or to generate a starting population for a genetic algorithm.

In the sequel, a basic guideline regarding how to parameterize a greedy algorithm for general purpose is given in 5.3.1, followed by the topic how to specifically parameterize a greedy algorithm for Heterogeneous VRP with Time Windows in 5.3.2, and various automated parameter tuning techniques are described and discussed in

5.3.3.

5.3.1 How to parameterize a greedy algorithm

Greedy algorithms are in general simple heuristics to construct feasible solutions for combinatorial optimization problems from scratch in short time. In some cases, they even find proven optimal solutions (for example, Prim's algorithm for spanning trees). For hard optimization problem however they only give sub-optimal solutions. Research interest in the past two decades has thus turned to more sophisticated meta-heuristics (such as genetic algorithms, tabu serach, ant colony optimization, GRASP, or simulated annealing). Our approach differs from that. We improve the classic greedy algorithm by introducing more greedy criteria, parameterizing each criterion with individual weight, and tuning automatically to find the best parameter setting.

It is well known that the scoring function (or heuristic evaluation function) plays an essential role in the success of a greedy algorithm. But how to define a good scoring function is quite problem-specific, and sometimes even instance-specific. Most of the classic greedy algorithms use a rather straightforward heuristic evaluation function, since the immediate benefit of a solution component is rather obvious, e.g. nearest neighborhood heuristic for TSP, which always selects the next node with the minimum distance from the previous selected node; or the heuristic evaluation function has been proved to be optimal, e.g. Dijkstra's algorithm and Prim's algorithm that use a label to measure the goodness of each remaining nodes (see Nemhauser and Wolsey (1999)). But we notice that in many highly constrained combinatorial optimization problems, such as HVRPTW, it is in general difficult to identify an immediate best local step during the construction process. Thus a simple scoring function statically depending on single criterion is unreliable and even misleading. To overcome this structural problem of greedy algorithms, and in order to make the greedy approach more robust, we introduce the parameterized greedy algorithm (or PGreedy for short). To the best of our knowledge, this idea of parameterizing a greedy algorithm for solving hard combinatorial problem was first proposed by Fuegenschuh 2005 as a new type of metaheuristic, while tackling the Vehicle Routing Problem with Coupled Time Windows.

A greedy algorithm makes use of a greedy scoring function, and always selecting the candidate with the best score. Assume that the set S denotes the set of candidate solution components, and the scoring function is denoted as $s_\lambda(i)$ with a vector of parameters λ. The best candidate in each local step will be selected as follow:

$$i_{best} = \arg\min_{i \in S}\{s_\lambda(i), i \in S\} \tag{5.16}$$

The first step of PGreedy is to find more than one criterion into the scoring function.

In general finding those criteria is more an art than science, and it is problem-dependent, and even instance-independent. Let's assume that a vector of criteria are found as $c := < c_1, c_2, ..., c_p >$.

The second step of PGreedy is to parameterize all criteria with individual weight into a linear scoring function, suppose $\lambda := < \lambda_1, \lambda_2, ..., \lambda_p >$ to be a vector of parameters corresponding to the vector c. The scoring function s_λ with respect to λ can be written into a linear form as follows:

$$s_\lambda(i) := \sum_{j=1}^{p} \lambda_j \cdot c_j(i) \tag{5.17}$$

Each parameter vector λ defines a unique scoring function s_λ, and let $x(\lambda)$ denote the solution returned by the greedy construction search according to the scoring function s_λ, and $z(\lambda)$ denote the objective value of the solution $x(\lambda)$. Then our goal of PGreedy is to find the best parameter setting that yields the minimum objective cost for the specific problem, as shown below:

$$z^{\text{pgreedy}} = \min\{z(\lambda) : \lambda \in \mathbb{Q}^k\} \tag{5.18}$$

Do we need exactly p parameters for p greedy criteria? The good news is, if the scoring function is parameterized in a linear form, as shown in (5.17), we can decrease the dimension of the parameter vector by 1, as the following lemma tells us.

Theorem 1: *let* $\lambda, \lambda' \in \mathbb{Q}^p$. *If there is a positive scalar* $t \in \mathbb{Q}_+$ *such that* $\lambda' = t \cdot \lambda$, *then* $x(\lambda) = x(\lambda')$ *and hence* $z(\lambda) = z(\lambda')$.

Proof Outline: notice that at each local step, for any candidate solution component element $i \in S$, $s_{\lambda'}(i) = s_{t\cdot\lambda}(i) = t \cdot s_\lambda(i)$ holds from (5.17). Equation (5.16) implies $i_{best} = \arg\min_{i \in S}\{s_\lambda(i), i \in S\} = \arg\min_{i \in S}\{t \cdot s_\lambda(i), i \in S\} = \arg\min_{i \in S}\{s_{\lambda'}(i), i \in S\}$, which shows each generated local best solution components with respect to scoring function s_λ and $s_{\lambda'}$ are identical if the parameter vectors λ and λ' are linearly dependent.

Hence $x(\lambda) = x(\lambda')$ and $z(\lambda) = z(\lambda')$ follows. □

The theorem above tells us, many elements in the parameter space $\lambda \in \mathbb{Q}^p$ are redundant, and we can actually reduce the dimension of the parameter space by one. How to make use of this one-dimension reduction?

A straightforward way is to fix one of the parameter coordinate to a non-zero value, without loss of generality, if we fix the first parameter coordinate $\lambda_1' = 1$, such that $\lambda' = <1, \frac{\lambda_2}{\lambda_1}, ..., \frac{\lambda_p}{\lambda_1}>$. Note that in such a way λ' is unbounded. If we are going to search the parameter space and find the best parameter setting, the parameter space has to be bounded. To this end we can bound each parameter coordinate manually by setting a lower bound $\underline{\lambda_i'}$ and an upper bound $\overline{\lambda_i'}$, according to common sense and the magnitude of quantitative terms of each greedy criteria c_i. In such a way, the parameter space will be reduced to a $(p-1)$-dimensional cuboid.

This straightforward dimension reduction technique is especially helpful and necessary for some manual parameter tuning technique, such as grid search, since firstly it substantially reduce the redundant parameter settings that will come up in the grid search process, and secondly bounding the parameter space into a cuboid makes the grid distribution much easier.

Besides the straightforward $(p-1)$-dimensional cuboid reduction described above, Fuegenschuh 2005 has introduced some other automated dimension reduction technique, using norm restriction to project the $\mathbb{Q}^p$ space onto a unit hyper-plane, based on the following corollary:

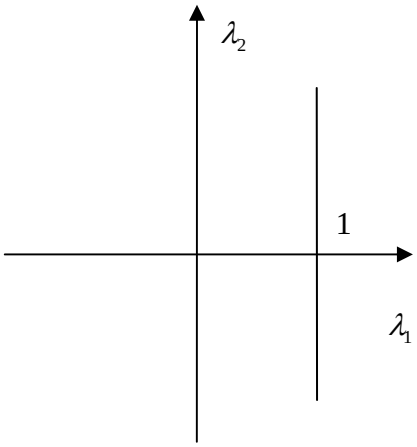
Corollary 2: *Let* $\|\cdot\|$ *be an arbitrary norm. For every solution* $x(\lambda)$ *with* $\lambda \in \mathbb{Q}^p \setminus \{0\}$ *there is a* $\lambda' \in \mathbb{Q}^p$ *with* $\|\lambda'\| = 1$ *such that* $x(\lambda) = x(\lambda')$.

Proof Outline: is an easy deduction from Theorem 1 by setting $\lambda' = \frac{1}{\|\lambda\|} \cdot \lambda$. □

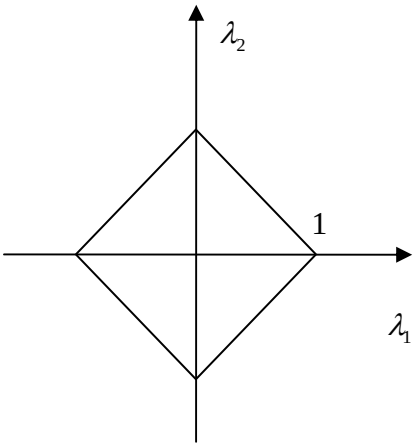
In view of Corollary 2, we do not have to search entire $\mathbb{Q}^p$, a bounded proper subset

will be sufficient, which results in a considerable speed-up in computation. This type of resulting hyperplanes is called norm-based. Note that the automated parameter tuning phase will require each parameter space to be bounded, and this holds for the norm-based hyperplanes automatically, and there is no need to manually define a bound for each parameter. However, the norm-based hyperplanes will be hard for performing grid search on, since they do not have a “rectangular” structure for the grid distribution.

With different norm, our parameter space will be projected onto different hyperplanes, as shown in the figure below:

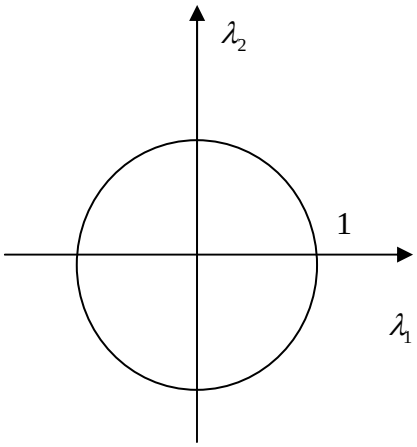


fix $\lambda_1 = 1$ and project the whole parameter space onto a straight line



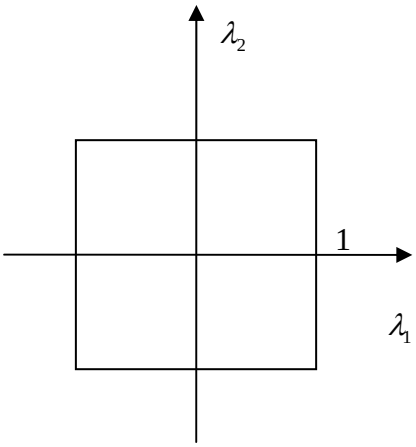
Apply the 1-norm

$$|| \lambda ||_1 = \sum_{i=1}^{p} | \lambda_i | = 1$$



Apply the Euclidean norm

$$|| \lambda ||_2 = \sum_{i=1}^{p} | \lambda_i^{\,2} | = 1$$



Apply the maximum norm

$$|| \lambda ||_\infty = \max_{i=1,\ldots,p} | \lambda_i | = 1$$

Figure 5.2: Reduce the two-dimensional parameter space to its hyperplane.

5.3.2 How to parameterize a greedy algorithm for HVRPTW

The adaption of PGreedy for finding good feasible solutions of Heterogeneous VRP with Time Windows (HVRPTW) can be done as follows. As shown in Box 1 there are essentially three different steps during the greedy construction. We make use of three individual scoring functions, one for each step as follows:

The first scoring function evaluates each possible assignment of a “fresh” vehicle from the depot. More detailed, for each vehicle type, this parameter-dependent scoring function takes into account the fixed cost for using a vehicle of this type. As a second criteria, it takes into account the capability of the vehicle, that is, how many customers can potentially be served by it. In some cases the velocity of the vehicle will be also considered.

The second scoring function uses three parameters for evaluating which customer to serve first with the previously selected “fresh” vehicle. To this end, it takes into account the driving cost from the depot to each customer, the earliest possible starting time for this customer, and the number of possible further customers after this one has been served.

The third scoring function is the most critical one for the quality of the produced solution. It also makes use of three parameters. The first parameter takes the driving cost between two customers into account. The second parameter evaluates the idle time the vehicle has to accept when possibly arriving early at the next customer's site. The thrid parameter evaluates the compatibility of the time windows of the two customers. We do not take the number of further customers into account here (as we did for the second scoring function), because the generation of this number take too much time and turned out to have too little effect on the outcome.

These three scoring functions are called iteratively. By the first, we find the most promising vehicle. By the second, we find the most promising first customer for it. Then we apply the third scoring function to generate its route until no further customer can be added, and the vehicle has to be sent back to the depot. Then we start again with the first scoring function, and so on, until all customers are served. At that point in the algorithm, we have a feasible solution to the problem, which awaits a local search phase for further improvement.

