BSc thesis APPLIED MATHEMATICS

“Reduction of the amount of yield data for KLM using set cover”

EMMA DE WIT

Delft University of Technology

Supervisors
Prof. dr.ir K.I. Aardal    Dr. G.A.P. Kindervater

Rest of committee
Dr. J.G. Spandaw    Dr. ir. F.J. Vermolen

January, 2012    Delft
Abstract

In the Revenue Management department of KLM the prices for tickets are determined. The information of the prices are sent to a reservation system, in the form of yield data points. A reservation system is an external company that takes care of the selling of the tickets to the consumers for multiple airlines. The reservation system sets a restriction of 100 million yield data points.

Because of the merger of KLM and Air France the Revenue Management systems are combined. Both companies generate about 70 million yield data points and therefore the restriction will be exceeded when the departments are combined.

In my project I looked at the possibilities of reducing the amount of yield data in order to satisfy to the restrictions of the reservation system. This problem can be seen as the mathematical set cover problem, which is an NP-hard optimization problem. With the use of two solution methods, I found some surprising results to reduce the amount of yield data.

Figure 1: Communication between KLM, the reservation system and the consumers
# Contents

1 Introduction 4

2 Background Information about Revenue Management and Reservation Systems 5
   2.1 Revenue Management and Pricing 6
   2.2 Communication between Revenue Management and Altea 9
      2.2.1 Passenger Net Value 10
      2.2.2 Closures 10
      2.2.3 Bid Prices 11
      2.2.4 Acceptance of a Passenger 11
   2.3 Sending Data to Altea 12
      2.3.1 Sending a PNV to Altea 12
      2.3.2 Example: Reduction of PNVs to Altea 13
      2.3.3 Sending a Closure to Altea 14
      2.3.4 Example: Reduction of Closures to Altea 15

3 Problem Definition and Approach 16

4 Set Cover Problem 17
   4.1 The Set Cover Problem 17
   4.2 Applications of the Set Cover Problem 18
   4.3 Characteristics of the Set Cover Problem 18
   4.4 Solution Methods 18
      4.4.1 Greedy Algorithm 19
      4.4.2 CPLEX 20

5 Reduction of Closures 23
   5.1 Mathematical Formulation 23
   5.2 The Solution and Programming Part 27
      5.2.1 Step 1: A recursion method is used to generate feasible subsets 28
      5.2.2 Step 2: The problem is solved 30
   5.3 Results 31

6 Reduction of PNVs 32
   6.1 Mathematical Formulation 32
   6.2 The Solving and Programming Part 35
      6.2.1 Step 1: A recursion method is used to generate feasible subsets 35
      6.2.2 Step 2: The problem is solved 36
   6.3 Results 37

7 Conclusions 38

8 Recommendations 39
1 Introduction

The Revenue Management departments at KLM and Air France determine the prices of their tickets. A new software program is built, where the both Revenue Management departments are combined. The objective is to maximize overall revenue of the companies. After the prices of the tickets are generated by KLM and Air France, the prices are sent to a reservation system. This reservation system is handled by an external independent company (Amadeus) that takes care of the direct contact with the passenger and of the selling of the tickets.

A huge amount of the yield data is sent to this reservation system, about 70 million “data points” for each company. The data contains information about the price, the origin and destination of the flight and the day of departure. The data for KLM and Air France will both be included in the new software. The amount of data will therefore be doubled, and exceeds the restrictions of the reservation system. In my project I look at the possibility of reducing the data where my main goal is to:

Reduce the amount of yield data sent to the reservation system in order to satisfy the restrictions of the reservation system

In this thesis I will start with some background information about revenue management at KLM in Chapter 2. Next, in Chapter 3 I will give a more detailed problem definition, together with the goals for the project. In order to satisfy the restriction of Altea, the amount of yield data has to be reduced with about 30%.

The reduction of the yield data will be approached as the mathematical set cover problem. Chapter 4 gives some information about the set cover problem and how such problems can be approached.

