Optimization Models and Algorithms for Large-Scale, Capacity Constrained Pick-up and Delivery Problems with Time Windows

by

Raphaël Tardy

Ingénieur de l'Ecole Polytechnique (2003)

Submitted to the Department of Civil and Environmental Engineering
and Operations Research Center
in partial fulfillment of the requirements for the degree of

Master of Science in Transportation
and
Master of Science in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005

Signature of Author .....
Department of Civil and Environmental Engineering
and Operations Research Center
May 16, 2005

Certified by .....
Cynthia Barnhart
Professor of Civil and Environmental Engineering, and Engineering Systems
Thesis Supervisor

Accepted by .....
James B. Orlin
Edward Pennell Brooks Professor of Operations Research
Co-director, Operations Research Center

Accepted by .....
Andrew Whittle
Chairman, Departmental Committee on Graduate Studies

Optimization Models and Algorithms for Large-Scale, Capacity Constrained Pick-up and Delivery Problems with Time Windows

by

Raphaël Tardy

Submitted to the Department of Civil and Environmental Engineering
and Operations Research Center
on May 16, 2005, in partial fulfillment of the
requirements for the degree of
Master of Science in Transportation
and
Master of Science in Operations Research

Abstract

Major package delivery companies employ hundreds of thousands of people, generate billions of dollars in revenues and operate very large fleets of ground vehicles ranging from custom-built package cars to large tractors and trailers. A crucial point for the profitability of these companies is, for a given level of service, to be able to run their operations at the lowest possible cost. In this thesis, we will contemplate the problem of the scheduling and routing on a regional and daily basis of the large tractor and trailer fleet of a large package delivery company. Our aim is to design a method for building the schedules associated with minimal operating costs. We consider deterministic situations in which all parameters are known exactly and we exclude possibilities of disruptions. Nonetheless even with these simplifications, the problem we consider is complex and large-scale, containing a very large number of constraints and parameters. Throughout this thesis, we examine different theoretical approaches including optimization models and algorithms. We implement some of these approaches in order to get practical results which can be implemented in practice.

Thesis Supervisor: Cynthia Barnhart
Title: Professor of Civil and Environmental Engineering, and Engineering Systems

This page intentionally left blank

Aknowledgments

I would like to express my gratitude to my advisor, Professor Cynthia Barnhart, for her support and the guidance she provided me with in my research.

This page intentionally left blank

Contents

List of Figures

This page intentionally left blank

List of Tables

This page intentionally left blank

Chapter 1

Introduction

1.1 Statement of the problem

1.1.1 Objective

The objective of this thesis is to develop a schedule satisfying a package delivery company's requirements for their ground operations. The problem we have to solve can, at first, be roughly formulated as follows: given a set of loads or trailers characterized by their origins and destinations and by their earliest available times and latest arrival times, we are to find optimal routes and schedules for both the drivers and the tractors.

1.1.2 Main parameters and constraints

In this section, we will develop successively the main parameters and constraints involved in the problem. To clarify our exposition, some of the requirement and specificities will be rediscussed in Section 1.2 where we will review the data provided.

First, we are given a network. This network is a directed graph $\mathcal{G}=(\mathcal{N},\mathcal{A})$ with $\mathcal{N}$ the set of nodes and $\mathcal{A}$ the set of arcs. The set of nodes correspond to the locations used by the company. These locations are of different types. Some of these locations correspond to hubs, others to sorting facilities, railyards or customers.The arcs link two different locations and are defined both in distance and in time. Arcs span the amount of time required to travel between two given locations from the tail node to the head node of the arcs. This travel time can vary

depending on the route taken between the locations.

Second, as previously stated, the main requirement is to route correctly the different loads. Here, a load corresponds to a trailer full of packages. Each load is associated with a time window. A time window is defined by both an earliest available time and a latest delivery time. The two times are determined beforehand in order to be consistent with: 1) the schedules of the carrier sorting facilities; 2) other time windows; for example if a load is carried over more than one region, the time windows affecting the load will be defined to satisfy any time restrictions of theregions involved.

Third, of tremendous importance is the fact that there exists two different types of trailers. There are short and long trailers. This distinction is important for different *Tractor/Trailer* configuration are allowed. In fact, a tractor can haul either one short trailer, one long trailer or two short trailers. When the tractor hauls only one load, the*Tractor/Trailer* configuration is denoted as *Single*; if the tractor hauls two trailers, its configuration is denoted as *Double.* In order to keep things simple, from now on, we will not dissociate the trailer from the load. Thus, we will apply the adjective short or long directly to the loads.

Fourth, it is allowed that two drivers can exchange loads. This means that a driver can drop a load at a location from which it is later picked up by another driver.

Fifth, for a driver's route to be valid, it has to satisfy various work rules. First of all, a driver has to begin and to finish his work day at the same location called a domicile. Only a small subset of locations are used as domiciles. Next, work rules also implies that the working time cannot exceed a legal limit. This working time limit depends on the domicile to which the driver is associated. Furthermore, within a work day, a driver must have up to two breaks and one meal break. The length of the breaks and of the meal, and the time at which they occur, also depend on the domicile associated with the driver.

Finally, other secondary constraints exist. In particular, when a truck arrives or exits a location, "turn" time is needed for unloading and loading the tractor and for turning around the location among tother things. Another issue is that it is forbidden to drive a tractor trailer on some arcs because of local regulations. We will discuss these secondary constraints in further detail in a later section.

1.2 Details of the data provided

We introduce in this section the different types of information provided by the data presented in the form of Excel worksheets. In this review will be the occasion to somewhat we quantify the number of arcs, locations, loads and different work rules of the problem to indicate the complexity of the problem. We also introduce some of the notations used later as we develop our optimization model. Finally, we present selected constraints..

1.2.1 The network

The network we consider consists of 98 nodes and 496 arcs. This network covers a region corresponding to Virginia, Washington D.C, Maryland, Pennsylvania and New York.

The arcs

The 496 arcs, defined both in distance and in time, are represented as:

arcID	ptA	ptB	travelTime	miles
1	58	28	97	54
⋮	⋮	⋮	⋮	⋮
496	61	55	93	28

Table 1.1: Arcs

where the fields represent:

- *ptA*: origin of the arc;
- *ptB*: destination of the arc;
- *travelTime*: time to travel from origin to destination of the arc; and
- *miles*: number of miles from origin to destination of the arc;

In fact, we should have considered two different networks for on some arcs it is forbidden to have a tractor without trailer travelling on. However, in a first approach, we will not take into account this specificity and we will consider only one network that trucks in any configuration can use.

The locations

The following table depicts how locations are specified:

locID	toIn	toOut	sIn	sOut	dIn	dOut	doWash	washTime
1	11	6	15	14	22	36	Y	7
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
98	8	0	12	9	18	30	N	N/A

Table 1.2: Locations

where the fields represent:

- *locID*: integer number ranging from 1 to 98 corresponding to the ID of the location;
- *toIn*: time to enter the location if we have a tractor without a trailer;
- *toOut*: time to depart of the location if we have a tractor without a trailer;
- *sIn*: time to enter the location if we have a tractor with one trailer;
- *sOut*: time to depart of the location if we have a tractor with one trailer;
- *dIn*: time to enter the location if we have a tractor with two trailers;
- *dOut*: time to depart of the location if we have a tractor with two trailers;
- *doWash*: equals Y if we have to wash the tractor at the location; N otherwise; and
- *washTime*: duration of the washing;

As mentionned before, we have to consider some slack times when a truck enters and exits a location. These slack times which are represented by the fields *toIn, toOut, sIn, sOut, dIn* and *dOut* reflect the time needed for loading and unloading the trailer and the time for driving around the location. At first, we conjectured that the *in* and *out* times are not significant and cannot be ignored. Consider, however, a driver who makes 6 stops during the day. The sum of these times, can exceed 2 hours, roughly 20% of the maximum authorized work time, which is all but negligible.

In some locations, the truck will have to be washed before it may leave the location. We distinguish the locations at which we have to perform that operation by a "Y" in the *doWash* field and, in that case, the wash time is given by the field *washTime*.

1.2.2 The loads

In this section, we examine the manner in which requirements for the loads are presented to us.

The different types of trailers

The trailers are defined as follows:

trlrTypeID	doubleable
1	Y
⋮	⋮
52	N

Table 1.3: Loads

where the fields represent:

- *trlrTypeID*: ID of the trailer;
- *doubleable*: equals Y if the trailer can be doubled; N otherwise.

The type of trailer is specified for each load. Therefore, we can incorporate the field *doubleable* directly into the specification of the load. In what follows, we will no longer consider the trailers as independant of the loads.

The requirement for the loads

The table below presents the inputs for the loads.

loadID	ptA	ptB	eavl	larr	isOverRideTime	overrideTime	trlrTypeID
1	52	13	2048	217	N	0	37
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
364	3	27	244	722	N	0	38

Table 1.4: Requirements

where the fields represent:

- *loadID*: ID of the trailer;
- *ptA*: ID of the origin location;
- *ptB:* ID of the destination location;
- *eavl*: earliest available time for picking up the load;
- *larr*: latest arrival time at destination for the load;
- *isOverride*: indicates whether there is some override time as a requirement for the load;
- *overrideTime*: length of the override; and
- *trlrTypeID*: ID of the trailer associated with the load;

The loads are defined by an origin (ptA) and a destination (ptB). Each load is associated with a trailer designated by its trailerID. A load cannot be picked up before it is ready for departure, denoted its earliest available time *eavl.* Similarly, a load has to arrive at its destination before a certain hour, designated by the field *larr.*

Overrides correspond to movements which have to be made at a location, for example, moving trailers within a sorting facility.

Finally, we note that the time frame in which we will develop the schedule correspond to a 2 days period. Hence, the earliest time considered is 0 and the latest time considered is 4800. An hour corresponds to 100.

1.2.3 Work Rules

As previously stated, only a small number of locations within the network are domiciles. Information pertaining to each domicile, for which, different work rules apply, is provided in the following format:.

domID	locID	minDr	maxDr	maxDay	minDay	sw	fw	unpaidBr	paidBr
1	24	0	2	1100	800	17	9	100	17
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
6	28	0	100	1150	800	17	9	100	17

Table 1.5: Work rules

where the fields represent:

- *domID*: ID of the domicile;
- *locID*: ID of the location corresponding to the domicile;
- *minDrivers*: minimum number of drivers located at the domicile;
- *maxDrivers*: maximum number of drivers located at the domicile;
- *maxDay*: maximum number of hours which can be worked in a day;
- *minDay*: mimimum guaranteed number of hours paid;
- *sw*: time needed for starting the work;
- *fw*: time needed for finishing the work;
- *unpaidBr*: amount of non-paid time for the breaks; and
- *paidBr*: amount of paid time for the breaks.

We see that there are only 6 domiciles out of 96 locations and that work rules are not homogenous within domiciles. There is a specified guaranteed paid day denoted *minDay* and

a maximum legal work day denoted *maxDay*. Any work beyond *minDay* will be considered as over time work, and there is a maximum of *maxDay-minDay* of possible overtime.

Slack time *sw* represents the elapsed time between the arrival of the driver at work and the time the driver actually start working. The second slack time denoted *fw* equals the elapsed time between the moment the driver finishes work and the time at which the driver leaves work. These slack times have to be included in the total work time.

Finally, there is at least 1 hour of non-paid break, denoted by *unpaidBr*, and 0.17 hour of paid break, denoted by *paidBr*. We discuss more of the issues related to the breaks in the next subsection.

1.2.4 Breaks

Two different break combinations, represented as follows, can be applied:

comboID	B1Early	B1Late	B1Dura	MEarly	MLate	MDura	B2Early	B2Late	B2Dura
1	200	400	42	400	600	50	500	1100	25
2	150	350	25	350	550	67	500	1100	25

Table 1.6: Breaks

where the diffenrent entries represent:

- *B1Early*: minimum elapsed time of the route at which the first break can occur;
- *B1Late:* maximum elapsed time of the route at which the first break can occur;
- *B1Dura:* duration of the first break;
- *MEarly:* minimum elapsed time of the route at which the meal break can occur;
- *MEarly*: maximum elapsed time of the route at which the meal break can occur;
- *MDura*: duration of the meal break;
- *B2Early*: minimum elapsed time of the route at which the second break can occur;

- *B2Late:* maximum elapsed time of the route at which the second break can occur; and
- *B2Dura*: duration of the second break;

Each break combination contains three elements, namely: first break, meal and second break. Each break has a specific duration denoted respectively by *B1Dura, MDura* and *B2Dura,* and has to occur between a precise time window delimited by an earliest time *B1Early, MEarly* and *B2Early,* respectively and latest time denoted respectively *B1Late, MLate* and *B2Late.*

1.2.5 Cost Function

Cost is computed as a function of the following different parameters:

- *perMileTO:* cost per mile in *Tractor Only* configuration;
- *perMileSingle:* cost per mile in *Single* configuration;
- *perMileDouble:* cost per mile in *Double* configuration;
- *perHourTO:* cost per hour of regular (non-overtime) hours in *Tractor Only* configuration;
- *perHourSingle:* cost per hour of regular (non-overtime) hours in *Single* configuration;
- *perHourDouble:* cost per hour of regular (non-overtime) hours in *Double* configuration;
- *OTperHour:* cost per hour of overtime in *Tractor Only* configuration;
- *OTperHourSingle:* cost per hour of overtime in *Single* configuration; and
- *OTperHourDouble:* cost per hour of overtime in *Double* configuration.

We have the following relationships between these different parameters:

1. $perMileTO < perMileSingle < perMileDouble$, $perMileDouble < 2 \times perMileSingle$
2. $perHourTO < perHourSingle < perHourDouble$, $perHourDouble < 2 \times perHourDouble$
3. $OTperHour < OTperHourSingle < OTperHourDouble$, $OTperHourDouble < 2 \times OTperHourSingle$, $OTperHour > perHourTO$, $OTperHourSingle > perHourSingle$, $OTperHourDouble > perHourDouble$

These relationships show that there are economies to hauling several trailers together. Morever, the hourly rate at which a driver is paid is the same throughout the day, even if the driver hauls different configurations. Specifically, the hourly rate is set to the most expensive rate the driver is to be paid if based on the configuration of his tractor throughout the day. For example, if a tractor is in *Double* configuration for only one hour an in Single configuration the rest of the day, the driver is paid all day at the *Double* rate.

1.3 Terminology: route, path, schedule

For clarity of the exposition, we use the following terminology:

- a **route** is associated with a tractor-driver pair; a driver and his tractor remain together thoughout the whole day;
- a **path** is used indifferently for a load or a trailer, a trailer and a load remain together; and
- a **schedule** consists of a combination of routes meeting all problem requirements (service of loads, work rules, domicile locations, etc...).

1.4 Literature survey and sources of complexity in the problem

In this section, we examine related problems. By comparing these problem to ours, we identify and highlight complexities of our problem.

1.4.1 Unknown paths and routes

Besides its size, what make this problem particularly complex is primarly the fact that we know neither what are the path of the loads nor the routes of the drivers. For many applications, either the paths or the routes are known. For example, in the well known problem crew scheduling problem, the paths of the planes are known a priori and what has to be determined are the routes of the crews (Barnhart, *et al* (2000)). If we fix the paths of the loads in our problem, the resulting problem is similar to the crew scheduling problem. A sequential approach first

fixing paths of loads and then routes of drivers is sub optimal however because routes of drivers and paths of loads are interdependant. A good solution for the loads could lead to be a bad solution for the drivers, and hence the two should be determined simultanously.

1.4.2 Multiple tractor-trailer configurations

Another very important source of complexity is the multiple tractor-trailer configurations. Different configurations introduce the following complexities:

- slack times differ according to the configurations when arriving and leaving a location; and
- hourly driver's rates depend on the higher rank configuration within a route.

If these multiple configurations did not exist, we could compute a priori for each load: 1) the earliest available time; 2) the latest arrival time; 3) a distance matrix of the travel time between two loads. The dimensions of the distance matrix correspond to the number of loads, and the entry in cell (i,j) represent the time needed to bring load i from its origin to its destination and then to the origin of another load j. If times did not vary by configuration, we could classify our as a *Multi Depot / Multiple Homogenous Vehicle / Pick-up and Delivery problem with Time Windows* (MD-MVH-PDP-TW). This would allow us to express our problem as a linear integer program with tour constraints, time windows constraints, route length constraints and visitation constraints.

There are numerous heuristics available for solving this type of problem for whcih *Travelling Salesman Problems* or *Vehicle Routing Problems* are special cases. Selected heuristics include tour improvement algorithms (Laporte (1992)), insertion heuristics (Solomon (1987)) or Tabu Search (Laporte (1992)).

Even without the added complexity of different configurations, the MD-MVH-PDP-TW formulations do not account for breaks. Other limitations of MD-MVH-PDP-TW are that MD-MVH-PDP-TW have to specify a priori the number of drivers and we have to use a proxy objective function corresponding to the total driving time (see Subsection 2.1.1). These two downsides can be overcome. First, we can account for breaks using some constraint generation. Second, we can compute various schedules with different numbers of drivers, then compute

the total costs with precision in order to decide which schedule to keep. However, multiple configurations prevent us from applying these kinds of approaches because we would have to consider all the possible combinations of two loads traveling together as entries in the distance matrix, making the problem too large.

1.4.3 Additional sources of complexity

The time windows associated with each load increase problem complexity in two ways. First, time windows are a key determinant of whether or not two loads can travel together in a *Double* configuration. It is clear that if two loads have incompatible time windows, it will not be possible to carry them together. Second, time windows also establish precedence constraints dictating the order in which certain loads must be picked-up or delivered.

Another source of uncertainty is provided by non linearities in the cost function resulting from minimum pay guaranteed for drivers and the manner in which driver hourly rates are determined. Finally, because arcs are associated with distance and time, problem size increases dramatically with increases in the number of loads, presenting tremendous tractability challenges.