However, not all criteria in all scoring function described above will be applied in each problem. Which criteria to be used usually depends on the problem context, and sometimes even depends on the instance size and characteristics. It is also natural that each problem may come up with different particular greedy criteria. Finding and

determining greedy criteria for the scoring function is in general more an art than a science. There are two types of criteria to be categorized:

- Reasonable criteria: criteria such as the starting cost and capability of a vehicle are important factors to be considered, when selecting a vehicle to start; in selecting next node, criteria such as the temporal and geographic distance of the next node from the previous node are important factors to be considered, as well as the time window size of the next node, which indicate the flexibility of the next customer.
- Perturbation criteria: during the local selection phase, sometimes some "non-sense" criteria will come into the scene, which is called perturbation critera. For example, in selecting next node, some more "non-sense" criteria can be added into the scoring function, such as the service duration of the next node and even the graph index of the next node. How to apply these perturbation criteria are according to the trade-off of intensity and diversity of the given problem instance. The more criteria introduced and the larger the parameter space, the more diverse the searchable space will be. If the searchable space becomes too intensive and restricted for instance, we can introduce some perturbation criteria to bring more distraction to broaden the searchable space.

Note that the basic PGreedy framework is extended here for HVRPTW in the following 2 aspects:

1. instead of adaption of only one scoring function all the time, we apply *more than one* scoring functions;
2. Parameterize these criteria in not only linear scoring function, but also *non-linear parameterizations* are allowed, as we will see in latter sections.

Since there are more than one scoring functions and parameterization is possibly not linear, we cannot bound the parameter space under a norm-based hyperplane as described above. However, we can still bound each parameter manually, within a reasonable range, such that the parameter space is bounded under a cuboid.

Now that we have parameterized a bunch of greedy criteria, and left a various-shaped parameter space to explore.

How do we start our search in the given parameter space landscape? An automated parameter tuning technique is integrated in the PGreedy framework, and is performed in the sequel to find the best parameter weight setting.

5.3.3 Automated parameter tuning for PGreedy

An automated parameter tuning is an integral part of the PGreedy algorithm for exploring a given parameter space. Each element of the parameter space, also known as a parameter setting, corresponds to a specific solution. We measure each parameter

setting with the quality of its generated solution. During our experiments, three different parameter tuning techniques are applied, and will be introduced here, namely, grid search, uninformed random walk and improving hit and run.

5.3.3.1 Parameter tuning with Grid Search

In order to find the best parameter combination, our first experiment for the parameter tuning is to apply a grid search. We divide the parameter cuboid into regular grid, and to try out each parameter combination at each vertex of the grid. The basic grid search trajectory is shown below:

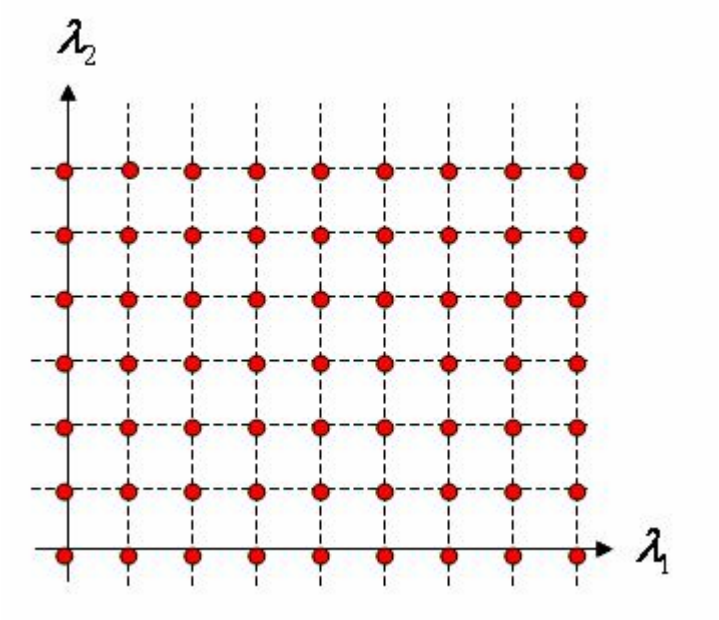


Figure 5.3: Grid Search for the two-dimensional parameter cuboid.

One evident drawback of the grid search for parameter tuning is that, it is not totally "automated". It is on one side very sensitive to the structure of the parameter space, and on the other side some careful manual tuning is still required and sometimes even essential if we want to achieve its best performance. Reasons are listed as follows:

1. it is very dependent on the structure of the parameter space. If the parameter space is not a rectangular cuboid, but a sphere for instance, a reasonable grid distribution in practical will be hard.
2. the parameter range must be well-designed and tailor-made, since the grid are divided regularly, a bad determination of parameter range may cause a large amount of search in vain. For example if a parameter has been assigned 10 individual values in the grid, but only 1 value can generate good solutions, then 90% of the whole grid search becomes a waste of time. Unfortunately in practice, the "good" range of a certain parameter is always a posteriori. During our practical experiment, we always keep track of the parameters that achieve good solutions over a certain quality, e.g. the top 5% elite solutions, and make manual observation, if certain range of a parameter never shows up, we will cut off this

parameter range from the parameter space manually.

3. hard to handle parameter inter-relations. Although the procedure introduced above in 2. will greatly eliminate the unnecessary parameter range and refine the given parameter space, when the parameters only make effect when they satisfy some inter-relations, for instance when parameters λ_1 and λ_2 satisfies the linear relation $\lambda_1 = c \cdot \lambda_2$, good solution will be generated, in such patterns the quality of parameter may be hard to judge. Besides, due to the regularity of the grid, if some linear parameter combinations are doomed to generate bad solutions, it will still be repeated multiple times.
4. hard to control the stopping criterion. A greedy construction will be built for each parameter combination. We usually wish the program to keep running until a certain time restriction. However, with grid search the number of vertices of the grid is predefined, and one has to estimate the time needed for each run and how many parameter combinations are appropriate. This requires again a certain degree of expertise to perform the work.

However, as our experiment shows, a fine-tuned grid search for PGreedy is able to achieve very good solutions. But the manual tuning by modifying each parameter's upper bound, lower bound and step size, and the manual observation of each parameter is relatively hard to implement. Therefore we seek further some better automated parameter tuning techniques.

5.3.3.2 Parameter tuning with Uninformed Random Picking

Our next approach is to break the determinism of grid distribution by bringing in randomness in the parameter selections. With uninformed random picking, our selection is uniformly random distributed, each element of the parameter space has the equal probability to be selected in each step. One possible two-dimensional uniform distribution is shown in the figure below:

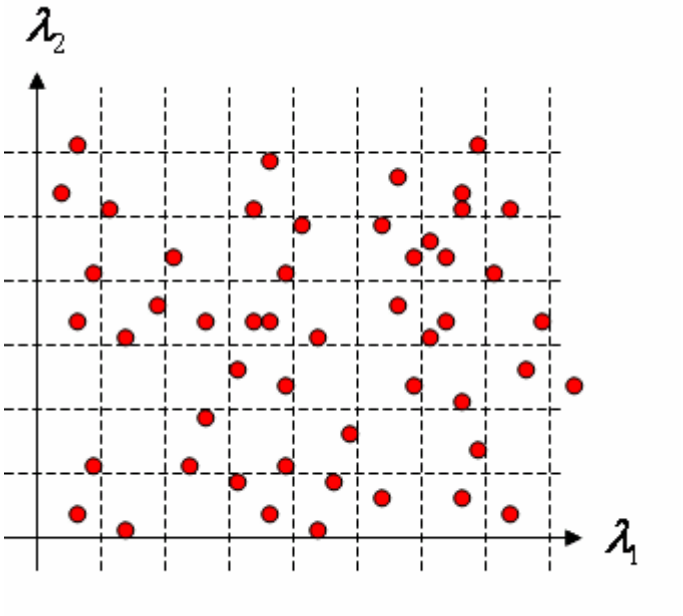


Figure 5.4[1]: Uninformed Random Picking for the parameter cuboid.

Apparently, parameter tuning using uninformed random picking for PGreedy also requires a refined parameter space, but the structure of space does not matter anymore, and the results are relatively less sensitive to the quality of parameter space comparing with grid search. What's more important, it is now much better to be "automated" since we do not need to worry about giving a bound and a step size for each parameter anymore, but simply keep the program running until its deadline comes.

5.3.3.3 Parameter tuning with Improving Hit and Run

Just scattering our search uniformly random on the search landscape automated, but as you might imagine it is not effective enough, since they don't provide any mechanism for steering the search towards solutions. In this section we will introduce another randomized parameter tuning technique, Improving Hit Run (IHR) first proposed for Global Optimization

Improving Hit and Run is first proposed by Zabinsky et al. 1993 for solving Global Optimization problems. Fuegenschuh 2005 first adopted this algorithm from Global optimization to tuning parameters for PGreedy, when tackling Vehicle Routing Problems with Couple Time Windows.

Improving Hit and Run differs from the uninformed random picking in the search trajectory in the search landscape, in a way that it stores the current best point, and during each new search step, a new point is generated randomly but biased. The point

[1] The figure is extracted from Fuegenschuh 2005hm.

which is “structurally” nearer to the current best point has a larger hitting probability. The new point is accepted and updated if and only if it is improving. An outline of IHR is given below:

Ihr(L)

Input: the bounded search landscape $L \in \mathbb{Q}^p$ **Output**: the current best minimum point $p_{\min} \in L$
(1) Initialize $p_0 \in L$, $k := 0$; (2) Generate a direction vector D_k using a *direction generator*; (3) Generate a new point along D_k from the current point by sampling uniformly with a *step size generator*; (4) Accept the new point if it is improving; (5) If the stopping criterion is met, stop and return current point p_{k+1}. Else increment k and continue from (1).

Box 5: Improving Hit and Run for search landscape.

In step (1), it is better to locate the initial point of the IHR at the “center” of the search landscape, since it is easier to get trapped at “corners”.

In step (2) we generate a direction around the current best point p_k by a *direction generator*. Available direction generators are:

1. ***HD - Hyperspherical Direction Generator*** generates a random direction uniformly distributed on a hypersphere around the current point.
2. ***CD - Coordinate Direction Generator*** generates a random direction uniformly distributed among n coordinate directions.
3. ***cCD -Cyclic Coordinate Direction Generator*** generates a direction from n coordinate directions in a cyclic manner.

A very interesting observation has been noticed by Zabinsky that, it is proven that IHR with HD generator converges to a global optimum with probability 1, while there are known examples for which IHR with CD or cCD will not converge. However, in some cases IHR with CD performs better than IHR with HD. Therefore it is encouraged to try all options and select the most appropriate one for the problem of interest.

In step (3) a *step size generator* is applied to determine how far the generated new point can be from the current best point. After the step size is determined, a new point

will be generated uniformly along the direction D_k from p_k with no further than the step size. Available step size generators are:

1. ***Full Range Step Size Generator*** generates a random point uniformly on the entire feasible portion of the specified line segment.
2. ***Adaptive Step Size Generator*** generates a random point uniformly on a reduced line segment. The length of the reduced line is modified in each iteration depending on the acceptance ratio. Detailed information about adaptive step size can be found in Neogi 1997.

In step (4) and (5) we accept a new point if it is improving, and continue the procedure until the stopping criterion is met.

During our IHR applications for parameter tuning, we have adopted the hyperspherical direction generator and further a full range step size generator. It can be illustrated in the figure below:

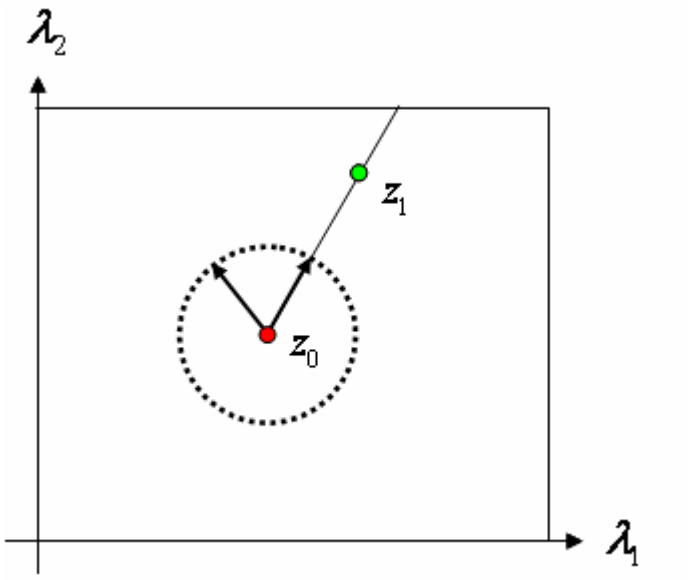


Figure 5.5: Improving Hit and Run with hyperspherical direction generator and full range step size generator. We firstly generate a random direction uniformly on a hypersphere around the current point, and then generate a random point uniformly on the entire feasible line segment along the direction from the current point. The new point is accepted if it is improving.