In Chapters 5 and 6 the problem is modeled mathematically and we find good solutions using optimization algorithms, where the amount of yield data is eventually reduced with about 90%. In Chapter 7 I will evaluate the results of the project. Finally some recommendations will be made to KLM with respect to future changes.
2 Background Information about Revenue Management and Reservation Systems

The Revenue Management department at KLM determines the prices for the tickets of the flights and the number of seats it wants to sell at what price. From there on this yield data is sent to a reservation system, that handles the selling of tickets. Based on the data set of the airline, the reservation system sends information about the prices of the tickets from the airline(s) upon request by travel agencies and websites. The travel agencies and websites sell the tickets to the consumers. The reservation system also sends up-to-date information about the selling of the tickets back to KLM. This makes it possible to adjust the settings when the actual demand is lower/higher than expected.

The reservation system used by KLM is Altea, owned by the company Amadeus. Altea does the bookings for many other airlines, such as Lufthansa and British Airways. Amadeus also owns reservation systems for train travel, ferry reservations and hotel bookings and is thereby the worldwide market leader in terms of the company with the most bookings per year.

The figure below shows how the airlines, Altea and the consumers communicate with each other:

![Communication between KLM, Altea and the consumers](image)

As we can see in the figure above, in the current situation the Revenue Management Systems of KLM and Air France are separate. In the new software these two Revenue Management Systems will be joint.

In the rest of this chapter first some more information about revenue management is given, then we explain how the communication with the reservation system works and finally we describe how the prices are sent to the reservation system.
2.1 Revenue Management and Pricing

The Revenue Management department at KLM determines the prices for the tickets and the number of seats it wants to sell at what price. The key aim is to maximize the total revenue for all seats on the network. Revenue management can be divided into three stages: Pricing, Inventory Management and Revenue Integrity. The figure below shows how these stages influence each other in order to maximize the total income of KLM.

In the stage where the prices are determined, about 25 different pricing classes are used. Fare and product conditions distinguish the different pricing classes. A well-known distinction between two fare conditions are the economy and business class, where the passengers are seated in separate cabins, each with their own service level. Examples of product conditions are the possibilities of canceling or re-booking a flight when you buy a ticket in a higher pricing class.
In the figure below an example is shown with 15 different pricing classes and 2 cabins. We see that there are three business pricing classes and twelve economy pricing classes.

![Figure 4: Different pricing classes](image)

In the following example the use of the different pricing classes is explained. We look at a simple representation of the demand against the number of seats that are expected to be sold.

![Figure 5: Maximizing the total income with the use of different pricing classes](image)

If just one fare (one pricing class) is introduced, then it is expected that not all the seats are sold and that there are passengers that are willing to pay more for their ticket. If we use segmentation, the total revenue becomes higher and more seats are expected to be sold.

The segments represent different pricing classes with a different fare. KLM computes a net value, which is the price that the passenger pays for his ticket minus the taxes at the airport, administration costs etc. This net value is called the Passenger Net Value (PNV). For each route the PNVs are determined by looking at the sales of the previous year. The determination of the prices is done one year ahead. The values of the PNVs are changed regularly, due to currency fluctuations and user overrules.
How do we get a passenger to pay the maximum value for his ticket?

We would like a passenger to pay the price for the ticket that he is willing to pay and thereby imitate figure 5. Two strategies are used:

- Fare and product conditions (as explained above)
- Closures

A Closure is used to stop accepting passengers in a low pricing class and to force a consumer to buy a ticket in higher pricing classes. Two forms of Closures are used by KLM: Delay Out Closures and Seats Sold Closures. A Delay Out Closure closes a route in a pricing class a number of days before departure and a Seats Sold Closure closes a pricing class when a given number of seats is already sold in that pricing class.

The Closures change on a daily basis, because they depend on the number of seats that are sold and on a number of days before departure. Therefore at least once a day a new update of the Closures are sent to Altea.