1.5 Outline of the Thesis

In Chapter 2, we present two approaches to solve our problem and assess the benefits of each from a theoretical point of view. Then, in Chapter 3 and Chapter 4, we develop each of these approaches and implement them. In the appendices, we present the different program and methods used to implement the algorithms.

Chapter 2

Strategy for solving the problem

In this chapter, we describe assumptions and simplifications we make in modelling the problem and then describe two different approaches for tackling the problem.

2.1 Assumptions and Simplifications

The large size and complex nature of our pick-up and delivery problem necessitate certain simplifying assumptions to enable solution of the problem.

2.1.1 Proxy function for the cost

Total schedule cost corresponds to the sum of the costs of all the routes included in the schedule. While the actual route cost is a function of the mileage of the truck and of the work time of its driver, our analysis of the data indicates that total schedule cost might be reasonably approximated as a linear function of the number of its drivers. This is due to the fact that the main component of the cost comes from the wages of the drivers and that paid driver time is roughly the same for each driver. Paid time per driver is typically between 8 and 11 hours, and most drivers are paid at the double rate. Hence, we use as proxy function for schedule cost the number of routes included in the schedule. We use true costs, however, after generating a solution to compute the actual cost of the schedule.

2.1.2 Shortest path in time and in distance

As mentionned previously it might happen that for two points in the network, the shortest path in time might not coincide with the shortest path in distance. Here again our analysis of the data suggests that the time needed to travel along an arc is resonably approximated by a linear function of the length of the arc. Therefore, we assume that the shortest path in time corresponds to the shortest path in distance.

2.1.3 Exchange of loads

Our last simplification, and the one with the greatest consequences, is that we do not allow drivers to exchange loads. Our analysis shows that these restrictions do not affect the feasibility of the problem, but they do reduce the set of feasible solutions. Although this simplication might hamper the quality of the solution, it simplifies the structure of the problem by making routes independent one from one another and by linking paths and routes as demonstrated in the following sections. Route independence and linkage with paths enhance our ability to find a solution, at the cost of reduced solution quality.

In practice loads exchanges are not widely used because they are difficult to execute due to the high degree of coordination needed between drivers. If a driver fails an exchange load on time, another driver is delayed and a chain reaction of disruptions all accross the network might be triggered; forbidding load exchanges might lead to more robust operations.

2.2 The shortest path problem

This section deals with an issue that arises recurrently in solving the pick-up and delivery problem. We need to have a procedure for computing the minimum routing time or, analogously, computing the shortest path from the origin to the destination of a load in a time-based network. We present two methods for solving this problem. One based on a network flow formulation and another based on Dijkstra's algorithm.

2.2.1 Computing the shortest path with a network flow formulation

Computing the shortest path from any location to another can be accomplished with a network flow as follows:

$$\begin{aligned} &\text{minimize} && \sum_{j \in \mathcal{A}} x_j . T_{jj} \\ &\text{subject to,} && \forall i \in \mathcal{N}, \sum_{j \in \mathcal{A}} x_j \times \Delta_{ij} = b_i \end{aligned} \tag{2.2.1}$$

where $\mathcal{A}$ is the set of arcs in the network; $\mathcal{N}$ is the set of locations in the network; T_j is the travel time associated with arc $j, \forall j \in \mathcal{A}$; Δ_{ij} equals 1 if the destination of arc j is location i, equals -1 if the origin of arc j is location i and equals 0 otherwise; b_i equals -1 if location n corresponds to the origin of the load, equals 1 if it corresponds to its destination and equals 0 otherwise; and x_j is a decision variable equal to 1 if arc j is in the solution and equals to 0 otherwise, hence $x_j \in \{0, 1\}, \forall j \in \mathcal{A}$.

2.2.2 Computing the shortest path with Dijkstra's algorithm

Alternatively, we can compute the shortest path for a load using the Dijkstra algorithm. There many variant of Dijkstra's algorithm. We present here a formulation of the algorithm proposed by Larson and Odoni (1981)

The algorithm begins at a specified node s (the "source" node) and successively finds its closest, second closest, third closest, and so on, node, one at a time, until all nodes in the network have been found. This procedure is aided by the use of a two-entry label, $(d(j), p(j))$, for each node j. $l(i, j)$ is the length of arc (i, j) and it is supposed nonnegative;$d(\ j)$ is the length of the shortest path from s to j discovered so far; and $p(j)$ is the immediate predecessor node of j in the shortest path from s to j discovered so far.

In the evolution of the algorithm, each node can be in one of two states:

1. The open state in which the node's label is still tentative; or

2. The closed state in which the node's label is permanent.

We use the symbol k to indicate the most recent (i.e., the last) node to be closed and the dummy symbol "$*$" to indicate the predecessor of the source node s. The algorithm can now be described as follows:

1. Let $d(s) = 0; .p(s) = *$; $d(j) = \infty, p(j) = -\infty$ for all nodes $j \neq s$. Consider node s as closed and all other nodes as open; set $k = s$ (i.e., s is the last closed node).

2. Examine all edges (k, j) out of the last closed node k; if node j is closed, go to the next edge; if node j is open, set

$$d(j) = Min[d(j), d(k) + l(k, i)]$$

3. Choose the open node with the smallest $d(j)$ value as the next node to be closed. Let this is node be i.

4. Consider, one at a time, the edges (j, i) leading from a closed nodes j to i until one is found such that

$$d(i) - l(j, i) = d(j)$$

   Let this predecessor node be j^*. Then set $p(i) = j^*$.

5. Close node i. If all nodes in the graph are closed, then stop; the procedure is finished. If there are still open nodes in the graph, set $k = i$ and return to Step 2.

   Upon termination of the algorithm, $d(j)$ indicates the length of the shortest path from s to j, while $p(j)$ indicates the predecessor node of j on this shortest path. By tracing back the predicted path, it is easy to identify the shortest path between s and each of the nodes of $\mathcal{G}$ can be identified.

2.3 Generation of a schedule from a set of routes

With the exclusion of load exchanges among drivers, we can characterize a route by the loads it carries. Moreover, we can consider schedule to be the number of drivers needed. We recast the pick-up and delivery problem as selecting the minimum number of routes that includes each load at least once in the selected route set. We refer to this problem as the master problem, a set partionning problem that can be formulated as follows:

- **Master Problem**

$$\text{minimize} \quad \sum_{j=1}^{n} x_j$$

$$\text{subject to,} \quad \sum_{j=1}^{n} a_{ij}\, x_j \geq 1\,, \qquad \forall i \in \{1...m\} \qquad (2.3.1)$$

$$x_j = 0\,, \qquad \forall i \in \{1...m\}, \forall j \in \{1...n\} \qquad (2.3.2)$$

where n represents the number of routes generated; m the number of loads; a_{ij} equals 1 if load i is carried by route j and equals 0 otherwise; and x_j is a decision variable equal to 1 if load j is included in the solution and equal to 0 otherwise.

We note that we do not require a load to be included in exactly one route. We will show in Section 3.1 that the restriction of an existing route is also a route. This means that if there exists a route carrying several loads, from that route we can easily build other routes which will carry only of subset of the loads carried by the original route.

This problem is easy to solve and it can be fed into CPLEX directly. We do not need to develop any special heuristic for CPLEX will found the optimal solution very quickly. Moreover, as we will explain later, we will also consider the linear relaxation of this problem which is even easier to solve.

The issue of how to build routes is examined in the next sections.

2.4 Two options for building routes

With the exclusion of load exchanges, the path of a load will always be included in a driver's route. Exploiting this fact, we devise two different approaches for generating routes.

2.4.1 Column generation

In this section, we will consider a column generation approach to generate routes. To describe it, we will first present the cutting stock problem and a solution approach for it, and show how it is related to our approach.

The Cutting Stock Problem

The cutting stock problem is to find the best way to cut rolls of paper, metal, etc... into rolls of smaller widths. Each different selection of the widths of the smaller rolls is referred to as a pattern. All possible widths that can be cut from roll j can be represented by a vector $A_j = (a_{1j}, ..., a_{mj})$ with a_{ij} corresponding to width w_i and the number of possible widths bounded by m. Each pattern cut from roll j, of width W must then satisfy the following constraints:

$$\sum_{i=1}^{m} a_{ij} w_i \leq W$$

$$a_{ij} \geq 0,\ a_{ij} \text{ integer}, \forall i \in \{1...m\}$$

Denote n as the number of feasible patterns that can be cut from a large roll and b_i as the number of rolls of width w_i needed. Now let a_{ij} equal 1 if pattern j contains a roll of width i; and equal 0 otherwise. The goal is to minimize the number of large rolls while satisfying the demand requirement for smaller rolls of varying widths. This can be captured by the following integer program referred to as the master problem in which each decision variable x_j equal to the numbers of times we cut a roll into pattern j.

- **Master problem**

$$\text{minimize} \quad \sum_{j=1}^{n} x_j$$

$$\text{subject to,} \quad \sum_{j=1}^{n} a_{ij}\ x_j = b_i\ ,\ \ \forall i \in \{1...m\} \qquad (2.4.1)$$

$$x_j \geq 0,\ x_j \text{ integer},\ \ \forall i \in \{1...m\}, \forall j \in \{1...n\} \qquad (2.4.2)$$

To find an intial basis, we simply select patterns b_i large rolls each into one smaller roll of width w_i for each $i \in \{1...m\}$. This provides a solution using a total of $\sum_{i \in \{1...m\}} b_i$ large rolls.

Because the number of patterns is typically too large to enumerate, only a subset of patterns is initially included in the master problem. Additional patterns that can improve the solution are iteratively added to the master problem using an approach called column generation. To

identify columns (or variables) to add to the master problem and improve the solution, we solve the relaxation of the master problem to obtain the dual solution **p**. It has been shown (Bertsimas, Tsitsiklis (1999)) that the best pattern we can add to the master problem at this point is the one which maximizes $\mathbf{p}'Aj$. Hence, to identify a pattern that should be added to the master problem to improve the solution, we solve the following subproblem, referred to as the pricing problem.

- **Pricing problem**

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{m} p_i \, a_i \\ \text{subject to,} \quad & \sum_{i=1}^{m} w_i \, a_i \leq W \ , \forall i \in \{1...m\} && (2.4.3) \\ & a_i \geq 0 \ , \ a_i \text{ integer}, \forall i \in \{1...m\} && (2.4.4) \end{aligned}$$

with p_i the optimal dual variable value corresponding to contraints i in the current master problem; and a_i equal to the number of times a roll of width w_i is included in the pattern pricing problem.

If the maximum of this pricing problem is less or equal to 1, the current solution is optimal. However, if the maximum is greater than 1, the new column $A = (a_1, ..., a_m)$ is added to the the master problem. Then resolve the master problem and repeat the process until the objective value of the pricing problem equals 1 or greater, that is until the optimal solution is reached.

Relation to route generation

In its simplest espression, a route can be described as the set of loads it carries. Therefore, we can draw an analogy between assembling loads into routes in our problem and cutting smaller width rolls from a larger width roll in the cutting stock problem. Just as there is a pricing problem for the cutting stock problem to find patterns; we will solve a pricing problem to assemble loads into a route. An advantage of this method is that we are able to use feedback between the master and the pricing problems in order to improve our solution from an iteration to the next To extend our analogy, the objective of our pricing problem will be:

$$\text{maximize} \quad \sum_{i=1}^{m} p_i \, a_i$$

where p_i is again the dual value associated with constraint i, which will define a route feasability condition as a function of the loads it contains; a_i equals 1 if load i is transported in the route and equals 0 otherwise. The complete mathematical formulations of the pricing problem to generate route are presented in Chapter 3.

2.4.2 Approach using clusters of loads

A sequential approach to solve the pick-up and delivery problem is to select paths for loads and construct vehicle route covering these paths. In constructing clusters, that is to say assemblings of loads, it is imperative to capture the fact that more than one load can be transported at the same time. Given the load clusters, our next step is to route vehicles over paths without regard to the number of loads carried. This method will be examined in Chapter 4.

Chapter 3

Building routes through column generation

In this chapter we first introduce properties and models pertaining to routes, and we present the formulation of the pricing problem to generate routes.

3.1 Definition and properties of the routes

3.1.1 Class of equivalent routes

We use time windows defined by earliest departure times and latest arrival times. In many cases, there is a high chance that a route and the same route shifted by one minute are both equivalent in terms of cost and of loads carried. Therefore, we can consider that these two routes belong to the same class of equivalent routes. The objective, then, of our computation is not to come up with an optimal route but rather with a class of optimal routes. Moreover, when we select routes for inclusion in a schedule, we are indifferent among all instances of a class of routes. Hence, we only need one instance from a class. But for some practical reasons some instances are more desirable than others and we later describe how we generate a desirable instance.

3.1.2 Definition of a route (or of a class of routes)

A route is defined as a succession of legs in which the first leg starts a domicile and the last leg ends at that same domicile. A leg accounts for everything happening between two stops and therefore it captures all the information concerning the movement of the driver and of the loads transported. Each leg is thus characterized by an origin and a destination, an earliest departure time and a latest arrival time, and the loads which are carried.

The number of the legs is decided beforehand and is denoted by $nVar$. Moreover, the legs are chronologically ordered and numbered from 1 to $nVar$. Because a tractor can haul up to two loads, we will define two trailer positions for each leg: *trailer position 1* and *trailer postion 2*. We define some rules specifying to which trailer position a load should be assigned; naturally, on a given leg a tractor does not haul any load, both trailer positions remain empty.

1. If the tractor hauls one short load, the load may be assigned to any trailer position;
2. If the tractor hauls two short loads, then both trailer positions are occupied; and
3. If the tractor hauls a long load, it is assigned by default to *trailer position 1* and *trailer position 2* is blocked and unable to accomodate another load.

Moreover, if a load is transported on more than one leg, the load has to remain in the same trailer position throughout its journey.

If a break occurs at a stop, between two legs, it could in fact be assigned to either the leg prior to the stop or to the leg after the stop. Here, in such a case, we assign the break to the later leg. We note that more than one break can be assigned to the same leg.Finally, we require that the routes start and finish at the same domicile. Figure 3-1 illustrates the composition of a route. For each leg, there aretwo boxes repesenting the two different trailer positions.

3.1.3 Properties of the routes

We can infer some properties related to our definition of a route.

1. **A route is consistent in time,** that is, the departure of a leg preceedes its arrival,. and the arrival of leg i should preceedes the departure of leg $i + 1$;

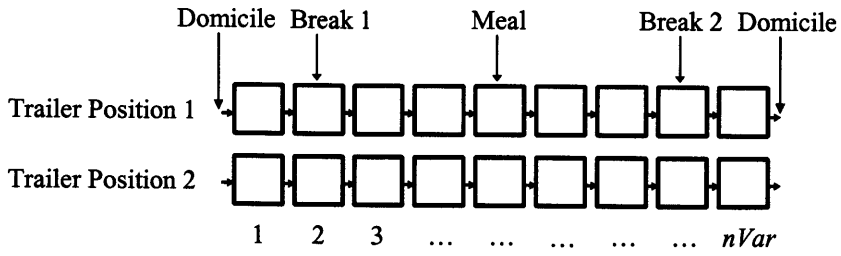


Figure 3-1: Composition of a route

2. **A route has to be consistent in space,** that is, the arrival location of leg i is identical to the departure location of leg $i+1$;

3. **Whatever the number of legs nVar, we can always find a feasible route** by including legs which remain at the domicile;

4. **Removing one or more load to a route creates a feasible route**, because travel time does not depend on the configuration of the truck and the times for entering and exiting a location decrease as configurations change from *Double* to *Single* and from *Single* to *Tractor Only*. Hence, by reducing the number of the loads along a route, the route remains valid, at least as long as we assume that *Tractor Only* configurations are allowed on all arcs. This point is particularly important as mentioned in Section 2.3 because it allows us to express constraints *(2.3.2)* as inequalities rather than equalities.

5. **The concept of lead load limits the number of legs considered in generating optimal routes** because loads must be picked up and delivered by the same driver, we can distinguish three possibilities for each leg in a route:

   (a) If there is no trailer position available (two short loads or one long load are currently being transported), the destination of a given leg must correspond to the destination of one of the loads transported on that leg . We define the leading load to be the load being carried and its destination will determine the destination of the leg;

   (b) If both trailer positions are available (truck in *Tractor Only* configuration) then the driver must pick up at least one load at the next leg. Thus, the destination of the

current leg is the origin of the one or two loads to be picked at the next leg. Note, we allow exception to this rule for the case in which a driver stays at his domicile during several legs at the end of his route; and

(c) If there is exactly one trailer position available (truck in *Single* configuration), the leading load is either the load currently being carried and, in that case, its destination is that of the leg, or the leading load can be a load to be picked up at the next leg and, in that case, its origin determines the destination of the leg.

It is easy to see why we can restrict ourselves to these different cases, with the rationale following directly from the fact that any other logic would result in non-productive legs requiring time and not increasing the number of loads served.

3.2 Definition of the sets

In this section, we present the notation we will use for the different sets of variables:

- *Loads*: set of loads;
- *Locations*: set of locations;
- *Domiciles*: set of domiciles;
- *Arcs*: set of arcs;
- *Legs*: set of legs in a route; $Legs = \{1..nVar\}$;
- $\overline{Legs}$: set of legs in route excluding the last leg; $\overline{Legs} = \{1..(nVar - 1)\}$; and
- $\underline{Legs}$: set of legs in route excluding the first leg; $\underline{Legs} = \{2..nVar\}$;

3.3 Objective function of the model

In Subsection 2.4.1 we have postulated an objective for the pricing problem to generate routs for the pick-up and delivery problem. Formalizing that, let $DUAL_j$ be the master problem's

dual values for load j and let $xload_j^k$ be the binary variable equal to 1 if load j is tranported at some point in the route in trailer position k and equal to 0 otherwise. Then the objective function can be written as:

$$\text{maximize} \quad \sum_{j \in Loads} ((xload_j^1 + xload_j^2) \times DUAL_j$$

To have values for dual variables $DUAL_j$, we need a basic feasible solution to the master problem. To achieve this, we consider an initial basis containing one route for each load.