The current experiment shows that every new generated point is quite far away from the current best point, but the local area of the current point is usually not sufficiently exploited. In the future, it should be a good starting point to further experiment IHR with an adaptive step size generator, so that the local area around the current best point can be better examined at first.

5.4 Randomized Greedy Search (RGS)

Another strategy to bring diversity into the standard greedy construction search algorithm while retaining a certain degree of intensity, is to multiply the greedy scoring function with a perturbation factor, which is drawn from some probability distribution. We suggest the term *Randomized greedy Search* (RGS) for such randomization procedures.

Similar to the semi-greedy construction heuristic, it is not necessary that only the best-scored solution components are able to be selected. However, the random selection of the solution components should be strongly biased by its goodness, measure by a given local scoring function. The better the score, the higher the chance that the solution component should be picked up.

Let S be the set of feasible solution component, and $s \in S$. Let $g(s)$ denote the greedy scoring value of s. The we set $g_{\max} = \max\{g(s) \mid s \in S\}$, and $g_{\min} = \min\{g(s) \mid s \in S\}$. We further define $h(s) = g(s) - g_{\max} \le 0$ to be a "normalized" greedy heuristic scoring value of s. We multiply the normalized greedy score $h(s)$ with a perturbation factor p, and acquire the perturbed normalized greedy score $h_p(s) = h(s) \bullet p$, where p is positive real number randomly generated from some probability distribution. We select the solution component s^* with the minimal perturbed score, i.e. $s^* := \arg\min_{s \in S}\{h_p(s)\}$.

We have applied two different probability distributions for p in our experiments:

- Uniform distribution within the value interval [l, u], e.g. [800, 1000]. If we set $\alpha := \frac{u-l}{u}$, it becomes similar with semi-greedy heuristic using RCL with value restriction and quality parameter α, solution component s might be possible to enter the candidate list, if $g(s) \le g_{\min} + \alpha(g_{\max} - g_{\min})$. But unlike semi-greedy, every element inside the candidate list has a biased probability to be selected. It is easy to see that, the better the heuristic score, the higher the probability that the solution will be chosen.

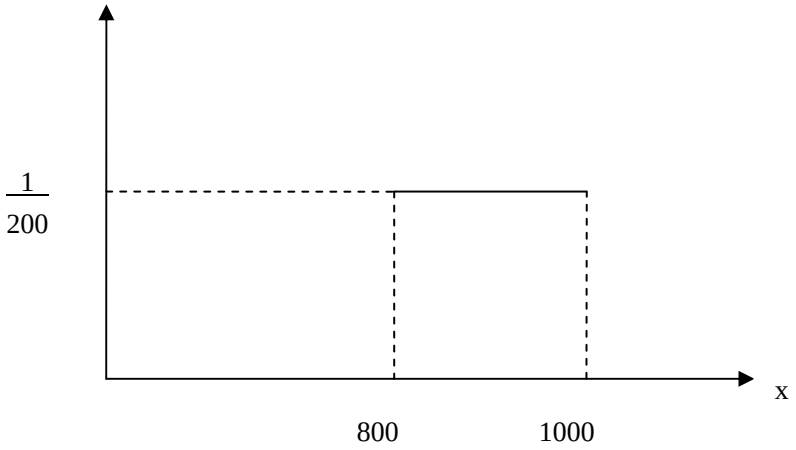


Figure 5.6: probability density function of uniform distribution in [800, 1000].

- Normal distribution with $N(\mu,\sigma)$, e.g. $N(1000, 50)$. With p drawn from this distribution, theoretically every feasible solution component has the possibility to be chosen, since normal distribution spreads from $-\infty$ to $+\infty$. For the distributed values are relatively centralized, solution component with better score will have bigger advantage than in the uniform distribution. That is to say, comparing with the uniform distribution, better solution components are even more likely to be chosen, while all other solution components still retain its possibility to be selected. In practice the perturbation parameter p drawn from normal distribution usually performs better than the one generated from uniform distribution.

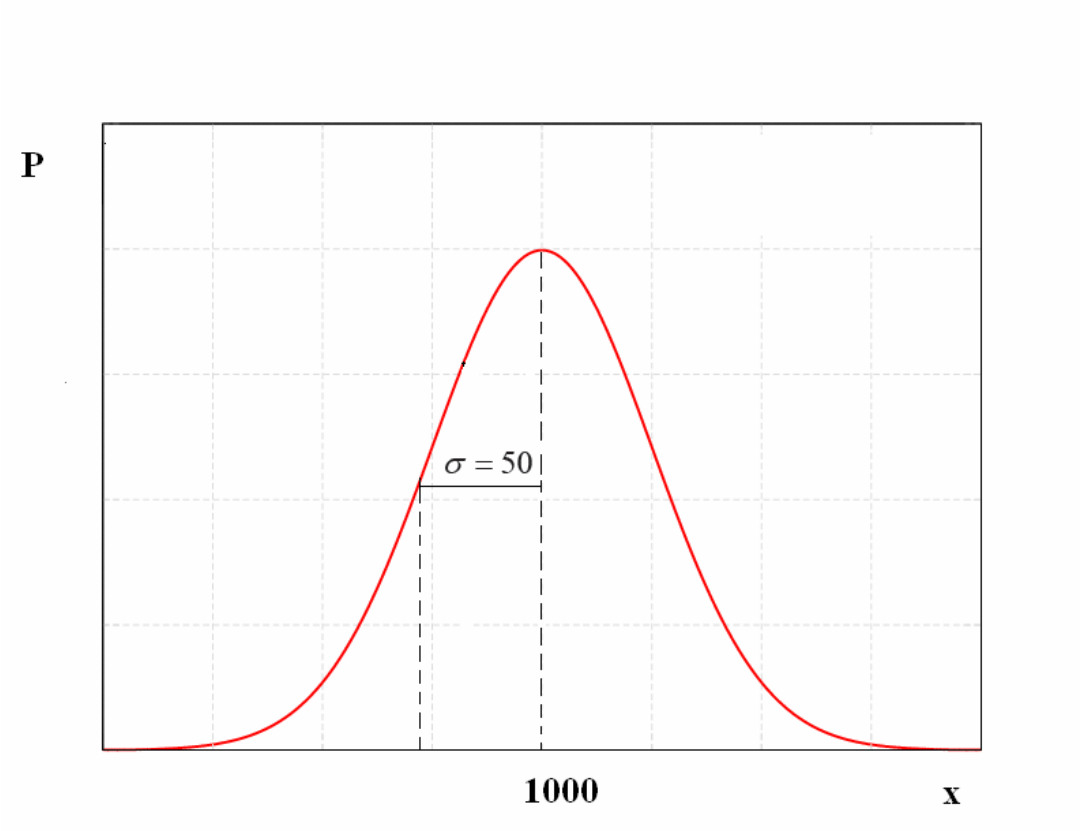


Figure 5.7: probability density function of normal distribution N(1000, 50).

Note that the larger the perturbation interval size in uniform distribution, or the larger the standard deviation σ in normal distribution is, the more diverse our searchable space will be, i.e. the bigger the searchable space we might reach, and the more divergent the search will be performed, which means it might take a longer time to reach a very good solution, but the probability to hit a perfect value is increased.

Besides the empirical observations introduced above, we will show some theoretical analysis about RGS here. Let s_1 and s_2 be two solution components at a local step, furthermore let $x_1 := |h(s_1)|$ and $x_2 := |h(s_2)|$ be the negated normalized score (note that all normalized scores are non-positive, and the lower the score is, the better the quality). Without loss of generality let $x_2 \geq x_1 \geq 0$, which means by the original score s_2 is better than s_1, and $r := \frac{x_2}{x_1} \geq 1$ be the ratio of the two components. Further let $f : \mathbb{R} \to [0,1]$ be the probability density function where the perturbation factor p is drawn. What is the probability that s_1 will have a better (lower) perturbed value than

s_2 despite its disadvantage in the original score? We call this kind of probability the *upset* probability. The *upset* probability can be calculated by the integral below:

$$\begin{aligned} P_{upset}(r, f) &= P(h_p(s_1) \le h_p(s_2)) \\ &= P(p_1 \cdot h(s_1) \le p_2 \cdot h(s_2)) \\ &= P(\frac{p_1}{p_2} \ge r) \\ &= \int_{-\infty}^{\infty} f(x) \cdot (\int_{x \cdot r}^{\infty} f(y) dy) dx \end{aligned} \tag{5.19}$$

Where p_1 denotes the perturbation factor that is generated for s_1, and p_2 the similar. Now let's take uniform distribution within the value interval $[l, u]$ for example, then the probability density function is defined as follows:

$$f_{l,u}(x) = \begin{cases} \dfrac{1}{u-l} & \text{if } x \in [l,u] \\ 0 & \text{elsewise} \end{cases} \tag{5.20}$$

First it is clear that if $r > \dfrac{u}{l}$, the *upset* situation will never happen, since $h(s_2) \cdot l < h(s_1) \cdot u$. We reformulate it as follows:

$$P_{upset}(r, f_{l,u} \mid (l,u)) = 0 \quad \forall r > \frac{u}{l} \tag{5.21}$$

Therefore we can focus only on the case that $1 \le r \le \dfrac{u}{l}$. We substitute (5.20) into the integral in (5.19) to compute the *upset* probability as follows:

$$\begin{aligned} P_{upset}(r, f_{l,u}) &= \int_l^{\frac{u}{r}} f_{l,u}(x) \cdot \left(\int_{x \cdot r}^{u} f_{l,u}(y) dy \right) dx \\ &= \int_l^{\frac{u}{r}} f_{l,u}(x) \cdot \left(\frac{1}{u-l} \cdot y \Big|_{x \cdot r}^{u} \right) dx \\ &= \frac{1}{u-l} \cdot \int_l^{\frac{u}{r}} f_{l,u}(x) \cdot (u - x \cdot r) dx \\ &= \frac{1}{(u-l)^2} \cdot \left(u \cdot x - \frac{1}{2} \cdot r \cdot x^2 \Big|_{l}^{\frac{u}{r}} \right) \\ &= \frac{1}{2r} \cdot \frac{(u - r \cdot l)^2}{(u-l)^2} \end{aligned} \tag{5.22}$$

We set $r_u := \frac{u}{l}$ to be the ratio of the upper bound and the lower bound of the uniform distribution, and divide the numerator and the denominator of the last term of (5.22) with l, then it can be further simplified as follows:

$$P_{upset}(r, f_{l,u}) = \frac{1}{2r} \cdot \frac{(r_u - r)^2}{(r_u - 1)^2} \qquad \forall 1 \le r \le r_u \tag{5.23}$$

And as we can deduce from (5.23), if $r = 1$, i.e. the component s_1 and s_2 have the same score indicating they have the same quality, the chance of either one to be superior is 50%; and the function from (5.23) is strictly decreasing in its admissible domain $1 \le r \le r_u$, the higher the r, which means the larger the quality gap between the scores, the smaller the chance for an upset; if $r \to r_u$, i.e. the quality gap of s_1 and s_2 is approaching the admission limit, the chance for an upset is also tend to 0. So we can conclude, in contrast to Semi-greedy, even the elements with the admissible quality (similar to elements in the RCL), are selected according to their qualities. The better the score, the higher its chance to win.

Using the integral from (5.19) we can compute the *upset* probability of other probability distribution too, such as normal distribution. But the detailed computation will be left to the interested reader.

5.5 The Hybrid of PGreedy and RGS: PARGS

There are two ways of hybridizing PGreedy and RGS, one is to apply PGreedy first to obtain the best parameter setting and then follow it with RGS with respect to the best parameterized scoring function; the other alternative is to embed RGS into each local step of PGreedy, and increase the number of runs at each local step so as to diversify the PGreedy approach.

5.5.1 Combine PGreedy and RGS subsequently

As we have seen in section 5.4, the algorithm RGS is based multiple runs of greedy construction algorithm based on a certain scoring function, and tries to bring some perturbation on it so as to explore it more extensively. Usually the scoring function plays also an essential role in a successful implementation of RGS. Therefore we can first apply PGreedy to find a good parameter combination, and then follow by an extensive exploration of this scoring function with RGS. This usually performs

better than applying RGS to the standard greedy scoring function, as well as finds a better solutions than the ones returned by PGreedy.

5.5.2 Embed RGS in PGreedy at local steps

The second way to hybridize PGreedy and RGS is to combine them together from the beginning. One drawback of PGreedy is that, in each greedy construction it applies strictly only the same scoring function, which limits the searchable space. Hence in each local step of the parameterized greedy construction we use RGS to extend the searchable space. And we spend some more iterations for every parameter combination during the parameter tuning due to the perturbation by RGS.

Note that PGreedy and RGS are both diversification mechanisms, the hybrid of PGreedy and RGS in such a way will considerably increase the diversity of the searchable space, while at the same time weaken the power of the parameterization somehow. Hence it does not necessarily improve the solution quality. It really depends on the trade-off of intensity and diversity of the searchable space, and should be only applied when the searchable space is too concentrative, and needs to bring in distraction.

One possible approach to improve the implementation is to control the number of greedy construction trials of each parameter setting adaptively, instead of using a fixed number of constructions. For a simple adaptive implementation, we propose the following approach: for a given parameter setting, we stop running trial at this parameter setting immediately if the first constructed value is worse than the median (50% Percentile), and elsewise continue to run 3 trials; further if the best constructed solution in the last 3 trials is better than the third quartile (75% percentile), we continue to run 10 trials, and elsewise stop immediately.

5.6 Starting time propagation

The starting time propagation problem arises when there are a series of starting time precedence constraints between two distinct customer nodes. Starting time propagation is applied in our practice for two purposes, one is to apply it at the beginning to check the feasibility of the instance; besides, as we also mentioned in the description of the greedy construction, constraint propagation is also applied during each construction local step to update the starting time of other corresponding the customers.

We first specify our starting time propagation problem into a special class of linear inequality system called IP2, and then present it in a graphical model. A more complete and sophisticated introduction of this topic can be found in PhD thesis of Fuegenschuh 2005.

5.6.1 The simplified monotone IP2 and its graphical model

In the following linear inequality system, each constraint has at most two non-zero coefficients, and it is given a name called IP2 (Integer Programs with two variables per row):

$$\text{(IP2)} \qquad \begin{aligned} \min \quad & \omega^{\mathrm{T}} x \\ \text{s.t.} \quad & a_k x_{i_k} + b_k x_{j_k} \geq c_k, \quad \forall k \in \{1,...,m\} \\ & \underline{x} \leq x \leq \bar{x} \\ & x \in \mathbb{Z}^n \end{aligned}$$

System (IP2) is called monotone if $a_k \cdot b_k < 0$ for all $k \in \{1,...,m\}$. In our application a substructure for constraint propagation exactly makes use of the following simplified monotone IP2 with $a_k = 1$ and $b_k = -1$ for all $k \in \{1,...,m\}$:

$$\text{(SIP2)} \qquad \begin{aligned} & x_{j_k} - x_{i_k} \geq c_k, \quad \forall k \in \{1,...,m\} \\ & \underline{x} \leq x \leq \bar{x} \\ & x \in \mathbb{Z}^n \end{aligned}$$

It arises, for instance, most commonly when the customer j_k should be visited directly after customer i_k, so that the starting time of the pair (i_k, j_k) should satisfy a precedence constraint. However, a precedence pair does not have to show up subsequently, not even in the some path, the precedence relation can be caused by, for instance, some transfer relations, maximum driving time constraints, customer visiting time difference, etc.

In the sequel we will present the linear inequality system (SIP2) into a directed graph, with nodes presenting the customers to be visited, and the directed arcs pointing from the previous customer to the next customer when they satisfy some precedence constraint. There will be in total m precedence pairs, namely (i_k, j_k) defined by (SIP2) for all $k \in \{1,...,m\}$. Such graph presenting the precedence relations is called *Relation Graph*. A relation graph is called *acyclic* if it contains no cycles in the associated undirected graph, and is called *cyclic* otherwise.

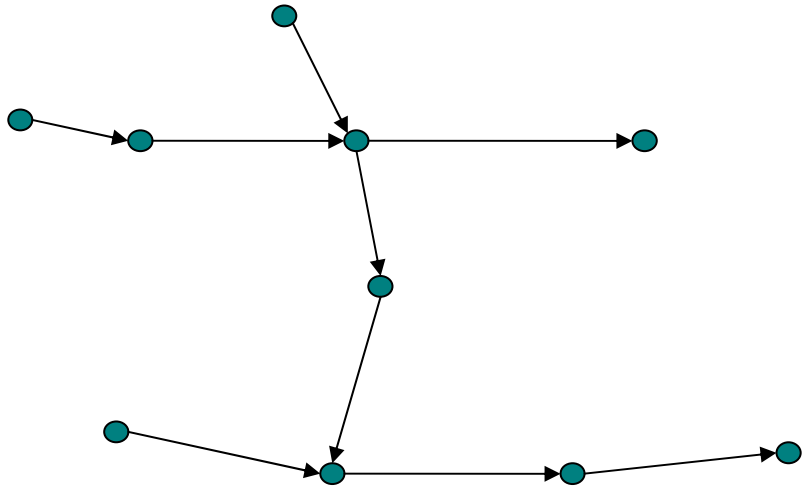


Figure 5.8: An acyclic relation graph

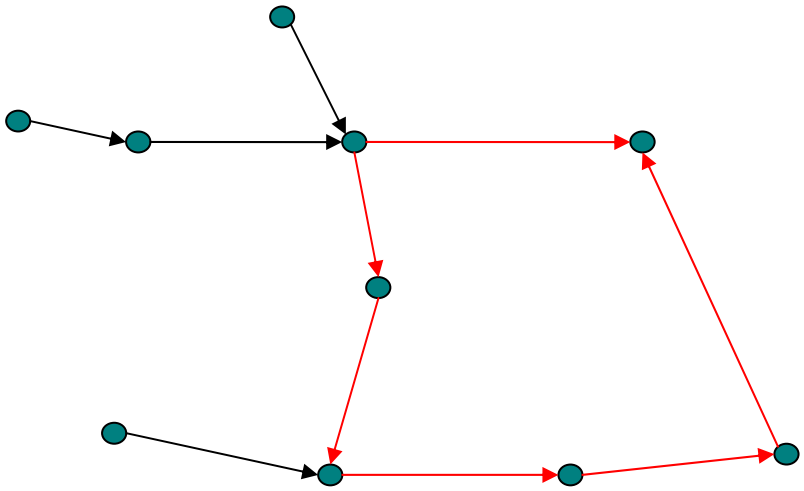


Figure 5.9: A cyclic relation graph

The reason that we distinguish acyclic relation graph from cyclic one is because when a new precedence relation is to be added to the relation graph, e.g. a new customer node is assigned to a specific path at a specific position, the feasibility check of such an assignment is much easier in an acyclic graph than in a cyclic one.

Our task in constraint propagation is to set up the SIP2 model and keep the variable bound updated every time a change occurs in the SIP2 model, e.g. some inequalities being added, deleted, or modified.

Consider the constraint $x_{j_k} - x_{i_k} \geq c_k$ of SIP2. What influence will it be to the

individual bounds of x_{i_k} and x_{j_k}? We calculate as follows:

$$x_{j_k} \geq c_k + x_{i_k} \geq c_k + \underline{x}_{i_k}$$

Now we compare this bound with the previous lower bound $\underline{x}_{j_k}$, and select the larger one as the new lower bound:

$$\underline{x}_{j_k} := \max\{\underline{x}_{j_k}, c_k + \underline{x}_{i_k}\}$$

Similarly, the upper bound of x_{i_k} must satisfy:

$$x_{i_k} \leq x_{j_k} - c_k \leq \overline{x}_{j_k} - c_k$$

and hence

$$\overline{x}_{i_k} := \min\{\overline{x}_{i_k}, \overline{x}_{j_k} - c_k\}$$

And we summarize the procedure above in the following routine *oneOnOneImpact*.

oneOnOneImpact ($\underline{x}, \bar{x}, i, j, k$)

Input: bounds $\underline{x}, \bar{x}$, variable index *i,j,* inequality index *k*. **Output**: updated bounds $\underline{x}, \bar{x}$.
(6) $\underline{x}_{j_k} := \max\{\underline{x}_{j_k}, c_k + \underline{x}_{i_k}\}$ (7) $\overline{x}_{i_k} := \min\{\overline{x}_{i_k}, \overline{x}_{j_k} - c_k\}$

Box 6: constraint propagation of one constraint with two variables

The next routine *oneOnAllImpact* evaluate the impact of $\underline{x}_i$ and $\overline{x}_i$ on all other bounds in $\underline{x}$ and $\bar{x}$.

oneOnAllImpact($\underline{x}, \overline{x}, t$)

Input: bounds $\underline{x}, \overline{x}$, variable index *t*.

Output: updated bounds $\underline{x}, \overline{x}$, or message "system infeasible".

(1) Let $L:=\{t\}$
(2) **while (** $L \neq \varnothing$ **)**
(3) Select and remove *i* from *L*
(4) **foreach (**constraint *k* with x_i and another variable x_j **)**
(5) Call *oneOnOneImpact*($\underline{x}, \overline{x}, i, j, k$)
(6) **if (** $\underline{x_j} > \overline{x_j}$ **)**
(7) return "system infeasible"
(8) **end if**
(9) **if (** $\underline{x_j}$ or $\overline{x_j}$ changed and $j \notin L$ **)**
(10) Let $L := L \cup \{j\}$
(11) **end if**
(12) **end foreach**
(13) **end while**

Box 7: constraint propagation according to the change of one variable on all other constratins

The routine *allOnAllImpact* is easy to interpret, just to call *oneOnAllImpact* iteratively for all customer nodes. This routine is important at the problem preprocessing phase, to help check the problem feasibility.

allOnAllImpact($\underline{x}, \overline{x}, N$)

Input: bounds $\underline{x}, \overline{x}$, the customer node set *N*.

Output: updated bounds $\underline{x}, \overline{x}$, or message "system infeasible".

(1) Let $L:=N$
(2) **while (** $L \neq \varnothing$ **)**
(3) Select and remove *i* from *L*
(4) Call *oneOnAllImpact*($\underline{x}, \overline{x}, i$)
(5) **end while**

Box 8: constraint propagation with all variables on all constraints

Theorem 3: *Given an (SIP2) with m inequalities and n variables and* $U := \max\{\overline{x_i} - \underline{x_i} : i = 1,...,n\}$. *Then either a feasible solution can be computed, or it can be shown that no solution exists, both in O(nU) time.*

Proof Outline: since the (SIP2) is clearly monotone, if the routine *allOnAllImpact* is applied, every time when it comes to the loop in line (4) of Box 2, it will either stop searching or narrow the value interval of by at least 1 of at least one variable. So we can at most enter this loop *nU* times, and the operation within the loop takes constant time, which concludes the time bound *O(nU)*. If the infeasibility message occurs, then the system is clearly infeasible, otherwise, after the routine *allOnAllImpact* ends, all the *m* inequalities are satisfied. Because of line (6) of Box 2, $\overline{x_i} \geq \underline{x_i}$ for all $i = 1,...,n$. Then $x := \underline{x}$ is a feasible solution of (SIP2). □

Note that an (SIP2) with *m* inequalities and *n* variables corresponds to the relation graph with *n* nodes and *m* directed arcs.

5.7 Local Search

The local search consists of two different steps which are carried out iteratively, until no further improvement is found. First, we do a local tour optimization, where we apply classical node insertion and node exchange steps between two routes. Second, we do a local exchange on the vehicle type, where we assign the cheapest possible type to each existing route. Finally, the generated solution is returned.