3.4 Constraints of the pricing problem

Similarly to the Cutting Stock pricing Problem, we must insure that routes do not violate any feasibiltiy.Because, the number of constraints involved is large. As a consequence, we define variables and the necessary notations when needed and we aggregate constraints in different sets depending on their functions. The different sets are:

1. **Shortest path;**
2. **Domicile;**
3. **Origin and destination of a leg;**
4. **Origin and destionation of the leading load;**
5. **Choice of the leading load;**
6. **Pick-up and delivery;**
7. **Time in and out;**
8. **Work time;**
9. **Extra time;**
10. **Latest arrival and earliest available;**
11. **Break; and**

12. **General time.**

In the following sections, we define these different sets of constraints in detail.

3.4.1 Shortest path

We introduce a set of constraints to compute the travel time on a given leg:

- $timeLeg_i \geq \sum_{k \in Arcs} arc_{i,k} \times TRAVELTIME_k, \forall i \in Legs,$ *(3.1.1)*
- $\sum_{k \in Arcs} arc_{i,k} \times ARCMATRIX_{n,k} = origin_{i,n} + dest_{i,n}, \forall i \in Legs, \forall n \in Locations$ *(3.1.2)*

where $TRAVELTIME_k$ is defined as the travel time along arc k; $ARCMATRIX_{n,k}$ equals -1 if location n is the origin of arc k, and equals 1 if location n if the destination of arc k, and equals 0 otherwise; $origin_{i,n}$ is a decision variable equal to -1 if the origin of leg i is location n ans equal to 0 otherwise; $dest_{i,n}$ is a decision variable equal to 1 if the destination of leg i is location n and equal to 0 otherwise; $arc_{i,k}$ is a decision variable equal to 1 if arc k is present in the path of leg i and equal to 0 otherwise; and $timeLeg_i$ is a decision variable equal to the total travel time at leg i.

Constraints *(3.1.1)* ensure that the travel time on leg i is greater or equal to the sum of the travel times along each arc composing the leg. Next, constraints *(3.1.2)* are similar to the set of constraints of a network flow problem; they state that:

1. the path of leg i starts among all locations n, $n \in Locations$, from the location designated by $origin_{i,n}$ and ends at the location designated by $dest_{i,n}$ where n ; and
2. the path of the leg is continous. If an arc is not the first or the last arc in the path, its destination has to match the origin of the succeeding arc and its destination has to match the origin of the preceding arc. If the arc is the first arc of the path, its origin has to match the origin of the leg, and if the arc is the last arc in the path, its destination has to match the destination of the leg.

3.4.2 Domicile Constraints

As we have stated, a route has to start and end at a domicile. We present in this section the set of constraints which will enfore that rule.

- $\sum_{i \in Domiciles} dom_i = 1$ *(3.2.1)*
- $origin_{1,n} = -\sum_{i \in Domiciles} dom_i \times DOMICILE_{n,i}, \forall n \in Locations$ *(3.2.2)*
- $dest_{nVar,n} = \sum_{i \in Domiciles} dom_i \times DOMICILE_{n,i}, \forall n \in Locations$ *(3.2.3)*
- $\sum_{j \in Loads} x^1_{nVar,j} \times LOADIN_{n,j} \leq dest_{nVar,n}, \forall n \in Locations$ *(3.2.4)*
- $\sum_{j \in Loads} x^2_{nVar,j} \times LOADIN_{n,j} \leq dest_{nVar,n}, \forall n \in Locations$ *(3.2.5)*
- $\sum_{j \in Loads} x^1_{1,j} \times LOADOUT_{n,j} \geq origin_{1,n}, \forall n \in Locations$ *(3.2.6)*
- $\sum_{j \in Loads} x^2_{1,j} \times LOADOUT_{n,j} \geq origin_{1,n}, \forall n \in Locations$ *(3.2.7)*

where $DOMICILE_{n,i}$ equals 1 if domicile i corresponds to location n and equals 0 otherwise; $LOADOUT_{n,j}$ equals -1 if if the origin of load j is location n and equals 0 otherwise; $LOADIN_{n,j}$ equals 1 if if the destination of load j is location n and equals 0 otherwise; $x^k_{i,j}$ is a decision variable equal to 1 if load j is tranported in trailer position k during leg i and equal to 0 otherwise; and dom_i is a decision variable equal to 1 if domcile i is the domicile of the route and equal to 0 otherwise.

The first constraint *(3.2.1)* ensures that there is one and only one domicile chosen for a route. The two following groups of constraints ensure that the route will start at a domicile *(3.2.2)* and that it will end at a domicile *(3.2.3)*. The two next set of constraints *(3.2.4)* and *(3.2.5)* ensure that the last leg is destined to the domicile. Similarly, constraints *(3.2.6)* and *(3.2.7)* ensure that the first leg of the route originates from the domicile.

3.4.3 Origin and destination of a leg

Here, we will present constraints enforcing the relationship between the loads carried and the destination and the origin of a leg.

- $\sum_{n \in Locations} origin_{i,n} = -1, \forall i \in Legs$ *(3.3.1)*
- $\sum_{n \in Locations} dest_{i,n} = 1, \forall i \in Legs$ *(3.3.2)*
- $\forall n \in Locations,\ dest_{i,n} + origin_{i+1,n} = 0, \forall i \in \overline{Legs}$ *(3.3.3)*

Constraints *(3.3.1)* and *(3.3.2)* state that for each leg, there is exactly one and only origin and exactly one destination. Constraints *(3.3.3)* ensure that the destination of leg i matches the origin of leg $i+1$.

3.4.4 Origin and destination of a the leading load

In this section , we model the relationships between the origin and the destination of a leg and of the leading load.

- $dest_{i,n} \geq \sum_{j \in Loads} leading_{i,j} \times LOADIN_{n,j}\ \ ,\forall i \in Legs, \forall n \in Locations$ *(3.4.1)*
- $-origin_{i,n} \geq \sum_{j \in Loads} leading_{i-1,j} \times LOADIN_{n,j}\ ,\forall i \in \underline{Legs}, \forall n \in Locations$ *(3.4.2)*

with $leading_{i,j}$, a decision variable equal to 1 if load j leads leg i and equal to 0 otherwise.

As stated earlier, in the case where there is at least one load, the origin and destination of the leg are be those of one of the loads. The load which directs the leg is referred to as the leading load. Hence, contraints *(4.4.1)* ensure that the destination is that of the leading load. Next, contraints *(4.4.2)* ensure that the origin of the leg corresponds to the destination of the leading load at the previous leg except for the first leg which start at the domicile of the route. This is done through constraints *(3.4.3)* and *(3.4.4)*.We note there can be no load tranported at leg i and thus no leading load. In that case $\forall j \in Loads$, $leading_{i,j} = 0$ and the destination and the origin are not constrained.

3.4.5 Choice of the leading load

This subsection deals with the constraints concerning the choice of the leading load and the relation between its origin and destination,

- $leading_{i,j} \geq x^1_{i,j} - choiceLeading_i, \forall i \in Legs, \forall j \in Loads$ *(3.5.1)*
- $leading_{i,j} \geq x^2_{i,j} - 1 + choiceLeading_i, \forall i \in Legs, \forall j \in Loads$ *(3.5.2)*
- $1 - choiceLeading_i \leq \sum_{j \in Loads} x^1_{i,j} + noLoad_i, \forall i \in Legs$ *(3.5.3)*
- $choiceLeading_i \leq \sum_{j \in Loads} x^2_{i,j} + noLoad_i, \forall i \in Legs$ *(3.5.4)*
- $\sum_{j \in Loads} leading_{i,j} \leq 1 - noLoad_i, \forall i \in Legs$ *(3.5.5)*
- $\sum_{j \in Loads} leading_{i,j} \geq \sum_{j \in Loads} x^1_{i,j}, \forall i \in \overline{Legs}$ (3.5.6)
- $\sum_{j \in Loads} leading_{i,j} \geq \sum_{j \in Loads} x^2_{i,j}, \forall i \in \overline{Legs}$ *(3.5.7)*
- $noLoad_i \leq 1 - \sum_{j \in Loads} (x^1_{i,j} + x^2_{i,j})/2, \forall i \in Legs$ *(3.5.8)*
- $\sum_{j \in Loads} x^1_{i,j} \leq 1 - noLoad_i, \forall i \in Legs$ *(3.5.9)*
- $\sum_{j \in Loads} x^2_{i,j} \leq 1 - noLoad_i, \forall i \in Legs$ *(3.5.10)*
- $x^2_{i,j} \leq 1 - \sum_{k \in Loads} x^1_{i,k} \times LONG_k \ , \forall i \in Legs, \forall j \in Loads$ *(3.5.11)*
- $\sum_{j \in Loads} x^1_{i,j} + x^2_{i,j} \leq 2, \forall i \in Legs$ *(3.5.12)*
- $xload^1_j \leq \sum_{i \in Legs} x^1_{i,j}, \forall j \in Loads$ *(3.5.13)*
- $xload^2_j \leq \sum_{i \in Legs} x^2_{i,j}, \forall j \in Loads$ *(3.5.14)*
- $xload^1_j + xload^2_j \leq 1, \forall j \in Loads$ *(3.5.16)*
- $x^1_{i+1,j} \geq x^1_{i,j} + choiceLeading_i - 1 - 2 \times bothFree_i \forall i \in \overline{Legs}, \forall j \in Loads$ *(3.5.16)*
- $x^2_{i+1,j} \geq x^2_{i,j} - choiceLeading_i - 2 \times bothFree_i, \forall i \in \overline{Legs}, \forall j \in Loads$ *(3.5.17)*
- $bothFree_i \leq 1 + \sum_{j \in Loads} (x^1_{i,j} - x^2_{i,j}) \times LOADIN_{n,j}, \forall i \in Legs, \forall n \in Locations$ *(3.5.18)*

where $LONG_j$ equals 1 if load j is a long load and equals 0 otherwise; $choiceLeading_i$ is a decision variable equal to 1 if load in trailer position 1 can be the leading load at leg i and equal

to 0 otherwise; $noLoad_i$ is a decision variable equal to 1 if there is no load transported on leg i and equal to 0 otherwise; $bothFree_i$ is a decision variable equal to 1 if trailer position trailer position 1 and 2 are free at the beginning of leg i; and $xload_j^k$ is a decision variable equal to 1 if load j is tranported somewhere in the route in trailer position k and equal to 0 otherwise.

This set of constraints establishes the relationship between the loads held in trailer position 1 and 2 and the load leading. First, contraints *(3.5.1)* and *(3.5.2)* guarantee that if *choiceLeading* equals 0, then the load in trailer position 1, if there is a load, will determine the direction of the leg. Similarly if *choiceLeading* equals 1, then the load in trailer position 2, still if there is a load, will determine the direction of the leg. Constraints *(3.5.3)* and *(3.5.4)* enforce that if there is no load in trailer position 1 and 2, then the choice of the leading load is unconstrained.Contraints *(3.5.5)* enforce that if there is no load, then there is no leading load. Constaints *(3.5.6)* and *(3.5.7)* ensure that if there is a load in trailer position 1 or in trailer position 2 then there is a leading load. Constraints *(3.5.8)* states that if there is a least one load in trailer position 1 or 2, then the binary variable *noLoad* must be set to 0. Conversely, constraints *(3.5.9) and (3.5.10)* state that if the binary indicator *noLoad* is equal 0, then there cannot be any load in trailer position 1 or 2. Constraints *(3.5.11)* ensure that if the load in posisition 1 is either of the type double or override, there cannot be a load in position 2. Constraints *(3.5.12)* ensure that at any time there cannot be more than 2 loads carried by a truck. Constraints *(3.5.13)* and *(3.5.14)* state that $xload_j^k$ has to be 0 if load j is not assigned to any leg in the route. Constraints *(3.5.15)* ensure that a load will be held in at most one given trailer position. Finally, constraints *(3.5.16), (3.5.17)* and *(3.5.18)* ensure the following two cases: 1) the leading load will come to its destination, hence the trailer position where the leading load was held will become free and we will be able to carry another load in that trailer position in the next leg; 2) it might also happen that the destination of the non-leading load might be the same than that of the leading one. In this case, both trailer positions will become free at the end of the leg, as indicated by variable *bothFree* equal to 1.

3.4.6 Pick-up and delivery

In this section, we derive the constraints capturing the relationships between the origins and destinations of the loads and the origins and destinations of the legs.

- $\sum_{j \in Loads} x^1_{i,j} \times LOADOUT_{n,j} \leq origin_{i,n} + 1 - pickup^1_i, \forall i \in Legs, \forall n \in Locations$ *(3.6.1)*
- $\sum_{j \in Loads} x^2_{i,j} \times LOADOUT_{n,j} \leq origin_{i,n} + 1 - pickup^2_i, \forall i \in Legs, \forall n \in Locations$ *(3.6.2)*
- $\sum_{j \in Loads} x^1_{i,j} \times LOADIN_{n,j} \geq dest_{i,n} - 1 + delivery^1_i, \forall i \in Legs, \forall n \in Locations$ *(3.6.3)*
- $\sum_{j \in Loads} x^2_{i,j} \times LOADIN_{n,j} \geq dest_{i,n} - 1 + delivery^2_i, \forall i \in Legs, \forall n \in Locations$ *(3.6.4)*
- $pickup^1_{i+1} \geq x^1_{i+1,j} - x^1_{i,j}, \forall i \in \overline{Legs}, \forall j \in Loads$ *(3.6.5)*
- $pickup^2_{i+1} \geq x^2_{i+1,j} - x^2_{i,j}, \forall i \in \overline{Legs}, \forall j \in Loads$ *(3.6.6)*
- $pickup^1_1 \geq x^1_{1,j}, \forall j \in Loads$ *(3.6.7)*
- $pickup^2_1 \geq x^2_{1,j}, \forall j \in Loads$ *(3.6.8)*
- $delivery^1_i \geq x^1_{i,j} - x^1_{i+1,j}, \forall i \in \overline{Legs}, \forall j \in Loads$ *(3.6.9)*
- $delivery^2_i \geq x^2_{i,j} - x^2_{i+1,j}, \forall i \in \overline{Legs}, \forall j \in Loads$ *(3.6.10)*
- $delivery^1_{nVar} \geq x^1_{nVar,j}, \forall j \in Loads$ *(3.6.11)*
- $delivery^2_{nVar} \geq x^2_{nVar,j}, \forall j \in Loads$ *(3.6.12)*

with $pickup^k_i$ is a decision variable equal to 1 if the load in trailer position k is picked up the beginning of leg i; and $delivery^k_i$ is a decision variable equal to 1 if the load in trailer position k is delivered at the end of leg i

Constraints *(3.6.1)* and *(3.6.2)* enforce that the loads in trailer position 1 and 2 have an origin consistent with the origin of the leg if these loads leave their origin at that given leg. Similarly, constraints *(3.6.3)* and *(3.6.4)* enforce that the loads in trailer position 1 and 2 have a destination consistent with the destination of the leg if these loads arrive at their destination at that given leg. Constraints *(3.6.5)* and *(3.6.6)* ensure that if the load in position k on leg i and $i+1$ are different, then $pickup_{i+1}$ must be equal to 0. Constraints *(3.6.7)* and *(3.6.8)* capture the special case for the first leg (remember there is no leg 0). Similarly, constraints *(3.6.9)* and *(3.6.10)* ensure that if the load on leg $i+1$ in position k is different from that on leg i in the same position k, then delivery of the position k load must occur on leg i. Finally, constraints *(3.6.11)* and *(3.6.12)* are just a special for the last leg.

3.4.7 Time In and Time Out

When a truck enters or exits a location there is some extra time which has to be decounted. These particular extra times are captured through the following constraints:

- $nbLoadsInOut_i = \sum_{j \in Loads} (x^1_{i,j} + x^2_{i,j}), \forall i \in Legs$ *(3.7.1)*
- $M \times moveInOut_i \geq timeLeg_i, \forall i \in Legs$ *(3.7.2)*
- $toInOut_i \geq moveInOut_i - nbLoadsInOut_i, \forall i \in Legs$ *(3.7.3)*
- $2 \times toInOut_i \leq 2 - nbLoadsInOut_i, \forall i \in Legs$ *(3.7.4)*
- $dInOut_i \geq -1 + nbLoadsInOut_i, \forall i \in Legs$ *(3.7.5)*
- $2 \times dInOut_i \leq nbLoadsInOut_i, \forall i \in Legs$ *(3.7.6)*
- $toInOut_i + sInOut_i + dInOut_i \leq 1, \forall i \in Legs$ *(3.7.7)*
- $M \times (toInOut_i + sInOut_i + dInOut_i) \geq timeLegs, \forall i \in Legs$ *(3.7.8)*
- $timeIn_i \geq \sum_{n \in Locations} dest_{i,n} \times TOIN_n - M \times (1 - toInOut_i). \forall i \in Legs$ *(3.7.9)*
- $timeIn_i \geq \sum_{n \in Locations} dest_{i,n} \times SIN_n - M \times (1 - sInOut_i), \forall i \in Legs$ *(3.7.10)*
- $timeIn_i \geq \sum_{n \in Locations} dest_{i,n} \times DIN_n - M \times (1 - dInOut_i), \forall i \in Legs$ *(3.7.11)*
- $timeOut_i \geq - \sum_{n \in Locations} origin_{i,n} \times TOOUT_n - M \times (1 - toInOut_i), \forall i \in Legs$ *(3.7.12)*
- $timeOut_i \geq - \sum_{n \in Locations} origin_{i,n} \times SOUT_n - M \times (1 - sInOut_i), \forall i \in Legs$ *(3.7.13)*
- $timeOut_i \geq - \sum_{n \in Locations} origin_{i,n} \times DOUT_n - M \times (1 - dInOut_i), \forall i \in Legs$ *(3.7.14)*
- $toInOut_i + toInOut_{i+1} \leq 1, \forall i \in \overline{Legs}$

where M is the size of the largest time window, here roughly 50 hours; $TOIN_n$ is the time to enter location n when the truck is in *Tractor Only* configuration; SIN_n is the time for coming into location n when the truck is in *Single* configuration;DIN_n is the time for coming into location n when the truck is in *Double* configuration; $TOOUT_n$ is the time for coming out of

location n when the truck is in *Tractor Only* configuration; $SOUT_n$ is the time for coming out of location n when the truck is in *Single* configuration; $DOUT_n$ is the time for coming out of location n when the truck is in *Double* configuration; $nbLoadsInOut_i$ is decision variable equal to the number of loads being transported between two different location at leg i; $moveInOut_i$ is decision variable equal to 1 if there is the leg's origin differs from its destination and equal to 0 otherwise; $toInOut_i$ is decision variable equal to 1 if the leg's origin differs from its destination and if the truck is in *Tractor Only* configuration and equal to 0 otherwise; $sInOut_i$ is decision variable equal to 1 if the leg's origin differs from its destination and if the truck is in *Single* configuration and equals 0 otherwise; $dInOut_i$ is decision variable equal to 1 if the leg's origin differs from its destination and if the truck is in *Double* configuration, and equal to 0 otherwise; $timeIn_i$ is decision variable equal to the time needed to enter the location at leg i; and $timeOut_i$ is decision variable equal to the time needed to exit the location at leg i.