Node insertion (aka. shift neighborhood**)** is well illustrated as the following graph shows**:**

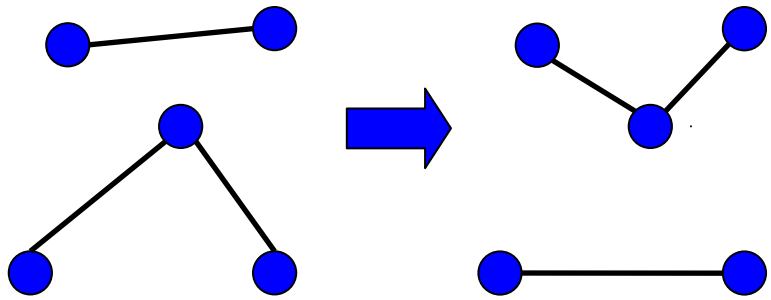


Figure 5.10: node insertion neighborhood between two paths.

Node exchange (aka. swap neighborhood**)** is well illustrated as the following graph

shows:

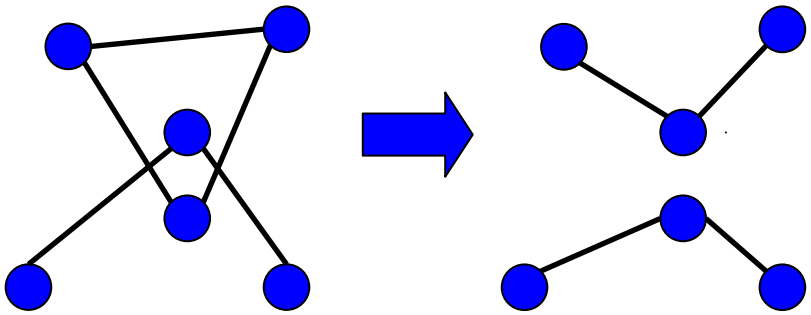


Figure 5.11: node exchange neighborhood between two paths

Vehicle type Exchange neighborhood is easy to imagine. If all the tasks assigned to a vehicle can be also taken by a cheaper type of vehicle, in the sense of objective cost, then the cheaper vehicle will be used instead.

As far as we are aware of, the local search procedure is not very effective for the Heterogeneous Vehicle Routing Problem with Time Windows. Since reducing the number of vehicles used is the first priority in our optimization goal, but to this end very few neighborhood can help except for the node insertion. Even the node insertion need special guidance to achieve the goal of reducing fleet size.

Some more effective neighborhood relations for VRP are introduced in details in Kindervater and Savelsbergh 1997. Another direction worth mentioning is a simple and effective metaheuristic strategy called Iterated Greedy (IG), which destroy part of the solution and try to reconstruct it again, see Ruiz and Stuetzle 2005. It is to expect, a hybrid of pgreedy and IG will be powerful in practice. Also Dorigo and Stuetzle 2004 has reported that a variance of Ant Colony Optimization (ACO) algorithm, named ACS, has yield state-of-the-art results for Vehicle Routing Problem.

5.8 Application examples

We applied our general PGreedy strategy to four different real-world applications of heterogeneous VRP with time windows. Besides the general VRP structure, each of the four problem has individual constraints depending on the type of application. The main implementational effort was to adapt PGreedy to each individual situation.

5.8.1 Home Health Care Services

In order to reduce fleet size at the same time also balance the work load, the local

search procedure is crucial for this project. After building a set of routes with our PGreedy construction search, a two-phase local search is performed, applying a neighborhood of node insertion followed by a node exchange in each phase. In the first phase we iteratively insert nodes (patient visits) from the nurses with less workload to those with heavier workload, in the hope of reducing the number of nurses. After the number of nurses cannot be reduced anymore, in the second phase conversely, we iteratively insert nodes from nurses with heavier workload to those with less workload, to balance among nurses. In order to ensure each step of local search to be feasible and efficient, a linear constraint propagation is performed, including intra-route propagation for time windows and maximum working time, and inter-route propagation for inter-visit day difference.

5.8.2 School Taxi Routing

We currently tackle the problem using our PGreedy heuristic. Our first implementation is to pick up pupils always from the same school into one car, and make sure they arrive at school within their driving time tolerance. Thereafter we implement it in a way that pupils from different schools can also be on one car simultaneously, which is more subtle, since during each route construction step one has to check all permutations of to be reached schools to determine a candidate node's feasibility. But experiment confirms that allowing pupils from different schools on board results in a noticeable route efficiency enhancement, in other words, reduced fleet size and cost.

5.8.3 Cyclic Locomotive Scheduling

We developed a randomized PGreedy-type heuristic that constructs feasible solutions from scratch. This heuristic consists of two steps, where the output of the first step is taken as input for the second.

Step 1: Wagon Transfers.
In the first step, the heuristic seeks to determine a minimum number of missed wagon transfers, because they have the largest impact in the objective function. To this end, the set P of all wagon transfers is split into three disjoint subsets $P = P^w \dot{\cup} P^g \dot{\cup} P^b$, called white, gray and black, respectively. The actual assignment of transfers to one of the three groups is based on the influence of the transfers on the starting time windows of trains *i* and *j*.

- **White:** the pair $(i, j) \in P^w$ if the wagon from train *i* will wait less than 720 minutes before it is picked up by train *j* without shrinking the time window of the 2 related trains.

- **Black**: the pair $(i, j) \in P^b$ if the wagon cannot be transfered from train *i* to train *j* in time, under all circumstance.

- **Gray:** the rest of the wagon transfers which can be either picked up in time or not, depending on how the starting time of the 2 related train *i* and *j* will be selected.

For the white transfers, they become feasible wagon transfers, and the black transfers are considered to be infeasible for good. It remains to decide which of the gray one can be considered as white and which as black. This is done iteratively by selecting one element $(i, j) \in P^g$, and tentatively setting it to be white, and updating the time windows by constraint propagation. If the system is feasible, we remove *(i, j)* from P^g and put it to P^w. Otherwise if the system is infeasible, we also remove *(i, j)* from P^g but now put it to P^b. Afterwards the nextelement from P^g is selected, as long as this set is non-empty. Note that each time we move an element from P^g to P^w, we keep the updated (hence smaller) time windows on the starting times of the trains.

In the end we obtain a disjoint partition of P into two subsets P^b and P^w. Since the size of these two sets depends on the actual ordering in which elements from P^g are selected, we repeat the overall procedure a number of times, each time with different ordering among the elements of P^g. The output of this first Stepp of the heuristic is the overall best partition, i.e. the partition with the smallest set P^b and the largest set P^w.

Step 2: Operating Costs.

In the second part of the heuristic we seek for a minimum number of locomotives and a minimum total length of all deadhead trips, given the wagon transfers from the first step as fixed input data. The heuristic iteratively assigns locomotives to trains and deadhead trips. For this, the following steps are repeated until all trips are assigned.

- **Locomotive selection:** We first make a selection of the locomotive type with which the heuristic shall proceed. We hereto count the number of available deadhead trips in *A* per class *b*. This number A_b is divided by the starting cost of the as c_b of this locomotive class *b*. The ratio A_b / c_b can be interpreted as a value for the flexibility of the particular type of locomotive; the higher this number, the more useful it is. We randomly select a type class *b*, with a bias corresponding to the flexibility ratios.

- **Start train selection:** Among all trains which are so far unassigned we seek one

with a high number of possible deadhead trips of the same type as our selected locomotive type class *b*. Again, we select this start train *i* randomly, with a bias corresponding to the number of deadheads per node.

- **Starting time selection:** We fix the starting time of the so selected train. The starting time t_i can be taken from the interval $[\underline{t_i}, \overline{t_i}]$. We select t_i in such way that the arrival time is the earliest possible. (Note that we have netload dependent driving times, which means that we have to subdivide the interval corresponding to the time slices.) Once the starting time is selected, we might have to evaluate the influence on the other starting time windows. Such an influence occurs if the train *i* is involved in some wagon transfers. Using constraint propagation the starting time interval of related trains might change in this case.

- **Deadhead trip selection:** The deadhead trip for the locomotive to the start of the next train is also selected randomly. Here the process is biased according to the length of the deadhead trip.

In the above heuristic all steps rely on random selections. Hence we repeat the whole procedure several time (up to a few hundred, depending on the given time limit and the size of the instance), and in the end output the best overall solution.

5.8.4 Multi-Depot Locomotive Scheduling -- Handling Coupling Trips

One of the most important and distinctive feature of the Multi-Depot Locomotive Scheduling project is the coupling trip feature. How to handle the coupling trip feature is essential for a successful implementation of the greedy construction heuristic. Here we make use of a special non-linear scoring function with an extra parameter *double penalty*.

Our experiment shows that for the heavy shipments, i.e. the shipments that requires a twin combination of Type I locomotives or a single Type II locomotive, we should put strong guidance to encourage them to be pulled by Type II locomotives, and meanwhile avoid Type I locomotives to serve the heavy ones. There are 3 reasons for doing so,

1) the trip with one Type II locomotive will definitely consume less energy than the sum of two Type I locomotives taking the same trip;
2) once a heavy shipment is taken by a Type I locomotive, this shipment will become unlikely to be selected by a second Type I locomotive, since its starting time is fixed by the first locomotive;

3) if Type I locomotives are run out, but some heavy shipments are still taken by just one Type I locomotive, the partial solution will become infeasible, since in this case the remaining Type II locomotives are helpless.

Unfortunately the second argument above makes the infeasibility disaster described in the third argument occur often.

To overcome the coupling trip difficulty, we propose the following 2 steps:

1) increase the probability that Type II locomotive depots should be selected at the early construction phase, to avoid the third catastrophe. This can be easily achieved by enlarging the parameter *capacity weight* λ used in equation (5.1).
2) introduce a new parameter *double penalty* ω to stimulate Type II locomotives to pull heavy shipments, as well as Type I locomotives to stay away from them. The scoring function of selecting next shipment is modified as follows:

$$score'_{next}(j\,|\,(i,k)) := \begin{cases} \dfrac{score_{next}(j\,|\,(i,k))}{\omega}, & \text{if } k=2 \wedge s_j \text{ heavy} \\ score_{next}(j\,|\,(i,k))\cdot\omega, & \text{if } k=1 \wedge s_j \text{ heavy and not taken} \\ \dfrac{score_{next}(j\,|\,(i,k))}{\omega}, & \text{if } k=1 \wedge s_j \text{ heavy and partly taken} \\ score_{next}(j\,|\,(i,k)), & \text{else} \end{cases} \tag{5.24}$$

where $s_j \in S$, $k \in T$, and the *double penalty* $\omega > 1$. The equation (5.24) says, a heavy shipment should be favored for a Type II locomotive, and should be punished by a Type I locomotive, with exception that if the heavy shipment is partly taken, i.e. it is already taken by one Type I locomotive, then it should be favored in a second Type I locomotive's selection phase.

Note that a reasonable *double penalty* ω should be greater than 1, but its value should not be too large, otherwise some "lazy Type II locomotives", i.e. the type II locomotives that idles too long in order just to catch the next heavy shipment, will occur.

6 Computational Results

Computational results for the 4 practical rich vehicle routing problems will be presented in this chapter.

6.1 Solution Process of our software system for Rich Vehicle Routing Problems

The software is designed in an object-oriented manner, implemented using Java programming language. It is also designed in a modulized way so that each specific application project can be integrated into the software system with as few modifications to the core part of the system as possible.

In general our solution process adopts two approaches to solve the rich vehicle routing problems, one way is to solve it with mathematical programming formulation, under the support of the open-source modeling tool zimpl and the commercial MILP solver Cplex; a second way is to attack it by the self-developed PGreedy heuristic methods.

A third approach is to provide the best heuristic solution to the MILP solver as a starting value, in the hope of obtaining better feasible solutions, and in the meanwhile also helping the solver gain a better dual bound to the problem. Experiment shows, feeding the MILP solver with initial feasible solution usually improves the dual bound, or solves the problem to optimality with shorter computation time.

Exception is the Multi-Depot Locomotive Scheduling project, which is developed in Simens AG, Munich in 2005, makes use of ILOG OPL Studio with MILP solver Cplex integrated.