Constraints *(3.7.1)* insure that $nbLoadsInOut_i$ is equal to the number of loads being transported between the origin and the destination of leg i. For constraints *(3.7.2)*, if the travel time is equal to 0 for a given leg i, then the tractor remains at the same location and there is no movement, hence $moveInOut_i$ will equal 0. Given the other constraints, if the travel time is not equal to 0, however, $moveInOut_i$ must equal 1. Constraints *(3.7.3)* and *(3.7.4)* set $toInOut_i$ equal to 1, if the configuration on leg i is Tractor Only and if the tractor changes location ($nbLoadsInOut = 0$) and if the truck changes location ($moveInOut = 1$). In all other cases, $toInOut$ equals 0. Constraints *(3.7.5)* and *(3.7.6)* ensure that the truck is in Double Configuration if and only if there are two loads being moved. Constraints *(3.7.7)* with constraints *(3.7.8)* enforce that if a trucks moves and if it is not in *Double* nor *Tractor Only* configuration, then it is in *Single* configuration. Contraints *(3.7.9)*, *(3.7.10)* and *(3.7.11)* set the time to enter a location, if the tractor, at least as great as the time specified in the data given the configuration, if and only if the truck moves. Similarly, contraints *(3.7.12)*, *(3.7.13)* and *(3.7.14)* set the time to exit a location for the case in which the tractor leaves the location. Finally, contraints *(3.7.15)* ensure that there cannot be two consecutive legs during which the tractor moves in *Tractor Only* configuration.

3.4.8 Work Time

Total work time on a route is constrained and depends on the domicile to which the route is associated. We express this with the following constraints:

- $edpt_1 - sw \geq 0$ *(3.8.1)*
- $sw = \sum_{j \in Domiciles} dom_j \times SW_j$ *(3.8.2)*
- $fw = \sum_{j \in Domiciles} dom_j \times FW_j$ *(3.8.3)*

where SW_j is the elapsed time for briefing at domicile j before a route can depart; FW_j is the elapsed time for debriefing at domicile j; sw is a decision variable equal to the elapsed time before a route can depart; and fw is a decision variable equal to the elapsed time before the driver's work day end.

Constraint *(3.8.1)* ensure that the actual at which a route begins is positive. Moreover, contraints *(3.8.2)* and *(3.8.3)* ensure that the times for starting work and finishing work for the route are those associated with its domicile.

3.4.9 Extra Time

This set of contraints deals with the small amounts of time needed on a leg for washing and override. These additional times plus the times to enter and exit locations are summed up and captured in a variable $extraTime_i$. These corresponding constraints are expressed as follows:

- $washing_i \geq - \sum_{n \in Locations} origin_{i,n} \times WASH_n - M \times (1 - moveInOut_i),\ \forall i \in Legs$ *(3.9.1)*
- $override_i \geq \sum_{j \in Loads} (x^1_{i,j} + x^2_{i,j}) \times OVERRIDETIME_j, \forall i \in Legs$ *(3.9.2)*
- $extraTime_i = timeOut_i + timeIn_i + washing_i + override_i, \forall i \in Legs$ *(3.9.3)*

where $WASH_n$ is the washing time at location n; $OVERRIDETIME_j$ is the override time for load j; $override_i$ is a decision variable equal to time spent in override at leg i;$washing_i$ is a decision variable equal to the time spent washing a vehicle after leg i; $extraTime_i$ is a decision

variable equal to total extra times at leg i; and $edpt_i$ is a decision variable equal to the earliest departure time from the origin of leg i

Constraints *(3.9.1)* set the washing time to the time specified for the location, if there is some movement from one location to another one on leg i. Constraints *(3.9.2)* set the override time to the time specified for the load. Constraints *(3.9.3)* set the total extra time needed on each leg to the sum of these various "extra" times.

3.4.10 Latest arrival and earliest departure

This subsection dealts with the constraint related to the latest arrival time and the earliest availble time for the loads. Thus, we have:

- $edpt_i \geq \sum_{j \in Loads} x^1_{i,j} \times EAVL_j, . \forall i \in Legs$ *(3.10.1)*
- $edpt_i \geq \sum_{j \in Loads} x^2_{i,j} \times EAVL_j, \forall i \in Legs$ *(3.10.2)*
- $larr_i \leq \sum_{j \in Loads} x^1_{i,j} \times LARR_j, \forall i \in Legs$ *(3.10.3)*
- $larr_i \leq \sum_{j \in Loads} x^2_{i,j} \times LARR_j, \forall i \in Legs$ *(3.10.4)*
- $larr_{nVar} - edpt_1 \leq \sum_{i \in Domiciles} dom_i \times MAXDAY_i - fw - sw$ *(3.10.5)*
- $extraTime_i + timeLeg_i + \leq larr_i - edpt_i \ , \forall i \in Legs$ *(3.10.6)*

where, $EAVL_j$ is the earliest available time for load j; $LARR_j$ is latest arrival time for load j; $MAXDAY_i$ is the maximum working time for a route starting at domicile i; $edpt_i$ is a a decision variable equal to the earliest departure time from the origin of leg i; and $larr_i$ is the decision variable equal to the latest arrival time at the destination of leg i.

Contraints *(3.10.1)* and *(3.10.2)* ensure that the earliest departure time for the origin of leg i cannot be ealier than the earliest available time of the loads on leg i. Similarly, contraints *(3.10.3)* and *(3.10.4)* ensure that the latest arrival time for leg i is no earlier than the earliest available time of the loads on leg i. Constraint *(3.10.5)* enforce that the time difference the latest arrival time back to the domicle and the earliest departure from the domicile should be

less than the maximum actual work time. Finally, contraints *(3.10.6)* ensure that the time difference between the latest arrival time and the earliest departure time for leg i is greater than the sum of the "extra" times plus the travel time of leg i.

3.4.11 Breaks

In what follows, we wil see how we modelize the choice of the break combination and how the breaks occur in the route.

- $firstBreak_i^1 \leq breakChoice, \forall i \in Legs$ *(3.11.1)*
- $mealBreak_i^1 \leq breakChoice, \forall i \in Legs$ *(3.11.2)*
- $secondBreak_i^1 \leq breakChoice, \forall i \in Legs$ *(3.11.3)*
- $firstBreak_i^2 \leq 1 - breakChoice, \forall i \in Legs$ *(3.11.4)*
- $mealBreak_i^2 \leq 1 - breakChoice, \forall i \in Legs$ *(3.11.5)*
- $secondBreak_i^2 \leq 1 - breakChoice, \forall i \in Legs$ *(3.11.6)*
- $\sum_{i \in Legs} firstBreak_i^1 \leq breakChoice$ *(3.11.7)*
- $\sum_{i \in Legs} mealBreak_i^1 \leq breakChoice$ *(3.11.8)*
- $\sum_{i \in Legs} secondBreak_i^1 \leq breakChoice$ *(3.11.9)*
- $\sum_{i \in Legs} firstBreak_i^2 \leq 1 - breakChoice$ *(3.11.10)*
- $\sum_{i \in Legs} mealBreak_i^2 \leq 1 - breakChoice$ *(3.11.11)*
- $\sum_{i \in Legs} secondBreak_i^2 \leq 1 - breakChoice$ *(3.11.12)*
- $M \times \sum_{i \in Legs} (firstBreak_j^1 + 1 - breakChoice) \geq edpt_i - edpt_1 - BREAK_1^1$, $\forall i \in Legs$ *(3.11.13)*
- $M \times \sum_{i \in Legs} (mealBreak_j^1 + 1 - breakChoice) \geq edpt_i - edpt_1 - BREAK_4^1$, $\forall i \in Legs$ *(3.11.14)*

- $M \times \sum_{i \in Legs} (secondBreak_j^1 + 1 - breakChoice) \geq edpt_i - edpt_1 - BREAK_7^1$

$$\forall i \in Legs \qquad (3.11.15)$$

- $M \times \sum_{i \in Legs} (firstBreak_j^2 + breakChoice) \geq edpt_i - edpt_1 - BREAK_1^2,$

$$\forall i \in Legs \qquad (3.11.16)$$

- $M \times \sum_{i \in Legs} (mealBreak_j^2 + breakChoice) \geq edpt_i - edpt_1 - BREAK_4^2,$

$$\forall i \in Legs \qquad (3.11.17)$$

- $M \times \sum_{i \in Legs} (secondBreak_j^2 + breakChoice) \geq edpt_i - edpt_1 - BREAK_7^2,$

$$\forall i \in Legs \qquad (3.11.18)$$

- $noBreak_i = 1 - aBreak_i, \forall i \in Legs,$ (3.11.19)

- $\sum_{i \in Legs} firstBreak_i^1 \geq \sum_{i \in Legs} mealBreak_i^1$ (3.11.20)

- $\sum_{i \in Legs} firstBreak_i^2 \geq \sum_{i \in Legs} mealBreak_i^2$ (3.11.21)

- $\sum_{i \in Legs} mealBreak_i^1 \geq \sum_{i \in Legs} secondBreak_i^1$ (3.11.22)

- $\sum_{i \in Legs} mealBreak_i^2 \geq \sum_{i \in Legs} secondBreak_i^2$ (3.11.23)

where, $BREAK_m^k$ describes the breaks for the combination k, now if

  - $m = 1$, it equals the minimum elapsed time of the route at which the first break can occur;
  - $m = 2$, it equals the maximum elapsed time of the route at which the first break can occur;
  - $m = 3$, it equals the duration of the first break;
  - $m = 4$, it equals the minimum elapsed time of the route at which the meal break can occur;
  - $m = 5$, it equals the maximum elapsed time of the route at which the meal break can occur;

- $m = 6$, it equals the duration of the meal break;
- $m = 7$, it equals the minimum elapsed time of the route at which the second break can occur;
- $m = 8$, it equals the maximum elapsed time of the route at which the second break can occur; and
- $m = 9$, it equals the duration of the second break;

Next, $firstBreak_i^k$ is a decision variable equal to the elapsed time of the route at which the first break occurs if the break occurs during leg i and if we have chosen break combination k, and equal to 0 otherwise; $mealBreak_i^k$ is a decision variable equal to the elapsed time of the route at which the second break occurs if the break occurs during leg i and if we have chosen break combination k, and equal to 0 otherwise; $secondBreak_i^k$ is a decision variable equal to the elapsed time of the route at which the second break occurs if the break occurs during leg i and if we have chosen break combination k, and equal to 0 otherwise; $breakChoice$ is a decision variable equal to 0 if break combination 1 is chosen and equal to 0 if break combination 2 is chosen; $aBreak_i$ is a decision variable equal to 1 if there is a break occuring at leg i and equal to 0 otherwise; and $noBreak$ is a decision variable equal to 0 if there is no break occuring at leg i and equal to1 otherwise.

Constraints *(3.11.1)* to *(3.11.6)* ensure that we select one combination for the break; $firstBreak_i^k$, $secondBreak_i^k$ and $thirdBreak_i^k$ can take a value different of 0 if break combination k is chosen. Constraints *(3.11.7)* to *(3.11.12)* ensure that each break is be taken at most once. Constraints *(3.11.13)* to *(3.11.18)* guarantee that breaks occur within the time windows provided in the data. Constraints *(3.11.20)* just make the connection between the two opposite indicators $noBreak_i$ and $aBreak_i$. Finally, constraints *(3.11.21)* and *(3.11.22)* allow the driver to stop for his meal if he has already taken a break. Similarly, constraints *(3.11.23)* and *(3.11.24)* allow the driver can to stop for his second break if he has already had his meal.

3.4.12 Time

In this section, we introduce the constraints linking break times and earliest departure and latest arrival times.

- $extraTime_i + timeLeg_i + firstBreak_i^1 \times BREAK_3^1 + mealBreak_i^1 \times BREAK_6^1$
$+secondBreak_i^1 \times BREAK_9^1 \leq larr_i - larr_{i-1}.\ \forall i \in \underline{Steps}$ *(3.12.1)*

- $extraTime_i + timeLeg_i + firstBreak_i^2 \times BREAK_3^2 + mealBreak_i^2 \times BREAK_6^2$
$+secondBreak_i^2 \times BREAK_9^2 \leq larr_i - larr_{i-1}, \forall i \in \underline{Steps}$ *(3.12.2)*

- $extraTime_{i-1} + timeLeg_{i-1} + firstBreak_{i-1}^1 \times BREAK_3^1 + mealBreak_{i-1}^1 \times BREAK_6^1$
$+secondBreak_{i-1}^1 \times BREAK_9^1 \leq edpt_i - edpt_{i-1}, \forall i \in \underline{Steps}$ *(3.12.3)*

- $extraTime_{i-1} + timeLeg_{i-1} + firstBreak_{i-1}^2 \times BREAK_3^2 + mealBreak_{i-1}^2 \times BREAK_6^2$
$+secondBreak_{i-1}^2 \times BREAK_9^2 \leq edpt_i - edpt_{i-1}, \forall i \in \underline{Steps}$ *(3.12.4)*

- $larr_i \geq firstBreak_i^1 \times (BREAK_1^1 + BREAK_3^1) + edpt_1, \forall i \in \overline{Legs}$ *(3.12.5)*

- $larr_i \geq firstBreak_i^1 \times (BREAK_1^1 + BREAK_3^1) + edpt_1, \forall i \in \overline{Legs}$ *(3.12.6)*

- $larr_i \geq mealBreak_i^1 \times (BREAK_4^1 + BREAK_6^1) + edpt_1, \forall i \in \overline{Legs}$ *(3.12.7)*

- $larr_i \geq firstBreak_i^2 \times (BREAK_1^2 + BREAK_3^2) + edpt_1, \forall i \in \overline{Legs}$ *(3.12.8)*

- $larr_i \geq mealBreak_i^2 \times (BREAK_4^2 + BREAK_6^2) + edpt_1, \forall i \in \overline{Legs}$ *(3.12.9)*

- $larr_i \geq secondBreak_i^1 \times (BREAK_7^2 + BREAK_9^2) + edpt_1, \forall i \in \overline{Legs}$ *(3.12.10)*

- $larr_i \geq secondBreak_i^2 \times (BREAK_7^2 + BREAK_9^2) + edpt_1, \forall i \in \overline{Legs},$ *(3.12.11)*

- $edpt_i \leq firstBreak_i^1 \times BREAK_2^1 + edpt_1 + (1 - firstBreak_i^1) \times M, \forall i \in Legs$ *(3.12.12)*

- $edpt_i \leq firstBreak_i^2 \times BREAK_2^2 + edpt_1 + (1 - firstBreak_i^2) \times M, \forall i \in Legs$ *(3.12.13)*

- $edpt_i \leq mealBreak_i^1 \times BREAK_5^1 + edpt_1 + (1 - mealBreak_i^1) \times M, \forall i \in Legs$ *(3.12.14)*

- $edpt_i \leq mealBreak_i^2 \times BREAK_5^2 + edpt_1 + (1 - mealBreak_i^2) \times M, \forall i \in Legs$ *(3.12.15)*

- $edpt_i \leq secondBreak_i^1 \times BREAK_8^1 + edpt_1 + (1 - secondBreak_i^1) \times M,$
$\forall i \in Legs$ *(3.12.16)*

- $edpt_i \leq secondBreak_i^2 \times BREAK_8^2 + edpt_1 + (1 - secondBreak_i^2) \times M,$
$\forall i \in Legs,$ *(3.12.17)*

Constraints *(3.12.1)* and *(3.12.2)* ensure that the time differences between two consecutive latest arrival times should be greater than the sum of the time required by the different events (extra time, travel time and break time) occuring between on the leg between these arrivals. Similarly, constraints *(3.12.3)* and *(3.12.4)* ensure that a leg does not depart before its predecessor can arrive. Constraints *(3.12.5)* to *(3.12.11)* ensure that if a break occurs on leg i is a least as great as the earliest ime for starting the break plus the duration of the break. Similarly, constraints *(3.12.12)* to *(3.12.17)* ensure that if a break occurs on leg i then the earliest departure time is not later than the latest time for the break to occur.

3.5 Impact of the large number of variables and constraints

As stated in Section 2.3, the set partitioning master problem and its relaxation are easy problems to solve. The pricing problem however is much more difficult to solve to optimality.

3.5.1 Analysis of the number of variables and constraints

Our formulation for the pricing problem involves a very large number of variables and constraints. For a given number of legs, denoted $nVar$, in the routes, we compute the number of constraints and variables included in the model. The results are summarized in the table below:

$nVar$	# Constraints	# Variables
3	13721	7546
4	18138	9815
5	22555	12084
6	26972	14353
7	31389	16622
8	35806	18891

Table 3.1: Number of contraints and variables for a given number of route legs

We see that even for small values of $nVar$ the number of constraints and variables in this mixed integer program suggests tractability might be an issue unless heuristics are employed.

3.5.2 First computational results

We can study the behaviour of the algorithm for different values of $nVar$ and conclude that the pricing problem even for small values of $nVar$ poses insurmontable tractability challenges. Specifically, for $nVar = 5$, it is difficult to find a feasible solution within a resonable amount of time (less than 5 minutes).

Our experience shows that the more legs allowed for the route, the harder it is for the CPLEX solver to find a starting solution. Moreover, memory requirements increases sharply.

3.6 Heuristics for solving the pricing problem

3.6.1 Initializing the pricing problem with a starting basis

We have stressed that the solver has difficulty an initial solution. A simple approach is to construct an initial solution in which each route remains at a domicile At first thought, we could think giving a starting basis to the solver. A simple approach is to construct an initial solution in which each route reamins at a domicile during $nVar$ legs. Although this initial basis indeed does provide an initial solution, it is far away from optimal. Another idea is to build routes in a gradual manner. We start from a small value of $nVar$ for which we can easily find a good route with. Next, we translate this route into a set of constraints (which will be detailed subsequently) and we plug these constraints into a new model with a larger value of $nVar$. We increase $nVar$ by one or two units and solve the model again. A clear advantage of this approach is that it enables us to find feasible routes with a large number of legs, which might be good candidates for entering the basis of the master problem.

To illustrate the mechanics of this heuristics, we can consider the example of Figure 3-2 containing a route computed with $nVar = 5$.

We can transform this route into a set of constraints as follows:

- Aditional Constraints for $x^1_{i,j}$:

  - $x^1_{1,23} = 1$;
  - $x^1_{2,2} = 1$;
  - $x^1_{3,206} = 1$;

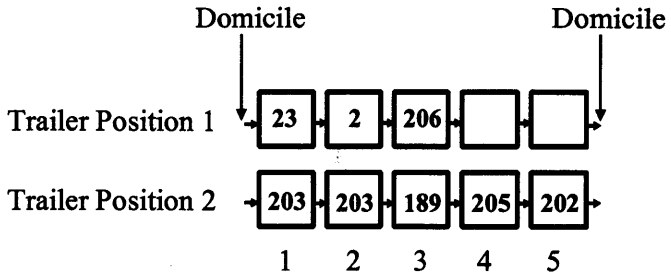


Figure 3-2: Example of route transformed into constraints

- Aditional Constraints for $x^2_{i,j}$:

  - $x^2_{1,203} = 1$;
  - $x^2_{2,203} = 1$;
  - $x^2_{3,189} = 1$; and
  - $x^2_{4,185} = 1$.

Note that the constraint $x^2_{5,202} = 1$ for the last leg is not included to allow the route to be expanded rather than terminating at its domicile. Another possible variation of this method includes removing the loads on the first leg to allow for other domicile locations for the route. This has the advantage of allowing us to expand the route on both sides (with respect to Figure 3-2). Finally, we could also translate the indices of the $x^1_{i,j}$ from i to $i+1$ in order to expand also on the left side of the route. We do not however implement this variant and we will allow the route to expand only on one side. This has the consequence that the route depends on the order in which loads are added.

3.6.2 Generating the desired instance with a class of routes

In order to generate the desirable instance of routes, we generate a set of routes add a penalty term to the objective function and generate a set of routes, all of which lie in the same neighborhood. That is, once the solver has found a route, it can generate additional routes of interest to us by adding to the objective function the following penalty term:

$$\lambda \sum_{i \in Legs} (larr_i - edpt_i)$$

where $\lambda < 0$ and $|\lambda|$ very small

To minimize this penalty term, we minimize the differences between $larr_i$ and $edpt_i$ for each leg i. This has the effect of removing slack and making $timeLeg_i$ tight, thereby reducing travel time to the minimum.

3.6.3 Alternative objective function

Because the current objective function of our heuristic does not capture driver productivity, we need to invoke a more sophisticated process when we build routes with a small number of legs to ensure that we build routes that fill driver's work time to the greatest extent possible. We achieve this by modifying dual variable values to reflect the benefit associated with each load in a route. We relate benefit to the minimum routing time of the load, and whether a load is short or long.

Our modified objective is of the form:

maximize *the sum of the minimum routing time of the loads transported within the route plus the correction terms related to load times and the dual for each load modified by the penalty term*

Mathematically, we formulate this as follows:

$$\begin{aligned}\text{maximize} \sum_{j \in Loads} & ((xload_j^1 + xload_j^2) \times (minTimeLoads_j \times (single_j + \alpha \times LONG_j) \\ & + \beta \times OVERRIDETIME_j) \times (\gamma + \delta \times DUAL_j)) \; + \lambda \sum_{i \in Legs} (larr_i - edpt_i)\end{aligned}$$

where, $minTimeLoads_j$ is the minimum time needed to bring load j from its origin to its destination; $single_j$ equals 1 if load j is a single-load, equals 0 otherwise; $LONG_j$ equals 1 if load j is a double-load, equals 0 otherwise; $OVERRIDETIME_j$ is the override time for load j; $DUAL_j$ is the master problem's dual values for load j; $\alpha, \beta, \gamma, \delta$ are parameters to be discussed later; and $xload_j^k$ is abinary variable equal to 1 if load j has been tranported in trailer position k, and equal to 0 otherwise.

Typically, parameter α, which ponderates the relative benefit of carrying a long load compared to a short load, should take a value between 1 and 2. Parameter β, which reflects the relative benefit of an override compared to a classic load, should take a value close to 1. Finally, parameters γ and δ adjust the magnitude of the dual vector. To keep the dual vector close to ist value, γ should take a value close to 0 and δ a value close to 1. To determine appropriate values for the parameters α, β, γ and δ, we must make several trials with different values.

3.6.4 Additional mechanics of the heuristic

We still need to specify a few rules before having the complete picture of the heuristic used:

1. We define $nVarMin$ as the number of legs we will consider when building the first route (the route with the smallest number of legs). In order to ensure that there will be a feasible route, we set $nVarMin \geq 3$. We also define $nVarMax$ as the number of legs we will consider for the longest routes, those generated in last step.

2. We generate multiple routes at each iteration between the master and the pricing problem. Because, we can expand routes on one side only for each route, we define a compulsory load, that is a load which has to be picked up in the expanded route. This allow us to generate routes in which the order of the loads varies, and facilitates the construction of sets of routes transporting all loads carrying all loads. By successively defining each load as a compulsory load, we generate as many routes as there are loads at each iteration between the master and the pricing problem. To fix a compulsory load in a route, we add the following constraint:

$$\sum_{j \leq nVarMin} xload_j^1 + xload_j^2 = 1$$

We start by selecting load 1 as the compulsary load and we finish by selecting load 364 as the compulsory load. The process is summarized in Figure 3-3.

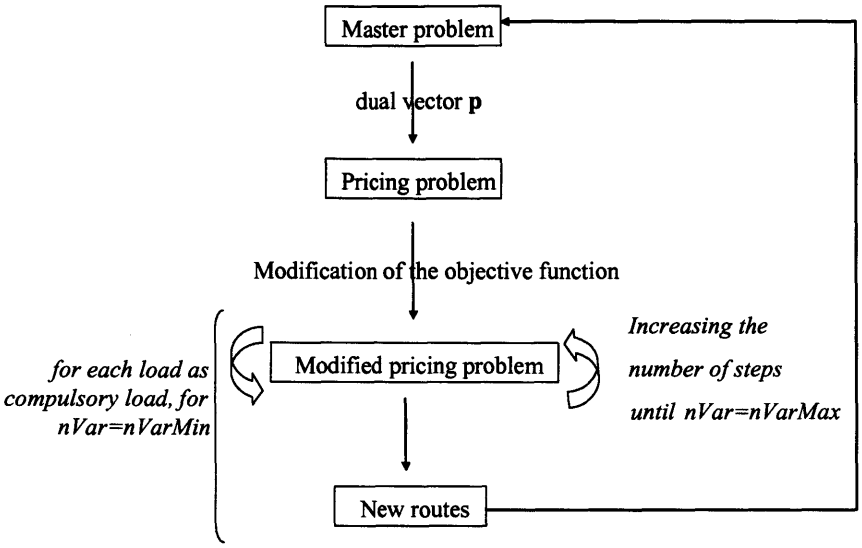


Figure 3-3: Mechanics of the heuristic

3.7 Results

We investigate whether the linear relaxation of our master problem is close to the solution to the original master problem. From Table 3.2 below, we see that the differences between the solutions of the relaxed problem and those of the integer problem never exceed 8%.

Routes Generated	Integer Problem Optimum	Relaxed Problem Optimum
382	155	152.2
464	146	144.4
638	125	134.1
996	107	104.3
1386	105	102.2
1581	96	93.2
1911	94	91.8
2369	93	89.6
2985	93	89.1

Table 3.2: Results of the heuristic for the integer and the relaxed master problems

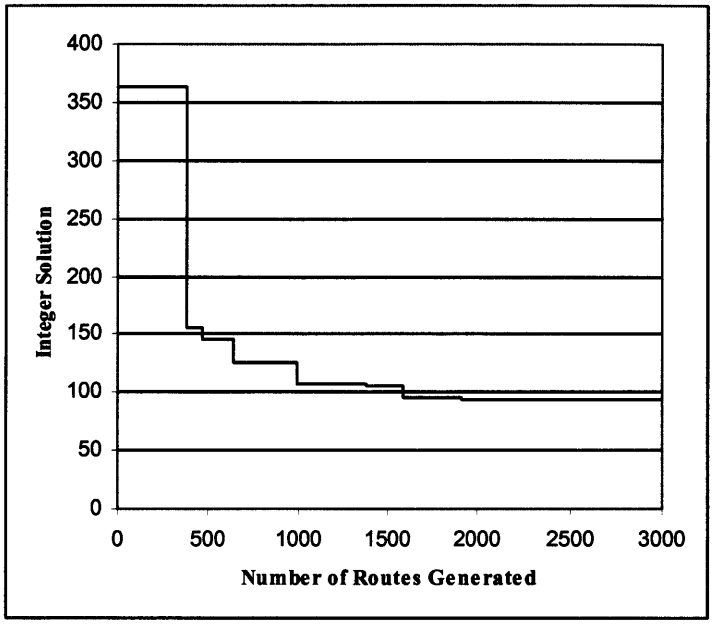


Figure 3-4: Evolution of the solution wih respect to the number of routes generated

We see that the solution improves greatly with the first iterations, but it quickly becomes more and more difficult to improve the solution. With the computing power available, we cannot find a schedule with fewer than 93 drivers.

Chapter 4

Sequential approach for generating routes using clusters of loads

4.1 Definition and motivation

4.1.1 Motivation

A sequential method constitutes an alternative to what was described in Chapter 3. The column generation method was adversely affected by the very large number of constraints and variable arising in the pricing formulation. Here, we want to develop a method in which we do not have to deal simultanously with such a large constraints and variables.

The problem is to find both the paths of the loads and the routes of the drivers. We solve these two problems somewhat separately. Doing so, we know that it is unlikey that we will find the optimal solution but we hope to find a near-optimal solution. The idea is to partially specify paths of the loads using heuristics. The problem then reduces to finding adequate routes to cover these paths. To identify paths, we first assemble short load together in a preprocessing step. This fixes which load is tranported with which load.

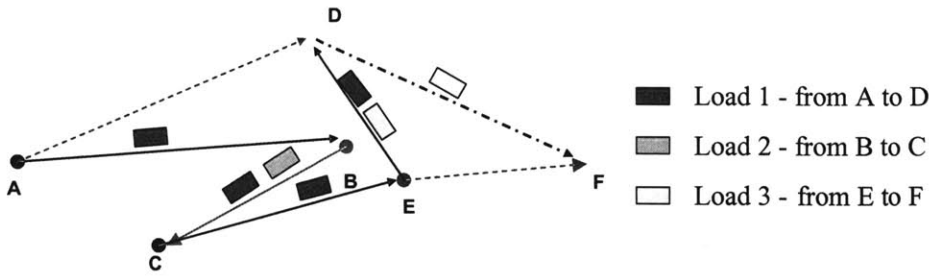


Figure 4-1: Cluster with 3 loads

4.1.2 Definition of a cluster

Cluster

We introduce the concept of clusters. Basically a cluster is a twofold extension of the concept of loads; namely it first aggregates loads together, and second, it defines the paths of these aggregated loads. The clusters can, hence, be seen as partitions of routes.

Clusters are characterized by three elements.

1. The set of loads included in the cluster;
2. The paths of these loads; and
3. Time windows at each location.

Figure 4-1 provides an illustration of a cluster with 3 loads and the paths of these loads. We note that time windows for a clusters is determined by time windows for the loads, as detailed in a following section

Pre-cluster

We constuct clusters in a gradual manner using the additional concept of pre-cluster. A pre-cluster is a cluster for which a portion of the path might not be defined. The portion of the path which remains unknown is that corresponding to the last leg. However, if we do not know exactly the path we still have information about the origin and destination of the last leg based on the destination of the current last leg and the undelivered trailer. Thus we can tranform a

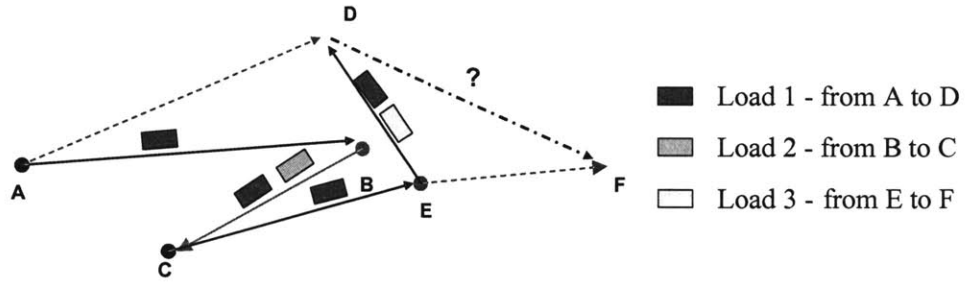


Figure 4-2: Pre-cluster with 3 loads

pre-cluster into a proper cluster if we select a shortest path as the path of the last leg. Note that a load, as defined throughout this thesis, is a pre-cluster.

We can list the elements characterizing the pre-cluster.

1. The set of loads included in the pre-cluster
2. The paths of these loads except for the last leg
3. Time windows at each location except for the two last locations where we consdier an earliest available time and latest arrival for the load carried during the last leg

A pre-cluster can be seen as a cluster plus one undelivered "load" refered to as the pseudo-load of the cluster. For example, in figure 4-2, the origin of the pseudo load is D and its destination is F, its true destination. A pseudo-load's destination is always the destination of an actual load. However, its origin is either the origin or the destination of an actual load.

From now on, we will use the generic term cluster for designing either a cluster or a pre-cluster; we can always transform a pre-cluster into a cluster by determining the path of the pseudo-load and adding it to the last leg of the pre-cluster. We restrict ourselves to clusters including only short loads because our main objective is to determine clusters allowing short loads to be transported together in an efficient manner.

Some properties of the clusters

The null cluster, consisting of no load and associated with the time window [0,4800], is a valid cluster. Moreover, any load can be added to the null cluster to create a valid cluster. Hence,

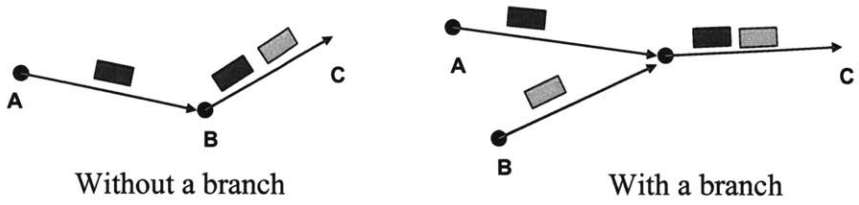


Figure 4-3: Cluster with a branch and cluster without a branch

any load can be added to the null cluster to create a valid cluster. Hence, a single load is cluster.

Denote by $\mathcal{C}$ the set of clusters, $\mathcal{S}$ the set of single loads and $\mathcal{S}_c$ the set of single loads compatible with pre-cluster c, where a load is compatible with a cluster if its inclusion with the cluster results in a feasible cluster. Then, we know the following properties of clusters:

- $(s + c) \in \mathcal{C}, \forall c \in \mathcal{C}, \forall s \in \mathcal{S}_c$
- $0 \in \mathcal{C}$
- $\mathcal{S} \subset \mathcal{C}$

Limitation in the geometry of the clusters: no branch

In this section, we describe further restrictions on cluster. We distinguish two types of clusters: with a branch or without a branch. Figure below 4-3 illustrates this distinction.

Clusters without a branch define paths which can always be performed by a single driver; whereas, clusters with a branch might require several drivers; hence, cluster which might require exchanges. Because we do not consider exchanges, we do not consider clusters with a branch.

Difficulties when dealing with clusters

We have two types of time windows in our problem. There are time windows which are defined in absolute time (EAVL, LARR) and time windows (such as for breaks) which are defined in relative time because they depend on the time at which the route begins. Clusters are defined in absolute time time, but we do not know the starting time of the route in which they will

ultimately be included. Hence, we relax time windows constraints for the breaks and ensure only that total time of breaks is accounted for in total time of the route.

Remark *A more complicated way to deal with this issue, but which would have guranteed that everything is feasible, is to generate clusters including different amounts of slack time in order to account for the different lengths of breaks.*

4.2 Generation of clusters of loads

In this section we present the different steps to generate clusters.

4.2.1 Matrix of Shortest Paths

We compute the table of shortest paths between any two locations in the network. We have described in Section 2.2 two methods for computing these shortest paths. We consider here the shortest paths in time and not those in distance. As already mentioned, the time needed for traveling along an arc depends on the direction of travel, therefore we generate a non-symetrical matrix of shortest path.

The table of shortest paths will thus have the following form. The first column denotes the origin and the first line the destination.