The general solution process is depicted in the following flow chart:

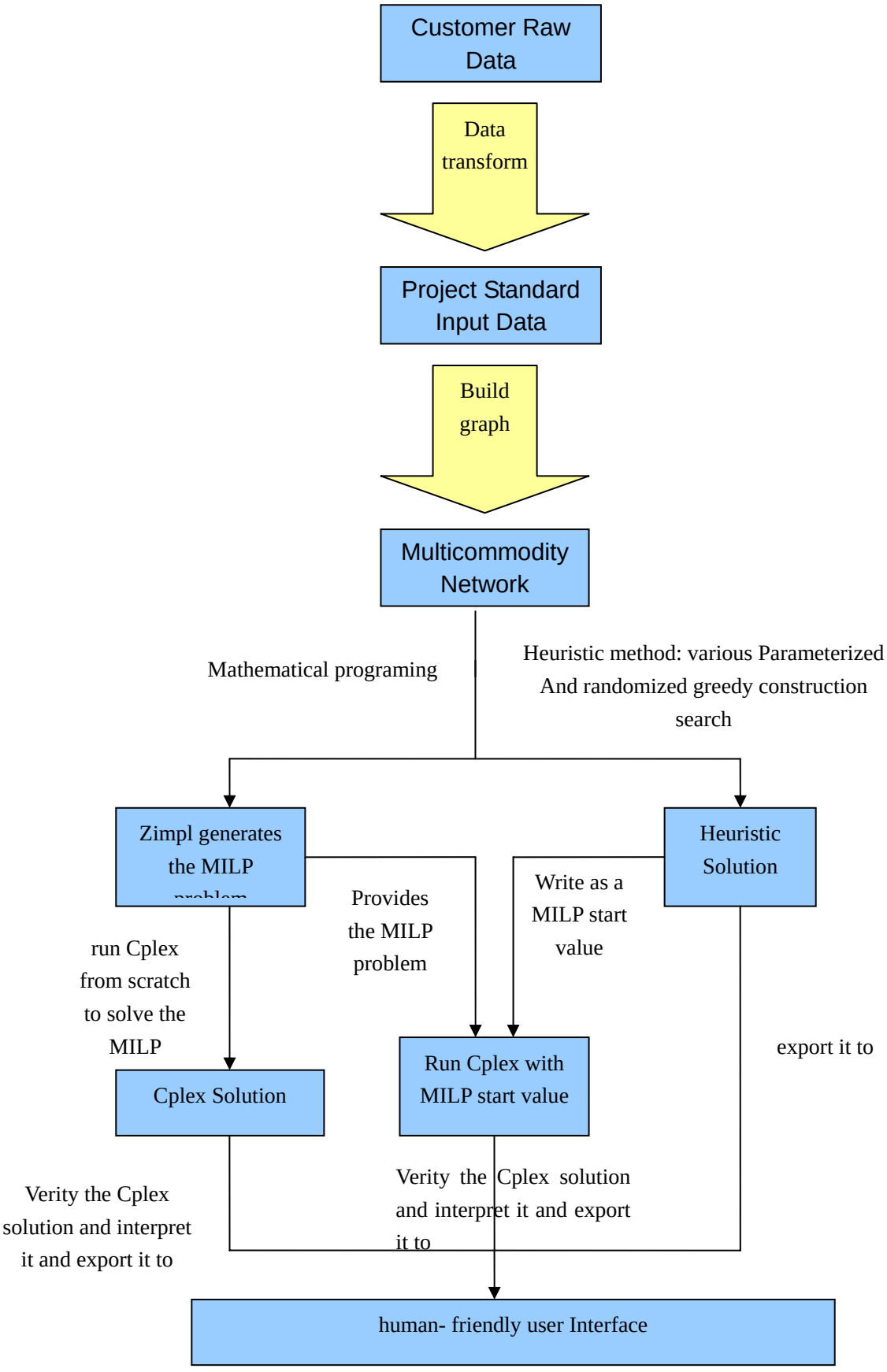


Figure 6.1: The solution process.

6.2 Solving the MILP model with zimpl, Cplex and OPL

Zimpl (Zuse Institute Mathematical Programming Language) is a modeling language to translate the mathematical model of a problem into a linear or (mixed-) integer mathematical program expressed in .lp or .mps file format which can be read and (hopefully) solved by a LP or MILP solver (Koch 2007).

ILOG OPL Studio is a commercial integrated development environment for modeling and solving combinatorial optimization problems with mathematical programming. It has an MILP solver Cplex integrated. It is easy to use, since the modeling and the solving functionalities are combined together. OPL is also a modeling language for mathematical programming. A brief user guide to the OPL language can be found by Martin (2002).

ILOG Cplex is a commercial software that solves linear programming, quadratic programming, quadratically constrained programming and mixed integer programming problems. See ILOG CPLEX (2006) for more details.

In the our solution process, the zimpl or OPL input data file is generated by our java software. After the MILP solver yields a solution, either partial solution or final solution, it will be read into our software, verified, interpreted and transformed into human-readable format and displayed on our user interface.

6.3 Test Environment

Our software is written in Java programming language, and is compiled and tested using JDK 1.5.0_06, on a personal computer running Windows XP as operating system, with hardware support from AMD AthlonXP 1800+ (1.5GHz) and 256MB DDR RAM.

The zimpl and ILOG Cplex 10 is run on the computer with 32 GB RAM, 8x 2.4GHz AMD Opteron 880, Suse 10.2 Linux OS.

6.4 Solving Mobile Nurse Scheduling for Home Health Care Services

In this section we first introduce the real-world instance given from our partner Diakonie and its results, and then show a set of generated instances of different sizes and their computational results.

6.4.1 The real-world input data Diakonie

Only one data set is available from our industry partner Diakonie, a local health care service provider. This instance, which is named “Diakonie”, consists of a real-world weekly mobile nurse schedule that has been currently in use.

From this data file we have almost every detail for the problem model, e.g. treatment preferred day and duration, patient_specific time window, nurse competence and working time limit, etc. Only one thing is missing, that is the deadhead trip driving time from one customer to another, which needs to be simulated. Fortunately we have the x-y coordinate of each patient and the nurse depot. And the driving time is computed by the Euclidean distance between every two customer locations, multiplied by a suitable factor. This driving time multiplier factor is empirically selected to be 400, which through our observation fits best to the reality. And we assume the driving time between each two customers is unvaried among nurses. Note that in such case the distance matrix will be symmetric, which means the driving time from customer A to customer B is always identical with driving from B to A. It is possible reduce the distance matrix to a upper (or lower) triangular form to save memory space, but we don’t encourage it for two reasons, firstly the distance matrix will be read very frequently (asymptotically proportional to the algorithm complexity) and keeping the full matrix can save an index comparison judgement by each read operation; secondly the memory space is far from being our computation bottleneck.

The real-world instance Diakonie contains 99 patients, 460 treatments, and 9 nurses. The number of pairs with inter-visit day difference is already greatly reduced in our dataset preprocessing phase, and 17 pairs are still left. Each treatment has a patient-specific time window, which is given in advance. The size of the time window varies greatly, ranging from 30 minutes to 360 minutes. And the large time window dramatically increases the difficulty of solving the problem optimally by MILP solver. More computational details about the instance size are as follows, including the size of the MILP model listed in the last three column:

Instance	#patient	#treatment	#nurse	#iv-pair	Time window size (min)
Diakonie	99	460	9	17	30 - 360

Table 6.1: The technical details of the Diakonie instance.

In order to have a first impression of the difficulty of the corresponding MILP problem, the size of the MILP model is listed below. The first 3 columns tells the size of the MILP model directly generated from the mathematical modeling tool zimpl, and the 3 following columns shows its reduced size after the Cplex preprocessing phase, the last column shows the Cplex preprocessing time.

Instance Name	Resulting MILP model from zimpl			Reduced MILP by Cplex preprocessing			
	#variable	#constraint	#non-zero	#variable	#constraint	#non-zero	Time(sec)
Diakonie	8501308	8909900	69791792	404310	338663	2649009	194

Table 6.2: The MILP model size of the Diakonie instance.

As is mentioned above, the MILP problem is hard for the commercial solver Cplex, not only because of its problem size, but also the large time window on each treatment, the resulting big-M constraints are well-known to have a bad dual bound with the LP relaxation. Exhaustive experiments show that the dual bound (the lower bound in the minimization case) can not be obtained within a reasonable time, i.e. 6 hours as our computation deadline. Therefore to the best of our effort it can be solved only heuristically without any known dual bound (except for a trivial lower bound 0).

A good scheduling solution can be found by our PGreedy heuristic algorithm within 5 minutes. Comparing with the original manual nurse schedule, we are able to save one nurse out of nine. Unfortunately the total working time as the secondary optimization goal is not comparable, since the working time in the original schedule is unknown. More details about the solution are listed below:

Instance Name	Computation time (sec)	Cost	Nurses (normal / small)	$\sum$	Total working time	Average working time (per nurse)
Diakonie	300	7713293	7 / 1	8	13293	1662

Table 6.3: The heuristic solution to the Diakonie instance.

6.4.2 The smaller instances extracted from the real-world instance

In order to perform further experiment so that our mathematical MILP model can come into play with the support of the commercial MILP solver Cplex, smaller instances, more specifically easier instances for the MILP solver, are needed. Since one of the goal of the project is also to estimate the computation limitation of such type of problems. To this end we developed an instance generator, to generate smaller instances for the problem.

In order to keep the generated instances still reflecting the reality, i.e. to keep the instances as close to the real-world information as possible, we decided to extract the instances randomly from the Diakonie instance. Note that each patient may have multiple treatments, so we do not extract individual treatments, instead we extract individual patients, with the complete set of treatments that the patients need and the corresponding inter-visit day difference of the patients.

The following lists 6 instances of different sizes extracted from the real-world dataset. The number in the postfix of each instance name refers to the number of treatments included:

Instance	#patient	#treatment	#nurse (deterministic)	#nurse (adaptive)	#iv-pair
dia_22	5	22	12	2	0
dia_75	15	75	12	3	4
dia_110	25	110	12	3	3
dia_153	33	153	12	4	5
dia_210	46	210	12	4	10
dia_285	60	285	12	6	9

Table 6.4: The technical details of the extracted instances.

As can be seen from above there are two different ways of assigning the number of nurses to the problem, the deterministic way and the adaptive way. The deterministic way assigned the nurses according to the available nurse information, unvaried for each instance. While the adaptive way assigned the nurses to each instance adaptively by the at least a feasible schedule so that the feasibility of the MILP is always satisfied.

The number of nurses plays a very important role in our MILP model, since each individual nurse determines a specific commodity layer in our multi-commodity network, hence the size of the network model is proportional to the number of nurses. The influence of the number of commodities in a multi-commodity network model is also investigated in Yuan 2007.

Therefore we have adopted the adaptive way of assigning the number of nurses to each MILP model of each instance. We first apply our heuristic to find the minimum number of nurses needed by the heuristic computation, and assign it to the MILP model. It might lose the provable optimality to some extent, but it works fine in practice, since firstly the nurse combination found by the heuristic is usually quite good already, secondly nurses of the same type are usually exchangeable. Most importantly, this adaptive approach greatly simplifies the MILP model and accelerates the process of solving the relaxation LP problem as well as solving some small instances to optimality.

First we will show the heuristic solution results of the 6 reduced instances introduced above:

Instance Name	Computation time (sec)	Cost	Nurses (normal / small)	$\sum$	Total working time	Average working time (per nurse)
dia_22	300	1702200	1 / 1	2	2200	1100
dia_75	300	2702071	2 / 1	3	2071	690
dia_110	300	2704577	2 / 1	3	4577	1526
dia_153	300	3705336	3 / 1	4	5336	1334
dia_210	300	3706175	3 / 1	4	6175	1544
dia_285	300	5707509	5 / 1	6	7509	1252

Table 6.5: The heuristic results of the extracted instances.

After the heuristic algorithm is performed, the MILP model can be finally determined using the adaptive nurse assignment. The sizes of the adaptive MILP model of the 6 extracted instances are shown below:

Instance Name	Resulting MILP model from zimpl			Reduced MILP by Cplex preprocessing			
	#variable	#constraint	#non-zero	#variable	#constraint	#non-zero	Time(sec)
dia_22	5287	8419	4332	147	167	672	0.02
dia_75	86718	99371	124072	3487	3204	21774	0.61
dia_110	184928	217821	201640	7115	5941	43966	1.26
dia_153	474477	541680	583113	21236	17914	135701	4
dia_210	890634	961610	1363906	43693	37424	284175	11
dia_285	2122515	2134117	3002755	92401	77325	595623	29

Table 6.6: The MILP model size of the extracted instances.