	1	2	$\cdots$	98
1	0	55	$\cdots$	97
2	57	0	$\cdots$	38
$\vdots$	$\vdots$	$\vdots$	$\ddots$	15
98	85	42	17	0

Table 4.1: Table of shortest path

4.2.2 Earliest - latest visit vectors

We compute for each load a vector representing the earliest visit times and the latest visit times at every location in the network. Based on loads earliest availability and latest delivery time, the earliest visit time corresponds to the earliest time at which the load can be at a

given location and the latest visit time corresponds to the latest time at which a load can be at destination.We take into accounts the amount of time to enter and exit each location and compute the earliest and latest visit vectors at location X for load i as follows:

$$earliest(X) = EAVL + d(A, origin) + Out(A) + In(X) + Wash(X)$$
$$latest(X) = LARR + d(X, destination) + Out(X) + In(A) + Wash(A)$$

where $EAVL$ is the earliest availability of the load; $LARR$ is the latest delivery time of the load; $earliest(X)$ is the earliest visit time at location X; $latest(X)$ is the latest visit time at location X; *origin* is the origin of the load; *destination* is the destination of the load; $d(X, Y)$ is the shortest path in time between location X and location Y; $In(X)$ is the time to enter location X; $Out(X)$ is the time to exit location X; and $Wash(X)$ is the washing time at location X

Knowing the time windows for a load and the shortest-path matrix, we can determine time windows for visiting all the other locations in the graph. We note that $In(X)$ and $Out(X)$ not only depends on the location but also on the configuration of the tractor. In a cluster, the tractor will never travels empty, hence, we only have to consider the *Single* and *Double* configurations. We do not beforehand, however what the configuration will be. Consequently, we compute earliest and latest visit vectors for both the *Single* and *Double* configurations.

Example

We consider the following example in which:

1. The origin of the load is A and its destination is B;
2. The load's earliest available time is 0, that is $EAVL = 0$;
3. The load's arrival time is 100, that is $LARR = 100$;
4. The times for exiting and entering all locations are 0 whatever the configuration;
5. At every location, washing time is set to 0; and
6. The underlying network contains locations A, B, C and D and the shortest path computed for the network are presented in Table 4.2.

	A	B	C	D
A	0	55	25	97
B	57	0	52	38
C	23	65	0	15
D	85	42	17	0

Table 4.2: Table of shortest path (example)

Given that the load has $EAVL = 0$, $LARR = 100$, and a minimum travel time between the origin and the destination of 60, the latest visit time at the origin is $100 - 60 = 40$. The visit times at all other locations, which are similarly computed, are depicted in Table 4.3

	earliest	latest
A	0	45
B	55	100
C	25	48
D*	9999	0

Table 4.3: Earliest and latest visit times

*In some cases, it is impossible for a load to visit a particular location while respecting its time windows. In these cases, we set the earliest visit time to 9999 and the latest visit time to 0.

4.2.3 Addition of successive loads to a cluster

We will constuct clusters in a gradual manner starting with a load which defines the first cluster, and subsequently adding loads one by one.

Two types of insertion

Depicted in Figure 4-4 are two types of additions that result when adding a load to a cluster without a branch.

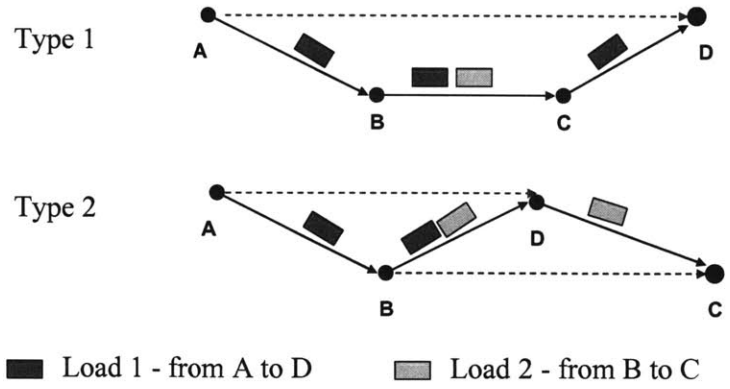


Figure 4-4: Two types of additions for clusters without a branch

Denote load 1 as the pseudo load of the cluster. A *Type 1* insertion addition involves the addition of load 2 whose path is totally included in the path of the pseudo load. A *Type 1* insertion involves the addition of load *2* whose path is not totally included in the path of the pseudo-load.

Adding successive loads to a cluster

In this section, we will develop the mechanism by which we add loads to a cluster. Figure 4-5 details two successive addition. First, load 2 is added to load 1 with a *Type 1* insertion. The updated cluster contains two links, namely AB and CD, to which additional loads can be assigned. Next, as shown in Figure 4-6, we add load 3 (*Type 2* insertion), by inserting a node E in the leg CD and creating a DF leg and assigning load 3 on the ED and DF legs.

The order in which loads are added to the cluster affect the structure of the cluster. Hence, if we consider three loads $1, 2, 3$ in different orders, we can generate different clusters, namely, $(1, 2, 3)$ does not necessarily equal$(2, 1, 3)$ which might not equal $(3, 1, 2)$.

4.3 Cluster construction algorithm

The algorithm to construct clusters involves the following three steps:

1. Check the compatibility of the loads;

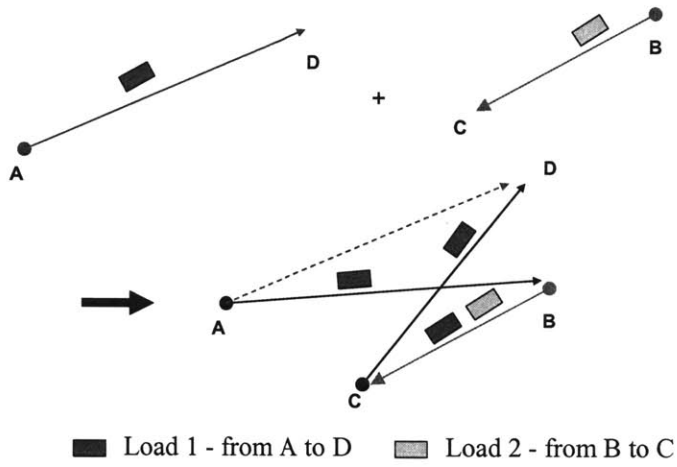


Figure 4-5: Adding successive loads to a cluster - Type 1

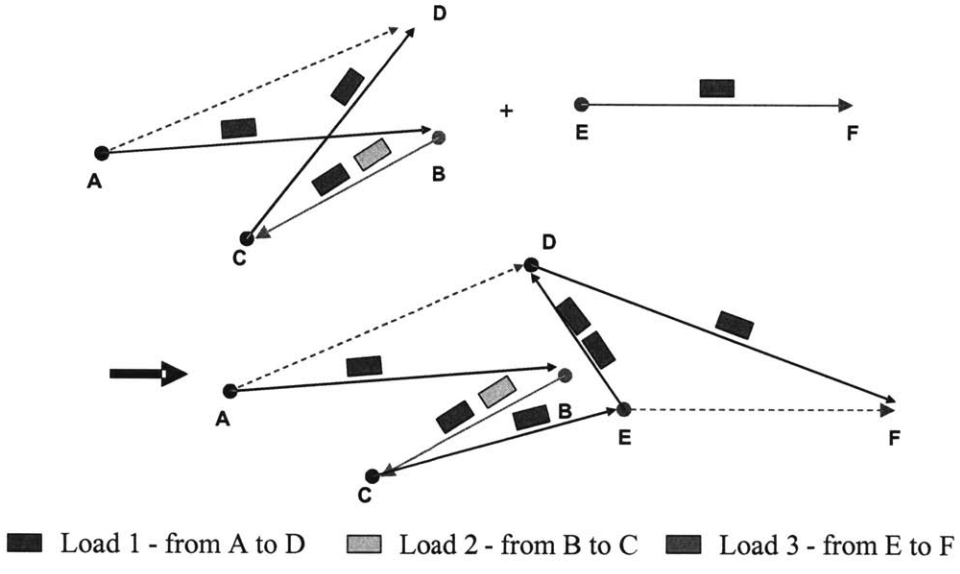


Figure 4-6: Adding successive loads to a cluster - Type 2

2. Update the pre-cluster; and

3. Allow locations to be included multiple times in a cluster.

Compatibility with the cluster of the loads to be added

The first thing we do is to check whether the load to added to the cluster is compatible with the cluster. The conditions of compatibilty differ with respect to the type of addition considered. Let P designate the the pseudo-load; L designate the load to be added; $earliest(U, X)$ designate the earliest visit time for load U at location X, $latest(U, X)$ designate the latest visit time for load U at location X; $origin(U)$ designates the origin of load U and $destination(U)$ the destination of load U; and $d(X, Y)$ designate the shortest path between location X and location Y.

- **For a type 1 addition,** we ensure that:

  - $earliest(P, origin(L)) \leq latest(L, origin(L))$; and
  - $earliest((P, destination(L)) \leq latest(L, destination(L))$

  Note that if the pseudo-load and the load to be added have the same origin, we compute the earliest visit vector with $In()$and $Out()$ for a *Double* configuration; if the origins are different then we use the values for the *Single* configuration. Moreover, if the pseudo-load of the cluster and the load to be added have the same destination, we compute the latest visit vector with $In()$and $Out()$ for a *Double* configuration; if the origins are different then we use the values for the *Single* configuration.

- **For type 2 addition**

  We have to check that

  - $earliest(P, origin(L)), earliest((P, destination(L))$
    $+d(origin(L), destination(P)) \leq latest(P, destination(P))$
  - $earliest(P, origin(L)), earliest((P, destination(L))$
    $+d(origin(L), destination(P))+d(destination(P), destination(L)) \leq latest(L, destination(L))$

Pseudo load update

We can derive rules for updating the pseudo-load characteristic after an addition. If the load L is compatible with the pseudo load of the cluster, we add load P to the cluster. The new pseudo load P^* will be characterized by:

- **For a type 1 addition**
  - $origin(P^*) = destination(L)$
  - $destination(P^*) = destination(P)$
  - $LARR(P^*) = LARR(P)$
  - $EAVL(P^*) = \max(earliest(L, destination(L)), earliest(P, destination(L))$
- **For a type 2 addition**
  - $origin(P^*) = destination(P)$
  - $destination(P^*) = destination(L)$
  - $LARR(P^*) = LARR(L)$
  - $EAVL(P^*) = \max($ $earliest(P, origin(L)), earliest((P, destination(L)))$

    $+d(origin(L), destination(P))$

We can implement this on an example. We consider a *Type 1* addition. The new pseudo load has C for origin and D for destination.

	earliest	latest
A	5	45
B	35	65
C	25	60
D	25	100

cluster load 1: A to B

and

	earliest	latest
A	9999	0
B	10	85
C	30	105
D	60	95

load 2: B to C

$\Longrightarrow$

	earliest	latest
A	9999	0
B	75	90
C	30	60
D	80	100

cluster (1,2): C to D

Table 4.4: Pseudo-load update

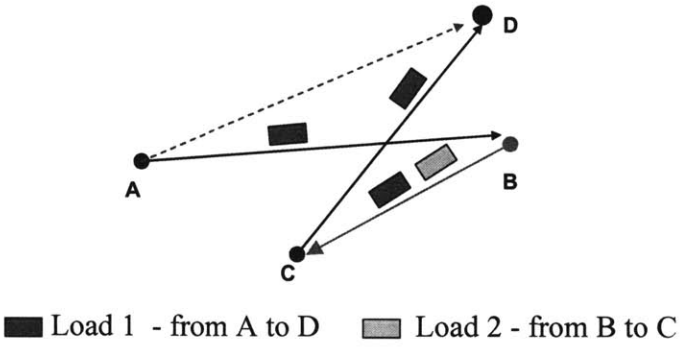


Figure 4-7: Constructing a Cluster

Update of the time windows at each location in the path

At each step we update the earliest and latest visit times at all locations. Location visited more than once can have different time windows, in which case, we create copies of these locations.

4.3.1 Generating clusters

Because the order in which the loads are added to the cluster matters, we develop a procedure to generate as many clusters as possible. For each short load, we enumerate all possible clusters recursively as shown in Figure 4-8. Each tree has a different root corresponding to a different short load. Starting at the root node of the tree, we explore the tree in depth-first manner, at each node we try to add the load, represented by a branch of the node, to the cluster. If we can add the load we later explore its corresponding node and branches. If the load is incompatible, we prune the tree above the node corresponding to the incompatible load, and continue the process, selecting the next unexplored node in the tree. Hence in Figure 4-8, we create the clusters in the following order $(A, AB, ABC, ABCD, ABD, AC, ACB, ACBD)$. However, to avoid too long clusters, given that multiple clusters are combined into a route, we restrict clusters to a maximum of five loads.

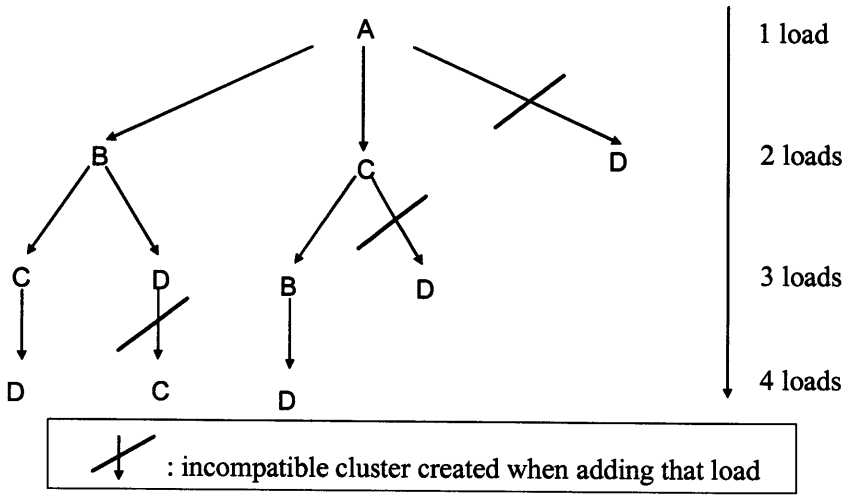


Figure 4-8: Visiting recursely the clusters

4.4 Selection of a subset of Clusters

4.4.1 Pre selection of clusters

Due to insufficient memory, we cannot exploit all the clusters visited. First, we discard all the clusters which cannot be possibly covered in route, that is the clusters for which the sum of the times to drive from a domicile to the beginning of the cluster, to cover the cluster and to drive from the end of the cluster to the domicile is greater than the maximum work time. Next, to decide among the remaining clusters which clusters to keep, we compute for each cluster a value G defined as:

$$G = \frac{\sum_{Loads\ in\ Cluter} Shortest\ Path}{Driving\ Time\ in\ Cluster}$$

It is easy to see that G takes values between 0 and 2. The greater the value G of the most promising the promising the cluster. In fact, G close to 2 signifies that we have an efficient grouping of loads; specifically we are close to a situation in which all loads are transported in *Double* configuration. At the opposite, G close to 0 means that most of the loads are transported in *Single* configuration.

We store the n best clusters for each possible size of clusters, that is we save n clusters with 2 loads, n with 3 loads, etc ... From a practical stand point n should take a value around 10.

Remark *We could have used a more sophisticated function for G. Because we are not only interested in the efficiency of the way single are transported in the cluster, we are also interested in the possibility of taking breaks within the cluster. Hence, G is computed as:*

$$G = \frac{\sum_{Loads\ in\ CLuter} Shortest\ Path}{Minimum\ Driving\ Time\ in\ Cluster} + bonusBreak$$

We add an additional term bonusBreak which takes a positive value if there is enough slack time in the cluster for taking a break.

4.4.2 Selection of clusters

At the end of the previous step, we generated a set of clusters denoted *ClusterShort*. We extend *ClusterShort* by adding to it all clusters composed of one short load. From this set we want to select the miminum number of clusters ensuring that every load is transported. For this purpose we use a linear program:

$$\begin{aligned} \text{minimize} \quad & \sum_{j \in ClusterShort} T_j^{Cluster} . x_j \\ \text{subject to} \quad & \sum_{j \in ClusterShort} \Delta_{ij} . x_j = 1, \forall\ i \in Short \end{aligned}$$

where *ClusterShort* is the set of clusters of short loads; *Short* is the set of short loads; $T_j^{Cluster}$ represents the minimum time needed to drive throughout the cluster; Δ_{ij} equals 1 if short load i is included in cluster j and 0 otherwise; and x_j is a decision variable equal to 1 if cluster j is in the solution, and equal to 0 otherwise.

4.4.3 Extension of the set of clusters

In the mechanism we just described, we generate a set of clusters containing all the short loads. We can also contruct an additional set of clusters containing long and override loads. These additional clusters consist of *single-load* clusters in which the paths of the clusters correpond to the shortest paths of the loads, and their time windows are those of the original loads adjusted by the "extra" times. By aggregating these two sets of clusters, we come up with a complete

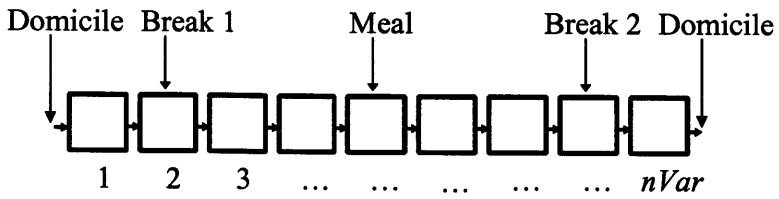


Figure 4-9: Route constructed with clusters

set a of clusters in which all the loads (single, long and ovveride) are included. We denote this complete set of clusters as *Clusters*.

4.5 Modeling routes

To model routes to cover clusters we use an approach very similar to that developed in Chapter 3. The main difference is that we do not have to take into account trailer position because clusters already define the particular combinations of loads.