Then we tried to solve the MILP model from scratch by the commercial MILP solver Cplex. For solving the LP relaxation we applied two different strategies, namely the Simplex algorithm and the Interior Point method. The results of the two LP relaxation solving strategies are listed below. It shows clearly that for this MILP model, the Cplex implementation of Interior Point method totally outperforms the Simplex method in 5 out of 6 instances and a draw in the instance “dia_110”. And it is also to be noticed that Cplex cannot even find a feasible solution for instances with more than 110 treatments within 6 hours. Therefore heuristic will play a very important role in solving this problem.

Instance Name	Simplex				Interior Point			
	Computation time (sec)	Best integer	Best node	gap	Computation time (sec)	Best integer	Best node	gap
dia_22	33	1702200	1702200	0%	27	1702200	1702200	0%
dia_75	21600	-	1003438	-	21600	2702103	2000098	26%
dia_110	21600	-	1000359	-	21600	-	1000359	-
dia_153	21600	-	1000000	-	21600	-	1333692	-
dia_210	21600	-	700633	-	21600	-	701800	-
dia_285	21600	-	0	-	21600	-	1000000	-

Table 6.7: MILP solver (Cplex) results with Simplex or Interior Point Method to solve the subsidiary LP relaxation.

In the sequel we applied the Cplex on the MILP model again, and only with the Interior Point method as the LP relaxation solving strategy other than the Cplex default Simplex method. And this time we added a heuristic solution as a MILP starting value provided to Cplex. The MILP starting value is provided in the hope of helping the branch and bound procedure to cut off more unnecessary subbranches when the bounding value of the subbranch exceed the global upper bound.

As can be observed from the diagram below, in the instances “dia_75”, “dia_110” and “dia_153” have both their upper bound and lower bound improved, comparing with the starting heuristic value and the pure Cplex value of “Best node” (lower bound) listed above.

Instance Name	Primal heuristic		Dual (Cplex)				Final Solution		
	Cost	Root gap	Comp. time (sec)	Best integer	Best node	gap	Nurses	$\sum$	Total working time
dia_22	1702200	0%	30	1702200	1702200	0%	1 / 1	2	2200
dia_75	2702071	26%	21600	2702001	2000144	26%	2 / 1	3	2001
dia_110	2704577	63%	21600	2704525	1000359	63%	2 / 1	3	4525
dia_153	3705336	60%	21600	3705336	1500359	60%	3 / 1	4	5336
dia_210	3706175	81%	21600	3706175	701800	81%	3 / 1	4	6175
dia_285	5707509	82%	21600	-	1000000	82%	5 / 1	6	7509

Table 6.8: The final results combining MILP solver Cplex with the heuristic results as MILP start values. For the subsidiary LP relaxation we apply Interior Point method for it.

The size of the Branch-and-Bound of the extracted instances is listed in the table below:

Instance Name	Comp. time (sec)	#visited nodes	#nodes left
dia_22	30	141572	10897
dia_75	21600	42252	36982
dia_110	21600	7712	5867
dia_153	21600	1234	1059
dia_210	21600	110	105
dia_285	21600	0	1

Table 6.9: The size of the Branch-and-Bound tree of the extracted instances.

Unfortunately for the instances larger than “dia_75”, the gaps are still vast, around 60% to 80%. More work should be done to improve the dual bound, including model reformulation and adding more problem specific valid inequalities. It is already on our future agenda.

6.5 School Taxi Routing for Handicapped Pupils

In this section we first introduce the real-world instances given from our industry partner ZIV (Zentrum für integerierte Verkehrssysteme, Darmstadt), and the computational results that are generated from our system will be presented. And then a set of generated instances of different sizes and their computational results will be shown.

6.5.1 The real-world input data

Two real-world instances are given from our industry partner ZIV, named “Bentheim” and “Aurich”, after the name of two German counties. Some technical details about the instance sizes are listed below:

Instance	#pupil	#school	#car type
Bentheim	76	7	5
Aurich	696	53	11

Table 6.10: The technical details of the 2 real-world instances.

In the “Aurich” instance there are about 100 different types of vehicles, but the differences among them are mostly minor, so it is regrouped and categorized into 11 types.

Note that in practice, the time for the pupils to get off at the school is variable, usually if there is no wheelchair on board, the get-off time will be 1 minute, and if there is wheelchairs on board, it will usually take 5 minutes. The results shown in the following tables are the results with variable school get-off time. And all the results are acquired within 5 minutes’ computation time.

A local search procedure is integrated into the our PGreedy approach, and it performs vehicle type exchange for each path. As is shown below, the local search step results in a great saving in the total cost:

Instance	Construction		Local search	
Instance	#vehicle	#cost	#vehicle	#cost
Bentheim	12	1030	12	982
Aurich	110	6471	110	5683

Table 6.11: The heuristic solutions with and without local search.

In the experiments hereafter, all the heuristic solution will undergo the local search step. Our first construction implementation is always to pick up pupils from the same school into one car, and make sure they arrive at school within their driving time tolerance. Thereafter we implement it in a way that pupils from different schools can also be on one car simultaneously, which is more subtle, since during each route construction step one has to check all permutations of to be reached schools to determine a candidate node's feasibility. But experiment confirms that allowing pupils from different schools on board results in a noticeable route efficiency enhancement, in other words, reduced fleet size and cost.

Instance	One school		More schools	
Instance	#vehicle	#cost	#vehicle	#cost
Bentheim	12	982.0	12	966.4
Aurich	110	5683	105	5466

Table 6.12: The heuristic solution allowing pupils from only one school on board and more schools on board.

The final best heuristic solutions to the two real-world instances are listed below.

Instance	Comp. time (sec)	Cost	vehicles	$\sum$
Bentheim	300	966.4	3/2/7/0/0	12
Aurich	300	5466	42/0/0/9/25/4/16/7/2/0/0	105

Table 6.13: The final heuristic solutions for the real-world instances.

As far as we are informed from our industry partner, the instance "Aurich" is currently using a manual schedule with 130 vehicles. And our partner ZIV also develops a optimization software called "OptiTours" for solving the problem. This software applies Genetic Algorithm. Their implementation was reported to be running several hours long with a best solution found using 109 vehicles for Aurich, and 13 vehicles for Bentheim. Our PGreedy heuristic outperforms their implementation in these 2 instances:

Instance	PGreedy	Current schedule	Saving potential	OptiTours by ZIV
Aurich	105	130	19.2%	109
Bentheim	12	-	-	13

Table 6.14: Results comparison to current schedule and OptiTours.

The instance Aurich is too large to be solved by MILP solver, zimpl cannot even build up an lp file from the input data since it takes zimpl too long to transform this instance into a MILP model for Cplex.

However, the instance Bentheim is able to be transformed into a MILP model and the size of it is listed below:

Instance Name	Resulting MILP model from zimpl			Reduced MILP by Cplex preprocessing			
	#variable	#constraint	#non-zero	#variable	#constraint	#non-zero	Time(sec)
Bentheim	498126	1939507	7548562	285916	1006873	4101522	32

Table 6.15: The MILP model size of the instance Bentheim.

The MILP instance is already too large to be solved to optimality. Our following approach is to apply the MILP solver Cplex to solve the problem from scratch. Similarly as in the mobile nurse scheduling project, we tried different LP relaxation solving strategies, Simplex and Interior Point method, see below:

Instance Name	Simplex				Interior Point			
	Computation time (sec)	Best integer	Best node	Root relaxation time (sec)	Computation time (sec)	Best integer	Best node	Root relaxation time
Bentheim	21612	-	382.6861	5104	24062	-	-	∞

Table 6.16: MILP solver (Cplex) results for Bentheim with Simplex or Interior Point Method to solve the subsidiary LP relaxation.

It is clear that for this instance Simplex performs better than Interior Point method, in contrast to the results in the mobile nurse scheduling. In the sequel we applied Cplex with Simplex method, and together with a MILP start value we gained from our heuristic. Since the MILP model cannot handle the variable school get-off time very well, we have to fix the school get-off time to 5 minutes, despite whether there are wheelchairs on board.

Instance Name	Primal heuristic		Dual (Cplex)				Final Solution
	Cost	Root gap	Computation time (sec)	Best integer	Best node	gap	#vehicle
Bentheim	1056.24	64 %	21600	1056.24	382.68	64%	16

Table 6.17: The final results of Bentheim combining MILP solver Cplex with the heuristic results as MILP start values.

The instance Bentheim is already considered to be too large for the MILP solver. Finding better dual bound is an important work for the future agenda. In the sequel we will generate some smaller instance and find out the computation limit for this type of problem.

6.5.2 The extracted instances

In order to perform further experiment so that our mathematical MILP model can come into play with better support of the commercial MILP solver Cplex, smaller instances, more specifically easier instances for the MILP solver, are needed. As in the mobile nurse scheduling project we developed an instance generator, to generate smaller instances for the problem.

The smaller instances are extracted from the Bentheim instance, so that the number of schools can be controlled under a low number.

The following lists 6 instances of different sizes extracted from the real-world Bentheim dataset. The number in the postfix of each instance name refers to the number of pupils included:

Instance	#patient	#school	#vehicle (deterministic)	#vehicle (heuristic)	#vehicle (adaptive)
ben_3	3	2	50	2	7
ben_10	10	3	50	4	9
ben_20	20	4	50	6	11
ben_30	30	6	50	9	14
ben_40	40	7	50	11	16
ben_50	50	7	50	13	18

Table 6.18: The technical details of the extracted instances.

As we have discussed in the mobile nurse scheduling project, there are two different ways of assigning the number of nurses to the problem, the deterministic way and the adaptive way. The deterministic way assigned the vehicles according to the available vehicle information, unvaried for each instance. The adaptive way assigned the nurses to each instance adaptively based on a feasible schedule, without losing the feasibility

of the MILP model. The influence of the number of commodities in a multi-commodity network model is also investigated in Yuan 2007.

We have adopted the adaptive way of assigning the number of vehicles to each MILP model of each instance. We first apply our heuristic to find the minimum number of vehicles needed by the heuristic computation, and then assign one more vehicle of each type to each instance, in the hope of keeping the optimality of the solution. It is true that in such case the optimality cannot be strictly proved anymore, but it works fine in practice, since firstly the vehicle combination found by the heuristic is usually quite good already, secondly vehicles of the different type are usually exchangeable. Most importantly, this adaptive approach greatly simplifies the MILP model and accelerates the process of solving the relaxation LP problem as well as solving some small instances to optimality.

First we will show the heuristic solution results of the 6 reduced instances introduced above:

Instance Name	Computation time (sec)	Cost	Vehicle	$\sum$
ben_3	300	85.3	1/1/0/0/0	2
ben_10	300	239.6	3/0/1/0/0	4
ben_20	300	366.5	5/0/1/0/0	6
ben_30	300	573.3	5/1/3/0/0	9
ben_40	300	704.0	7/0/4/0/0	11
ben_50	300	814.5	8/1/4/0/0	13

Table 6.19: The heuristic results of the extracted instances.

After the heuristic algorithm is performed, the MILP model can be finally determined using the adaptive vehicle assignment. The sizes of the adaptive MILP model of the 6 extracted instances are shown below:

Instance Name	Resulting MILP model from zimpl			Reduced MILP by Cplex preprocessing			
	#variable	#constraint	#non-zero	#variable	#constraint	#non-zero	Time(sec)
ben_3	3057	7212	23482	1956	3262	12162	0.03
ben_10	4341	14368	55303	2735	7161	29048	0.07
ben_20	19401	70282	276643	11751	37432	150592	0.47
ben_30	53835	201342	762192	30772	102576	413944	1.47
ben_40	107617	409168	1519814	59648	200392	814368	3.75
ben_50	187319	719354	2757730	107150	366726	1493596	7.79

Table 6.20: The MILP model size of the extracted instances.

Then we tried to solve the MILP model from scratch by the commercial MILP solver Cplex. For solving the LP relaxation we applied two different strategies, namely the Simplex algorithm and the Interior Point method. But the Interior Point method is

only applied to 3 of the instances, “ben_10”, “ben_50”, “ben_76” (the complete set of Bentheim instance). The results of the two LP relaxation solving strategies are listed below. It shows clearly that for this MILP model, the Cplex implementation of Simplex method totally outperforms the Interior Point method in all the 3 instances. It also shows that heuristic will play a very important role in solving this problem, for providing initial feasible solutions.

Instance Name	Simplex				
	Computation time (sec)	Best integer	Best node	gap	Root relaxation time (sec)
ben_3	1.98	85.27	85.27	0%	0.04
ben_10	21600	222.22	174.22	21.6%	0.2
ben_20	21600	-	181.98	-	4.3
ben_30	21600	-	232.18	-	73
ben_40	21600	-	278.26	-	353
ben_50	21600	-	307.99	-	762
ben_76	21600	-	382.69	-	5104

Table 6.21: The MILP solver (Cplex) results with Simplex to solve the subsidiary LP relaxation.

And the Interior Point method for the following 3 instances with a (relatively) small instance ben_10 and two largest instances ben_50 and ben_76. The results show that in contrast to the MILP model of the Mobile Nurse Scheduling shown in the last subsection, the Cplex implementation of Simplex algorithm outperforms Interior Point method for solving the LP relaxation of this problem formulation, in the sense of generating better integer solution, better dual bound and using less root relaxation time in all the 3 instances.

Instance Name	Interior Point				
	Computation time (sec)	Best integer	Best node	gap	Root relaxation time (sec)
ben_10	21600	228.69	164.25	28%	1.45
ben_50	21600	-	307.75	-	6346
ben_76	24062	-	-	-	∞

Table 6.22: The MILP solver (Cplex) results with Interior Point method to solve the subsidiary LP relaxation.

As a consequence of the results above, we will stick to Simplex as our LP relaxation solving strategy in the sequel.

In the sequel we applied the Cplex on the MILP model again, and with the heuristic solution as a MILP starting value provided to Cplex. The MILP starting value is provided in the hope of helping the branch and bound procedure to cut off more unnecessary subbranches when the bounding value of the subbranch exceed the global

upper bound.

As can be observed from the diagram below, in total 5 instances except for only “ben_20” have their dual lower bound improved, comparing with Cplex from scratch without MILP starting value. Concerning the upper bound, the tiny instance “ben_3” found its optimal solution by the heuristic, “ben_10” has improved about 10% comparing to its heuristic value, however, the rest of the instances cannot gain improvement to their original heuristic solution.

Instance Name	Primal heuristic		Dual (Cplex)						Final Solution	
	Cost	Root gap	Comp. time (sec)	Best integer	Best node	gap	#visited node	#node left	vehicles	∑
ben_3	85.3	0%	1.99	85.3	85.3	0%	33	1	1/1/0/0/0	2
ben_10	239.6	27%	21600	221.2	174.8	21%	307001	196330	1/0/1/1/0	3
ben_20	366.5	50%	21600	366.5	182.0	50%	7211	6311	5/0/1/0/0	6
ben_30	573.3	59%	21600	573.3	232.3	59%	159	159	5/1/3/0/0	9
ben_40	704.0	60%	21600	704.0	278.5	60%	62	63	7/0/4/0/0	11
ben_50	814.5	62%	21600	814.5	308.3	62%	0	1	8/1/4/0/0	13

Table 6.23: The final results of the extracted instances combining MILP solver Cplex with the heuristic results as MILP starting values.

As can be seen from above, unfortunately for the instances larger than “ben_20”, the gaps are still vast, around 50% to 60%. More work should be done to improve the dual bound, including model reformulation and adding more problem specific valid inequalities. It is already on our future agenda.

6.6 Cyclic Locomotive Scheduling

In this project, as we have done in the last two projects, the solution found by the PGreedy type heuristic serves as a feasible initial solution for the commercial MILP solver ILOG Cplex. Experiment results show that the heuristic usually generates a good solution (in average around 10% from optimality) in several minutes, and in combination with the MILP solver it has also significantly speeded up the process of solving the medium instances to optimality.

The computational results are already submitted to a scientific journal. For the interested readers we refer to the Fuegenschuh, Yuan et. al. 2006 or Fuegenschuh, Homfeld et. al. 2006.

6.7 Multi-Depot Locomotive Scheduling

From a large amount of the raw data given by our customer, a medium size logistic company, we have extracted some weekly deployment plans as test instances. Experiments on these real-world instances show, in the fixed timetable case the Random PGreedy heuristic solves the problem within 5 minutes with a less than 5% gap comparing to the optimal solution that was computed by a commercial MILP solver (ILOG OPL Studio 4.2) after hours. In the case of time window, no optimal solution can be found, however our PGreedy heuristic solves the problem within 5 minutes with a solution at least as good as the one returned by the MILP solver after running 24 hours. Our experimental results also show a great saving potential comparing the current manual schedule. In the fixed timetable case, the optimal schedule can save over 20% operational cost, while the introduction of a reasonable time window can even bring around 40% cost saving comparing to the original schedule.

We refer the interested readers to Yuan 2007 for more details.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin (1993), Network Flows: Theory, Algorithms, and Applications. Prentice Hall, New Jersey.

[2] A. Schrijver (1986), Theory of Linear and Integer Programming. Wiley Interscience, John Wiley & Sons Inc.

[3] G. Nemhauser, L. Wolsey (1999), Integer and Combinatorial Optimization. Wiley Interscience, John Wiley & Sons Inc.

[4] A. Löbel (1997), Optimal Vehicle Scheduling in Public Transit. PhD Thesis. Shaker Verlag. Online available at: ftp://ftp.zib.de/pub/zib-publications/books/Loebel.disser.ps

[5] G. B. Dantzig and R.H. Ramser (1959). "The Truck Dispatching Problem". Management Science 6, 80–91. 1959

[6] P. Toth, D. Vigo (2002), The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications, SIAM.

[7] C. E. Miller, A. W. Tucker and R. A. Zemlin (1960), Integer Programming Formulation of Traveling Salesman Problems. Journal of the ACM, 7(4):326 - 329, 1960.

[8] M. Desrochers, G. Laporte (1991), Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. Operations Research Letters, Vol. 10 (1991), pp. 27-36.

[9] P. Toth, D. Vigo (2002), Branch-and-Bound Algorithms for the Capacitated VRP. In P. Toth, D. Vigo (eds.), The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications, SIAM, 29 – 52.

[10] D.Naddef, G. Rinaldi (2002), Branch-and-Cut Algorithms for the Capacitated VRP. In P. Toth, D. Vigo (eds.), The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications, SIAM, 53 – 84.

[11] J. Bramel, D. Simchi-Levi (2002), Set-Covering-Based Algorithms for the Capacitated VRP. In P. Toth, D. Vigo (eds.), The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications, SIAM, 85 – 108.

[12] G. Laporte, F.Semet (2002), Classical Heuristics for the Capacitated VRP. In P.

Toth, D. Vigo (eds.), The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications, SIAM, 109 – 128.

[13] M. Gendreau, G. Laporte, J.-Y. Potvin (2002), Metaheuristics for the Capacitated VRP. In P. Toth, D. Vigo (eds.), The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications, SIAM, 129 – 154.

[14] P. Eveborn, P.Flisberg and M. Rönnqvist (2005), Laps Care - an operational system for staff planning of home care. European Journal of Operational Research, Volume 171, Issue 3, 16 June 2006, Pages 962-976.

[15] E. Cheng, and J. Rich (1998), A Home Health Care Routing and Scheduling Problem, Technical Report TR98-04, Department of CAAM, Rice University., 1998.

[16] S. Begur, D. Miller, J. Weaver (1997), An integrated Spatial DSS for scheduling and routing home-health-care nurses. Interfaces 1997;27(4):35-48.

[17] T. Fahle (2001), Production and Transportation Planning Modeling Report, online available under http://www.brics.dk/ALCOM-FT/Main/D9b.ps

[18] J. Braca, J. Bramel, B. Posner and D. Simchi-Levi (1997), A Computerized Approach to the New York City School Bus Routing Problem. IIE Transactions 29, 693 -- 702.

[19] H. Hoos, T. Stützle (2004), Stochastic Local Search - Foundations and Applications. Morgen Kaufman.

[20] J.P. Hart and A.W. Shogan (1987), Semi-greedy Heuristics: An Empirical Study. Operations Research Letters, 6:107–114, 1987.

[21] T. A. Feo and M. G. C. Resende (1989), A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters, 8:67–71, 1989.

[22] T.A. Feo and M.G. C. Resende (1995), Greedy Randomized Adaptive Search Procedures. J. of Global Optimization, 6:109–133, 1995.

[23] A. Fügenschuh (2005), The Integrated Optimization of School Starting Times and Public Transport. PhD Thesis. Logos Verlag Berlin.

[24] A. Fügenschuh (2005), Parametrized Greedy Heuristics in Theory and Practice. Volume 3636/2005 of Lecture Notes in Computer Science, pages 21-31, Springer Berlin.

[25] A. Fügenschuh (2005hm), Parametrized Greedy Heuristics in Theory and Practice. Proceedings of Hybrid Metaheuristics, page 21-31, Second International Workshop, HM 2005, Barcelona, Spain, August 29-30, 2005.

[26] A. Fügenschuh (2006). The vehicle routing problem with coupled time windows. Central European Journal of Operations Research, 14(2):157 - 176, 2006.

[27] A. Fügenschuh and B. Höfler (2006). Parametrized grasp heuristics for threeindex assignment. Lecture Notes in Computer Science, Vol. 3906: "Evolutionary Computation in Combinatorial Optimization: 6th European Conference, EvoCOP 2006, Budapest, Hungary, April 10-12, 2006", pages 61 - 72, 2006.

[28] A. Fügenschuh, H. Homfeld, A. Huck, and A. Martin (2006). Locomotive and wagon scheduling in freight transport. Proceedings of the ATMOS06, 2006.

[29] A. Fügenschuh, H. Homfeld, A. Huck, A. Martin, Z. Yuan (2006): Locomotive and Wagon Scheduling in Freight Transport. Submitted to Transportation Science.

[30] Z. Yuan, A. Fügenschuh (2007): Parameterized Random Greedy Algorithms for Heterogeneous VRP with Time Windows. Proceedings of Doctoral Symposium on Engineering Stochastic Local Search Algorithms, Brussels, Sept. 7, 2007, page 52 – 57. .

[31] Z. Yuan (2007). Multi-Depot Locomotive Scheduling Problem with Coupling Trips and Time Windows. Bachelor thesis in Darmstadt University of Technology, also available on request: ericyuanzhi@gmail.com, 2007.

[32] Z. Zabinsky, R. Smith, J. McDonald, H. Romeijn, and D. Kaufman (1993). Improving hit-and-run for global optimization. Journal of Global Optimization, 3:171 – 192, 1993.

[33] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp (2002), A Racing Algorithm For Configuring Metaheuristics. In W. B. Langdon, E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 11--18, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2002.

[34] G.A.P. Kindervater, M.W.P. Savelsbergh (1997), Vehicle Routing: Handling Edge Exchanges. E.H.L. Aarts, J.K. Lenstra (eds.). Local Search in

Combinatorial Optimization, Wiley, Chichester, 337-360.

[35] R. Ruiz and T. Stützle (2005), A Simple and Effective Iterated Greedy Algorithm for the Flowshop Scheduling Problem. European Journal of Operational Research, 177(3):2033-2049, 2007.

[36] M. Dorigo and T. Stützle (2004), Ant Colony Optimization, MIT Press, Cambridge, MA, USA.

[37] R. Martin (2002), A Short Introduction to OPL - Optimization Programming Language. Online available under: http://zuseex.algo.informatik.tu-darmstadt.de/lehre/2002ws/algomod/AlgMod_OPL.pdf

[38] Wikipedia. Online available at URL http://www.wikipedia.org

[39] VRP Web. Online available at URL http://neo.lcc.uma.es/radi-aeb/WebVRP/

[40] ILOG, Inc. ILOG CPLEX (2006): High-performance software for mathematical programming and optimization, 2006. See http://www.ilog.com/products/cplex/.