Recall that a route is defined as a succession of legs in which the first leg starts a domicile and the last leg ends at that same domicile. Figure 4-9 illustrates the composition of a route. In the constext of clusters, each leg of a route either covers one cluster or a *Tractor Only* movement. This movement orginates either at a domicile or at the destination of a cluster, and it ends either at a domicile or at the origin of a cluster. Each leg is characterized by an origin and a destination, an earliest departure time and a latest arrival time, and the cluster which is covered if applicable. Breaks occur along a route route at a stop or within a leg. Because, we cannot model breaks in a route precisely, we allow a break to occur in a leg covering a cluster if the difference between its latest arrival time and the sum of its earliest available time and the length of its path is no less than the duration of the break. Any break occuring at a stop may be assigned to either the leg prior to the stop or to the leg after the stop. Without loss of generality, we assign the break to the later leg. Finally, note that more than one break can be assigned to the same leg. Finally, we require that the routes to start and finish at the same domicile.

4.6 Integer program to generate routes

The integer program we use to generate routes is a simplification of the one detailed in Chapter 3.

4.6.1 Definition of the sets

In this section, we present the notation we will use for the different sets of variables:

- *Clusters*: set of clusters;
- *Locations*: set of locations;
- *Domiciles*: set of domiciles;
- *Arcs*: set of arcs;
- *Legs*: set of legs in a route; $Legs = \{1..nVar\}$;
- $\overline{Legs}$: set of legs in route excluding the last leg; $\overline{Legs} = \{1..(nVar - 1)\}$; and
- $\underline{Legs}$: set of legs in route excluding the first leg; $\underline{Legs} = \{2..nVar\}$.

4.6.2 Objective function of the model

We want to construct good routes to feed into the master problem. Therefore, we try to maximize the sum of the shortest paths of the loads carried in the route. To achieve this, we associate each cluster with a profit value. This profit value corresponds to the sum of the shortest paths of the loads included in the cluster. Let $x_{i,j}$ be the binary variable equal to 1 if cluster j is covered by the route on leg i and equal to 0 otherwise; and let $PROFIT_j$ be the profit associated with cluster j. Then the objective function is written as:

$$\text{maximize} \quad \sum_{j \in Clusters} \sum_{i \in Legs} x_{i,j} \times PROFIT_j$$

4.6.3 Constraints of the pricing problem

To insure that routes do not violate any feasibiltiy conditions, we aggregate constraints in different sets that are very similar to those detailed in Chapter 3

1. **Shortest path;**
2. **Domicile;**
3. **Origin and destination of a leg;**
4. **Latest arrival and earliest available;**
5. **Work time;**
6. **Break; and**
7. **General time.**

Compared to the formulation in Chapter 3, we no longer the following constraints:

- **Origin and destionation of the leading load;**
- **Choice of the leading load;**
- **Pick-up and delivery;**
- **Time in and out; and**
- **Extra time.**

Shortest path

We introduce the following set of constraints to compute the travel time on a given leg:

- $timeLeg_i \geq \sum_{k \in Arcs} arc_{i,k} \times TRAVELTIME_k - M \times \sum_{j \in CLusters} x_{i,j}, \forall i \in Legs$ *(4.1.1)*
- $timeLeg_i \geq x_{i,j} \times TIMECLUSTER_j$ *(4.1.2)*
- $\sum_{k \in Arcs} arc_{i,k} \times ARCMATRIX_{n,k} = origin_{i,n} + dest_{i,n}, \forall i \in Legs, \forall n \in Locations$ *(4.1.3)*

where $TIMECLUSTER_j$ is the time needed to cover cluster j; M equals the difference between the start and the end of the time period for our problem; $TRAVELTIME_k$ is defined as the travel time along arc k; $ARCMATRIX_{n,k}$ is equal to -1 if location n is the origin of arc k, equal to 1 if location n if the destination of arc k and equal to 0 otherwise; $origin_{i,n}$,

is a decision variable equal to -1 if the origin of leg i is location n and is equal to 0 otherwise; $dest_{i,n}$, is a decision variable equal to 1 if the destination of leg i is location n and is equal to 0 otherwise; $arc_{i,k}$ is a decision variable equal to 1 if arc k is present in the path of leg i and is equal to 0 otherwise; and $timeLeg_i$ is a decision variable equal to the total travel time on leg i.

Constraints *(4.1.1)* ensure that the travel time on leg i is greater or equal to the sum of the travel times along each arc composing the leg i corresponding to a *Tractor Only* movement. Constraints *(4.1.2)* ensure that the travel time on leg i is greater than or equal to the time needed to cover the cluster covered by leg i. Finally, constraints *(4.1.3)* are similar to the set of constraints of a network flow problem; they state that:

1. the path of leg i starts from the location designated by $origin_{i,n}$ and ends at the location designated by $dest_{i,n}$; and

2. the path of the leg is continous. If an arc is not the first or the last arc in the path, its destination has to match the origin of the succeeding arc and its destination has to match the origin of the preceding arc. If the arc is the first arc of the path, its origin has to match the origin of the leg, and if the arc is the last arc in the path, its destination has to match the destination of the leg.

Domicile Constraints

As we have stated, a route has to start and end at the domicile. We present in this section the set of constraints which will enfore that rule.

- $\sum_{i \in Domiciles} dom_i = 1$ *(4.2.1)*
- $origin_{1,n} = -\sum_{i \in Domiciles} dom_i \times DOMICILE_{n,i}, \forall n \in Locations$ *(4.2.2)*
- $dest_{nVar,n} = \sum_{i \in Domiciles} dom_i \times DOMICILE_{n,i}, \forall n \in Locations$ *(4.2.3)*
- $\sum_{j \in Clusters} x_{nVar,j} \times LOADIN_{n,j} \leq dest_{nVar,n}, \forall n \in Locations$ *(4.2.4)*
- $\sum_{j \in Clusters} x_{1,j} \times LOADOUT_{n,j} \geq origin_{1,n}, \forall n \in Locations$ *(4.2.5)*

with $DOMICILE_{n,i}$ equal to 1 if domicile i corresponds to location n and equal to 0 otherwise; $LOADOUT_{n,j}$ equal to -1 if if the origin of cluster j is location n and equal to 0 otherwise; $LOADIN_{n,j}$ equal to 1 if if the destination of cluster j is location n and equal to 0 otherwise; and dom_i is a decision variable equal to 1 if domcile i is the domicile of the route, and equal to 0 otherwise.

The first constraint *(4.2.1)* ensures that there is one and only one domicile chosen for a route. The two following groups of constraints ensure that the route starts at the domicile *(4.2.2)* and ends at the domicile *(4.2.3)*. The set of constraints *(4.2.4)* ensure that the last leg is destined to the domicile. Similarly, constraints *(4.2.5)* ensure that the first leg of the route originates from the domicile.

Origin and destination of a leg

Here, we will present constraints when combined with *(4.2.1-4.2.5)* that enforce the relationship between the clusters covered by legs and the destinations and origins of legs and ensure a continuous route with the destination of leg i matching the orgin of leg $i+1$.

- $\sum_{n \in Locations} origin_{i,n} = -1, \forall i \in Legs$ *(4.3.1)*
- $\sum_{n \in Locations} dest_{i,n} = 1, \forall i \in Legs$ *(4.3.2)*
- $\forall n \in Locations,\ dest_{i,n} + origin_{i+1,n} = 0, \forall i \in \overline{Legs}$ *(4.3.3)*

Constraints *(4.3.1)* and *(4.3.2)* state that for each leg, there is exactly one origin and exactly one destination. Constraints *(4.3.3)* ensure that the destination of leg i matches the origin of leg $i+1$.

Work Time

Total work time on a route is constrained and depends on the domicile to which the route is associated. We express this with the following constraints:

- $edpt_1 - sw \geq 0$ *(4.4.1)*

- $sw = \sum_{j \in Domiciles} dom_j \times SW_j$ *(4.4.2)*
- $fw = \sum_{j \in Domiciles} dom_j \times FW_j$ *(4.4.3)*

where SW_j is the elapsed time for briefing at domicile j; FW_j is the elapsed time for debriefing at domicile j; sw is a decision variable equal to the elapsed time before a route can depart; and fw is a decision variable equal to the elapsed time after a route ends and the driver work day ends.

Constraint *(4.4.1)* ensure that the actual time at which a route begins is positive. Moreover, contraints *(4.4.2)* and *(4.4.3)* ensure that a route elapsed times for briefing and debriefing are those associated with its domicile.

Latest arrival and earliest departure

This subsection deals with constraints related to the latest arrival time and the earliest available time for the clusters. Thus, we have:

- $edpt_i \geq \sum_{j \in Clusters} x_{i,j} \times EAVL_j, . \forall i \in Legs$ *(4.5.1)*
- $larr_i \leq \sum_{j \in Clusters} x_{i,j} \times LARR_j, \forall i \in Legs$ *(4.5.2)*
- $larr_{nVar} - edpt_1 \leq \sum_{i \in Domiciles} dom_i \times MAXDAY_i - fw - sw$ *(4.5.3)*
- $extraTime_i + timeLeg_i + \leq larr_i - edpt_i \ , \forall i \in Legs$ *(4.5.4)*

where $EAVL_j$ is the earliest available time for cluster j; $LARR_j$ is latest arrival time for cluster j; $MAXDAY_i$ is the maximum working time for a route starting at domicile i; $edpt_i$ is a decision variable equal to the earliest feasible departure time for leg i from its origin; and $larr_i$ is the decision variable equal to the latest feasible arrival time for leg i at its destination.

Contraints *(4.5.1)* ensure that the earliest departure time for the origin of leg i is not earlier than the earliest available time of the cluster covered on leg i. Similarly, contraints *(4.5.2)* ensure that the latest arrival time for leg i is no later than the latest available time of the clusters covered by leg i. Constraint *(4.5.3)* enforce that the total elapsed time of the routes

including brief and debrief times, is less than the maximum allowed. Finally, contraints *(4.5.4)* ensure that the total elapsed time of leg i is not less than the sum of the "extra" plus travel times associated with leg i.

Breaks

In what follows, we detail how we model the choice of break combination and how the breaks occur in the route.

- $firstBreak_i^1 \leq breakChoice, \forall i \in Legs$ *(4.6.1)*
- $mealBreak_i^1 \leq breakChoice, \forall i \in Legs$ *(4.6.2)*
- $secondBreak_i^1 \leq breakChoice, \forall i \in Legs$ *(4.6.3)*
- $firstBreak_i^2 \leq 1 - breakChoice, \forall i \in Legs$ *(4.6.4)*
- $mealBreak_i^2 \leq 1 - breakChoice, \forall i \in Legs$ *(4.6.5)*
- $secondBreak_i^2 \leq 1 - breakChoice, \forall i \in Legs$ *(4.6.6)*
- $\sum_{i \in Legs} firstBreak_i^1 \leq breakChoice$ *(4.6.7)*
- $\sum_{i \in Legs} mealBreak_i^1 \leq breakChoice$ *(4.6.8)*
- $\sum_{i \in Legs} secondBreak_i^1 \leq breakChoice$ *(4.6.9)*
- $\sum_{i \in Legs} firstBreak_i^2 \leq 1 - breakChoice$ *(4.6.10)*
- $\sum_{i \in Legs} mealBreak_i^2 \leq 1 - breakChoice$ *(4.6.11)*
- $\sum_{i \in Legs} secondBreak_i^2 \leq 1 - breakChoice$ *(4.6.12)*
- $M \times \sum_{j \in Legs} (firstBreak_j^1 + 1 - breakChoice) \geq edpt_i - edpt_1 - BREAK_1^1$,
  $\forall i \in Legs$ *(4.6.13)*
- $M \times \sum_{j \in Legs} (mealBreak_j^1 + 1 - breakChoice) \geq edpt_i - edpt_1 - BREAK_4^1$,
  $\forall i \in Legs$ *(4.6.14)*

- $M \times \sum_{j \in Legs} (secondBreak_j^1 + 1 - breakChoice) \geq edpt_i - edpt_1 - BREAK_7^1$
$\forall i \in Legs$ *(4.6.15)*

- $M \times \sum_{j \in Legs} (firstBreak_j^2 + breakChoice) \geq edpt_i - edpt_1 - BREAK_1^2,$
$\forall i \in Legs$ *(4.6.16)*

- $M \times \sum_{j \in Legs} (mealBreak_j^2 + breakChoice) \geq edpt_i - edpt_1 - BREAK_4^2,$
$\forall i \in Legs$ *(4.6.17)*

- $M \times \sum_{j \in Legs} (secondBreak_j^2 + breakChoice) \geq edpt_i - edpt_1 - BREAK_7^2,$
$\forall i \in Legs$ *(4.6.18)*

- $noBreak_i = 1 - aBreak_i, \forall i \in Legs$ *(4.6.19)*

- $\sum_{i \in Legs} firstBreak_i^1 \geq \sum_{i \in Legs} mealBreak_i^1$ *(4.6.20)*

- $\sum_{i \in Legs} firstBreak_i^2 \geq \sum_{i \in Legs} mealBreak_i^2$ *(4.6.21)*

- $\sum_{i \in Legs} mealBreak_i^1 \geq \sum_{i \in Legs} secondBreak_i^1$ *(4.6.22)*

- $\sum_{i \in Legs} mealBreak_i^2 \geq \sum_{i \in Legs} secondBreak_i^2$ *(4.6.23)*

where $BREAK_m^k$ describes the breaks for combination k as follows:

  - $m = 1$ equals the minimum elapsed time of the route at which the first break can occur;
  - $m = 2$ equals the maximum elapsed time of the route at which the first break can occur;
  - $m = 3$ equals the duration of the first break;
  - $m = 4$ equals the minimum elapsed time of the route at which the meal break can occur;
  - $m = 5$ equals the maximum elapsed time of the route at which the meal break can occur;

- $m = 6$ equals the duration of the meal break;
- $m = 7$ equals the minimum elapsed time of the route at which the second break can occur;
- $m = 8$ equals the maximum elapsed time of the route at which the second break can occur; and
- $m = 9$ equals the duration of the second break;

Next, $firstBreak_i^k$ is a decision variable equal to the elapsed time of the route at which the first break occurs if the break occurs during leg i and if we have chosen break combination k, and equal to 0 otherwise; $mealBreak_i^k$ is a decision variable equal to the elapsed time of the route at which the second break occurs if the break occurs during leg i and if we have chosen break combination k, and equal to 0 otherwise; $secondBreak_i^k$ is a decision variable equal to the elapsed time of the route at which the second break occurs if the break occurs during leg i and if we have chosen break combination k, and equal to 0 otherwise; $breakChoice$ is a decision variable equal to 1 if break combination 1 is chosen and equal to 2 if break combination 2 is chosen; $aBreak_i$ is a decision variable equal to 1 if there is a break occuring at leg i and equal to 0 otherwise; and $noBreak$ is a decision variable equal to 0 if there is no break occuring at leg i and equal tol otherwise.

Constraints *(4.6.1)* to *(4.6.6)* ensure that we select one combination for the break; $firstBreak_i^k$, $secondBreak_i^k$ and $thirdBreak_i^k$ can take a value different of 0 if break combination k is chosen. Constraints *(4.6.7)* to *(4.6.12)* ensure that each break is be taken at most once. Constraints *(4.6.13)* to *(4.6.18)* guarantee that breaks occur within the time windows provided in the data. Constraints *(4.6.19)* just make the connection between the two opposite indicators $noBreak_i$ and $aBreak_i$. Finally, constraints *(4.6.20)* and *(4.6.21)* allow the driver to stop for his meal if he has already taken a break. Similarly, constraints *(4.6.22)* and *(4.6.23)* allow the driver can to stop for his second break if he has already had his meal.

Time

In this section, we introduce the constraints linking break times and earliest departure and latest arrival times.

- $extraTime_i + timeLeg_i + firstBreak_i^1 \times BREAK_3^1 + mealBreak_i^1 \times BREAK_6^1$
$+secondBreak_i^1 \times BREAK_9^1 \leq larr_i - larr_{i-1}.\ \forall i \in \underline{Steps}$ *(4.7.1)*

- $extraTime_i + timeLeg_i + firstBreak_i^2 \times BREAK_3^2 + mealBreak_i^2 \times BREAK_6^2$
$+secondBreak_i^2 \times BREAK_9^2 \leq larr_i - larr_{i-1}, \forall i \in \underline{Steps}$ *(4.7.2)*

- $extraTime_{i-1} + timeLeg_{i-1} + firstBreak_{i-1}^1 \times BREAK_3^1 + mealBreak_{i-1}^1 \times BREAK_6^1$
$+secondBreak_{i-1}^1 \times BREAK_9^1 \leq edpt_i - edpt_{i-1}, \forall i \in \underline{Steps}$ *(4.7.3)*

- $extraTime_{i-1} + timeLeg_{i-1} + firstBreak_{i-1}^2 \times BREAK_3^2 + mealBreak_{i-1}^2 \times BREAK_6^2$
$+secondBreak_{i-1}^2 \times BREAK_9^2 \leq edpt_i - edpt_{i-1}, \forall i \in \underline{Steps}$ *(4.7.4)*

- $larr_i \geq firstBreak_i^1 \times (BREAK_1^1 + BREAK_3^1) + edpt_1, \forall i \in \overline{Legs}$ *(4.7.5)*

- $larr_i \geq firstBreak_i^2 \times (BREAK_1^2 + BREAK_3^2) + edpt_1, \forall i \in \overline{Legs}$ *(4.7.6)*

- $larr_i \geq mealBreak_i^1 \times (BREAK_4^1 + BREAK_6^1) + edpt_1, \forall i \in \overline{Legs}$ *(4.7.7)*

- $larr_i \geq mealBreak_i^2 \times (BREAK_4^2 + BREAK_6^2) + edpt_1, \forall i \in \overline{Legs}$ *(4.7.8)*

- $larr_i \geq secondBreak_i^1 \times (BREAK_7^2 + BREAK_9^2) + edpt_1, \forall i \in \overline{Legs}$ *(4.7.9)*

- $larr_i \geq secondBreak_i^2 \times (BREAK_7^2 + BREAK_9^2) + edpt_1, \forall i \in \overline{Legs},$ *(4.7.10)*

- $edpt_i \leq firstBreak_i^1 \times BREAK_2^1 + edpt_1 + (1 - firstBreak_i^1) \times M, \forall i \in Legs$ *(4.7.11)*

- $edpt_i \leq firstBreak_i^2 \times BREAK_2^2 + edpt_1 + (1 - firstBreak_i^2) \times M, \forall i \in Legs$ *(4.7.12)*

- $edpt_i \leq mealBreak_i^1 \times BREAK_5^1 + edpt_1 + (1 - mealBreak_i^1) \times M, \forall i \in Legs$ *(4.7.13)*

- $edpt_i \leq mealBreak_i^2 \times BREAK_5^2 + edpt_1 + (1 - mealBreak_i^2) \times M, \forall i \in Legs$ *(4.7.14)*

- $edpt_i \leq secondBreak_i^1 \times BREAK_8^1 + edpt_1 + (1 - secondBreak_i^1) \times M,$
$\forall i \in Legs$ *(4.7.15)*

- $edpt_i \leq secondBreak_i^2 \times BREAK_8^2 + edpt_1 + (1 - secondBreak_i^2) \times M,$
$\forall i \in Legs,$ *(4.7.16)*

Constraints *(4.7.1)* and *(4.7.2)* ensure that the time differences between two consecutive latest arrival times should be greater than the sum of the time required by the different events (extra time, travel time and break time) occuring on the leg between these arrivals. Similarly, constraints *(4.7.3)* and *(4.7.4)* ensure that a leg does not depart before its predecessor can arrive. Constraints *(4.7.5)* to *(4.7.10)* ensure that if a break occurs on leg i is at least as great as the earliest ime for starting the break plus the duration of the break. Similarly, constraints *(4.7.11)* to *(4.7.16)* ensure that if a break occurs on leg i then the earliest departure time is not later than the latest time for the break to occur.

4.7 Heuristic to generate routes

We begin by generating as many routes as there are clusters. To achieve this, at each iteration j involving the generation of a route containing cluster j, we add to the pricing problem the following constraint:

$$\sum_{i \in Legs} x_{ij} \geq 1$$

These constraints ensure that all clusters (hence all loads) are present in the set of routes generated. This is a condition for the master problem developed in Chapter 3 to be solvable. We do not iterate between the master and the pricing problem however we can improve the solution the solution by generating more routes.

As for high values of $nVar$, the pricing problem experiences difficulty in finding an optimal solution to the problem, we stop the program after a specified time by setting a time limit within CPLEX. We do not reach the optimal solution but we produce a valid route. We can adjust two parameters to generate routes: the number of legs in the routes and the time limit. For a given time limit and each value of $nVar$, we generate $|Cluster|$ routes.

In the table below, we provide some of our computational experience in using the cluster approach. We started generating routes, we had 219 clusters and set the parameters $nVar$ to 5 and the time limit to 100 seconds. The other loads are generated by changing the values of $nVar$ and of the time limit.

Routes Generated	Number of vehicles required
219	112
438	109
1095	105
2190	102

Table 4.8: Results of the heuristic for the master problem

As seen through these figures, the rate of improvement of the solution decreases quickly.

Chapter 5

Conclusions and future research directions

5.1 Summary

In this thesis research, we have developed two approaches for tackling the capacity constrained pick-up and delivery problem with time windows. Our first approach is a decomposition approach that iterates between solving a master and a pricing problem. This approach, while optimal, is hampered by the large sets of contraints and variables involved. Our second approach is a sequential approach. We simplify the structure of the pricing problem to make it more tractable, but by doing so we are no longer able to guarantee optimality of the solution. We have implemented these two approaches. Further details about these implementations are given in the Appendix A.

5.2 Future research

Much of the complexity of the problem we address comes from its numerous requirements, many of which do not derive from absolute requirements: they are rather the consequences of the way the company is currently managing its operations. If we consider the time requirements for

pick-ups and deliveries, it is clear that these requirements depend on other operations which occur before and after the load is transported. For example, if a load is brought to a sorting facility, its delivery time has to be consistent with the schedule of the facility. Future research should focus on how to adapt such requirements to the routes we generate instead of building routes to satisfy these requirements.

References

[1] Barnhart, Cohn, Johnson, Nemhauser and Shenoi (1998), "Airline Crew Scheduling", *Handbook of Transportation Science.*

[2] Barnhart, Rexing, Kniker, Jarrah, Krishnamurtgy (2000), "Airline Fleet Assignment with TimeWindows", *Transportation Science.*

[3] Bertsimas, Tsitsiklis (1997) , "Introduction to Linear Optimization", Athena Scientific.

[4] Cordeau,.Gendreau, Laporte, Potvin, Semet (2002), "A guide to vehicle routing heuristics", *Journal of the Operational Research Society.*

[5] Cordeau, Laporte, Mercier (2001), "A unified tabu search heuristic for vehicle routing problems with time windows", *Journal of the Operational Research Society.*

[6] Desroches, Desrosiers, Solomon (1992), "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows", *Operations Research.*

[7] Desroches, Desrosiers, Solomon (1994), "A Tabu Search Heuristic for the Vehicle Routing Problem", *Management Science.*

[8] Laporte (1992), "The Traveling Salesman Problem: An overview of exact and pproximate algorithms", *European Journal of Operational Research.*

[9] Laporte (1992), "The Vehicle Routing Problem: An overview of exact and pproximate algorithms", *European Journal of Operational Research.*

[10] Larson, Odoni (1981), "Urban Operations Researc", Prentice-Hall.

[11] Ralphs, Kopman, Pulleyblank, Trotter Jr (2003), "On the Capacitated Vehicle Routing Problem", *Mathematical Programming Series.*

[12] Solomon (1994), "Algorithm for the Vehicle Routing Problem and Scheduling Problems with TIme WIndows Contraints", *Management Science.*

[13] VanHentenryck (1999), "The OPL Optimization Programming Language", The MIT Press.

Appendix A

Computational experiments

A.1 Implementation of the algorithms

We have implemented the column generation and cluster algorithms in Java and OPL. The models are programmed in OPL but the central part of the implementation is an application developed in Java. The Java application has basically three functions:

1. **Export and Import to/from Excel** with JXL library because the data files and schedules are in Excel format;
2. **Generation of scripts and model files** to be fed into OPLStudio; and
3. **Control of OPLStudio** through OPLJNI library.

Figure A-1 describes the structure of the application.

A.2 Checking conformity of schedules

Because schedules are subject to many requirements, we have developed a tool to check that the schedules generated do not violate any constraints. This tool also prices schedules and compute statistics on the the number of hours worked, the rate of utilization of drivers...

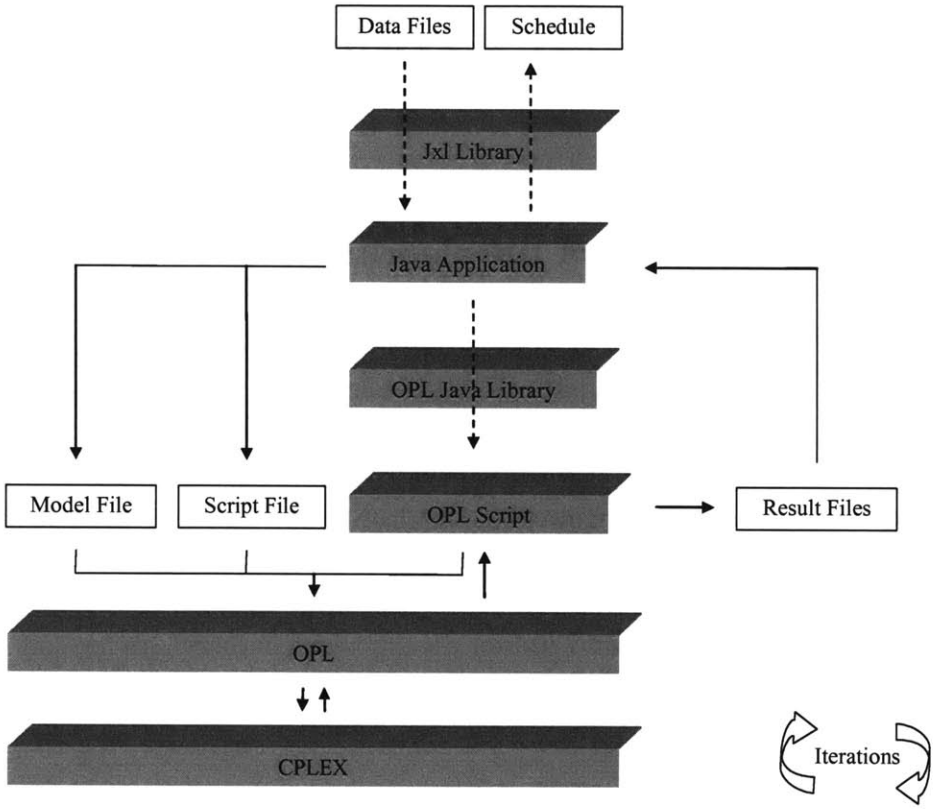
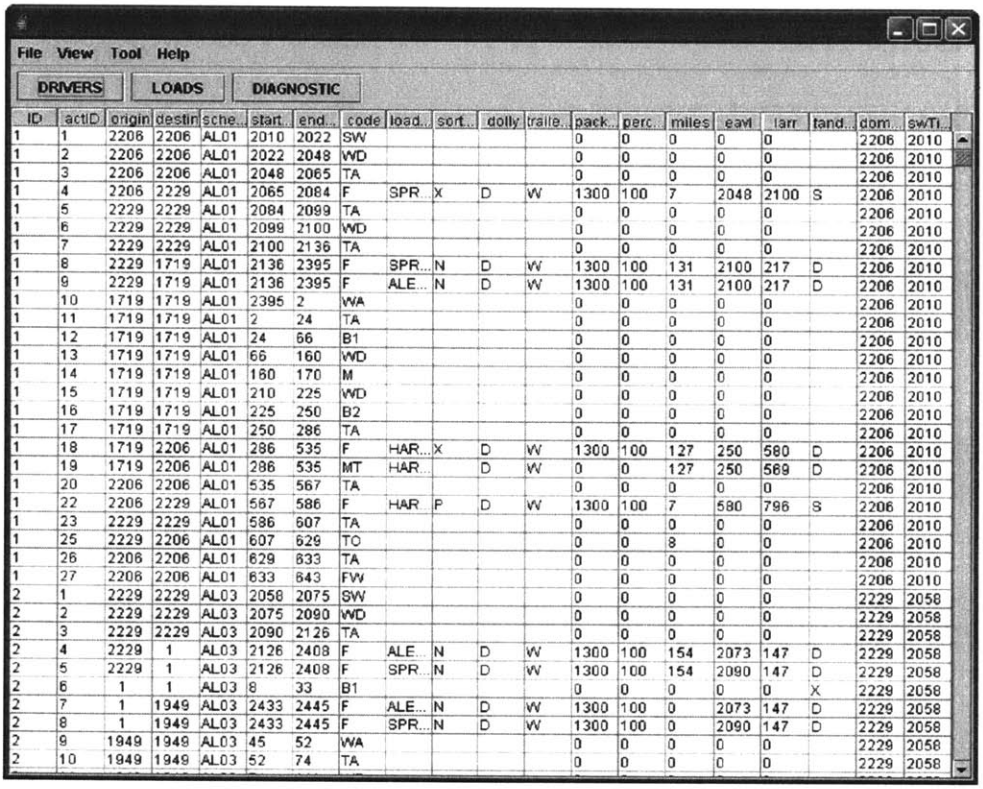


Figure A-1: General structure of the Java application



ID	actID	origin	destin	sche...	start...	end...	code	load...	sort...	dolly	traile...	pack...	perc...	miles	eavl	larr	tand...	dom...	swTi...
1	1	2206	2206	AL01	2010	2022	SW					0	0	0	0	0		2206	2010
1	2	2206	2206	AL01	2022	2048	WD					0	0	0	0	0		2206	2010
1	3	2206	2206	AL01	2048	2065	TA					0	0	0	0	0		2206	2010
1	4	2206	2229	AL01	2065	2084	F	SPR...	X	D	W	1300	100	7	2048	2100	S	2206	2010
1	5	2229	2229	AL01	2084	2099	TA					0	0	0	0	0		2206	2010
1	6	2229	2229	AL01	2099	2100	WD					0	0	0	0	0		2206	2010
1	7	2229	2229	AL01	2100	2136	TA					0	0	0	0	0		2206	2010
1	8	2229	1719	AL01	2136	2395	F	SPR...	N	D	W	1300	100	131	2100	217	D	2206	2010
1	9	2229	1719	AL01	2136	2395	F	ALE...	N	D	W	1300	100	131	2100	217	D	2206	2010
1	10	1719	1719	AL01	2395	2	WA					0	0	0	0	0		2206	2010
1	11	1719	1719	AL01	2	24	TA					0	0	0	0	0		2206	2010
1	12	1719	1719	AL01	24	66	B1					0	0	0	0	0		2206	2010
1	13	1719	1719	AL01	66	160	WD					0	0	0	0	0		2206	2010
1	14	1719	1719	AL01	160	170	M					0	0	0	0	0		2206	2010
1	15	1719	1719	AL01	210	225	WD					0	0	0	0	0		2206	2010
1	16	1719	1719	AL01	225	250	B2					0	0	0	0	0		2206	2010
1	17	1719	1719	AL01	250	286	TA					0	0	0	0	0		2206	2010
1	18	1719	2206	AL01	286	535	F	HAR...	X	D	W	1300	100	127	250	580	D	2206	2010
1	19	1719	2206	AL01	286	535	MT	HAR...		D	W	0	0	127	250	589	D	2206	2010
1	20	2206	2206	AL01	535	567	TA					0	0	0	0	0		2206	2010
1	22	2206	2229	AL01	567	586	F	HAR...	P	D	W	1300	100	7	580	796	S	2206	2010
1	23	2229	2229	AL01	586	607	TA					0	0	0	0	0		2206	2010
1	25	2229	2206	AL01	607	629	TO					0	0	8	0	0		2206	2010
1	26	2206	2206	AL01	629	633	TA					0	0	0	0	0		2206	2010
1	27	2206	2206	AL01	633	643	FW					0	0	0	0	0		2206	2010
2	1	2229	2229	AL03	2058	2075	SW					0	0	0	0	0		2229	2058
2	2	2229	2229	AL03	2075	2090	WD					0	0	0	0	0		2229	2058
2	3	2229	2229	AL03	2090	2126	TA					0	0	0	0	0		2229	2058
2	4	2229	1	AL03	2126	2408	F	ALE...	N	D	W	1300	100	154	2073	147	D	2229	2058
2	5	2229	1	AL03	2126	2408	F	SPR...	N	D	W	1300	100	154	2090	147	D	2229	2058
2	6	1	1	AL03	8	33	B1					0	0	0	0	0	X	2229	2058
2	7	1	1949	AL03	2433	2445	F	ALE...	N	D	W	1300	100	0	2073	147	D	2229	2058
2	8	1	1949	AL03	2433	2445	F	SPR...	N	D	W	1300	100	0	2090	147	D	2229	2058
2	9	1949	1949	AL03	45	52	WA					0	0	0	0	0		2229	2058
2	10	1949	1949	AL03	52	74	TA					0	0	0	0	0		2229	2058

Figure A-2: Checking validity of schedules (screenshot 1)

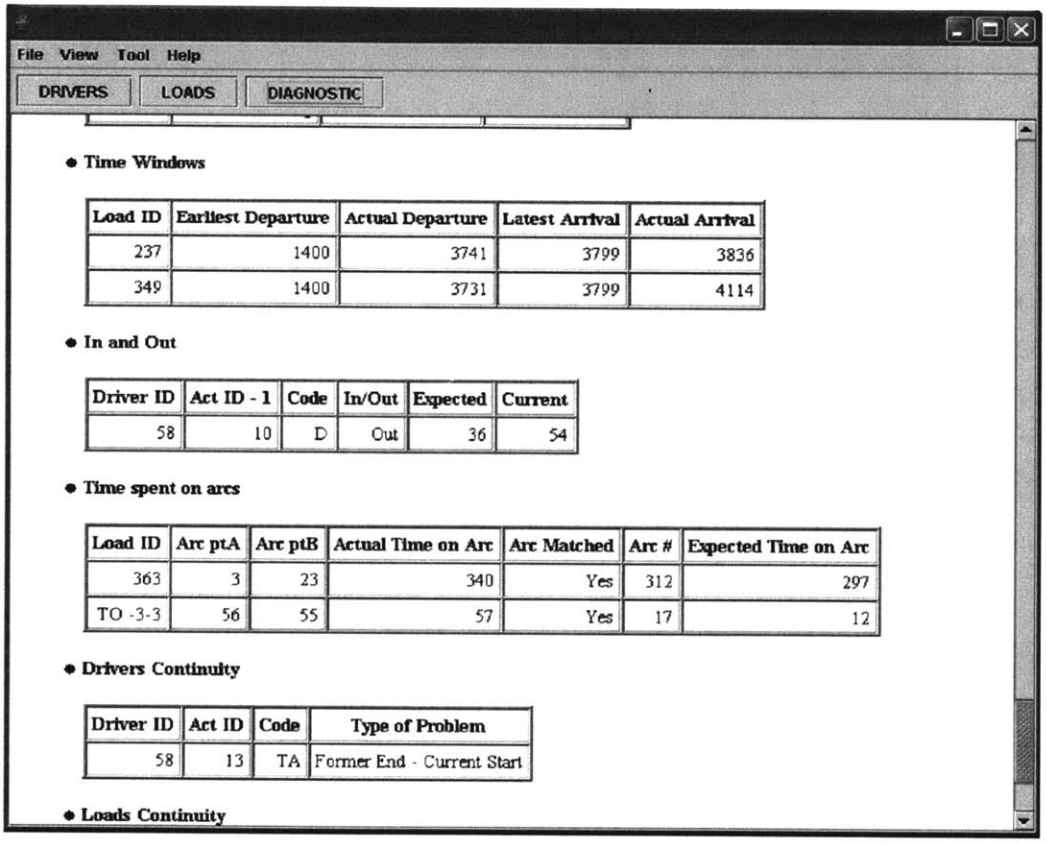


Figure A-3: Checking validity of schedules (screenshot 2)