We have seen that especially for the Closures, daily updates to the reservation system require efficient determination of a data set which meets the restrictions of the reservation system. The optimization procedure must be able to produce results within reasonable time to meet the demand of a daily update.
2.2 Communication between Revenue Management and Altea

After the PNVs and the Closures are determined by the department of revenue management the information is sent to Altea, together with a Bid Price (to be explained below). With this information Altea can determine whether or not to accept a passenger when he wants to buy a ticket:

![Diagram of communication between Revenue Management and Altea](image)

Figure 6: PNVs, Closures and Bid Prices are sent to Altea

In the figure we can see that the PNVs and the Closures that are sent to Altea are from an origin city to a destination city (O&D). This is called a route. The Bid Price is a minimum price that a passenger has to pay and is determined for a leg of the route by an optimization algorithm. A leg is the trip that the airplane makes between two stops. We look at an example where a passenger flies from Amsterdam to New York with a stopover in Paris. This flight has two legs and one route:

![Diagram of a route and legs](image)

Figure 7: Difference between a leg and a route

To understand how Altea determines when to accept a passenger, first some more information is given about PNVs, Closures and Bid Prices.
2.2.1 Passenger Net Value

A Passenger Net Value (PNV) is the value that KLM sets for a ticket and is the value that KLM receives when a passenger buys a ticket. In a record associated with each PNV the following information is specified:

- The value of the PNV
- The origin city of the route
- The destination city of the route
- Pricing Class
- Point of Sale (The place were the ticket is bought by the consumer)
- The date of the flight(s)

For example:
The PNV for a ticket from Amsterdam to London, in pricing class J, when the ticket is bought in the Netherlands, is €418,00 for tickets with departure date October 1, 2011.

2.2.2 Closures

Closures are used to close the lower pricing classes and force consumers to buy tickets in higher pricing classes. Again, a Closure is specified for:

- The origin city of the route
- The destination city of the route
- Pricing Class
- Point of Sale
- Route
- The date of the flight(s)

A Closure is specified for a certain “route”, which could be one flight number. In certain situations it is a bit more complicated than adding just one flight number, but for the sake of clarity we decided not to go into detail on all exceptions. An example will be given in 2.3.3.

For example:
The tickets from Amsterdam to London using the flight KL1000, in pricing class T, when the ticket is bought in the Netherlands, are closed for tickets with departure date October 1, 2011.
2.2.3 Bid Prices

A Bid Price is the minimum price that a passenger has to pay for a leg of the flight in a certain cabin. A Bid Price is specified for:

- A Bid Price
- A leg of a route that KLM/Air France flies on
- Cabin
- The date of the flight(s) (on the legs of the route)

2.2.4 Acceptance of a Passenger

Three questions must be answered to determine whether a passenger is accepted in a certain pricing class:

1. Is the value of the PNV of the route from origin to destination higher than the sum of the Bid Prices of the all the legs on the route?
   \[
   \text{PNV} \geq \sum \text{Bid Prices (of the appropriate cabins)}
   \]

2. Is the route open? A route is closed when a Closure is set on that route, otherwise the route is open.

3. Are there available seats on all legs of the route?

When all answers to the questions are positive, a ticket can be sold in the specific pricing class for the value of the PNV. Otherwise tickets in a higher pricing class are checked.
2.3 Sending Data to Altea

In this part of the thesis we describe how the yield data is sent to Altea for both PNVs and Closures and some examples are shown of how reduction of the data points is possible.

2.3.1 Sending a PNV to Altea

We introduce the format that represents a data point for a PNV that is sent to Altea, where we take the specifications of Altea into consideration:

<table>
<thead>
<tr>
<th>Value PNV</th>
<th>Origin City</th>
<th>Destination City</th>
<th>Pricing Class</th>
<th>Points of Sale</th>
<th>Period</th>
<th>Frequency</th>
</tr>
</thead>
</table>

For the value of the PNV, the origin city, the destination city and the pricing class, only a single value can be added per data point. For the points of sale, the period and the frequency, there are more possibilities:

- **Points of sale:**
  A set of points of sale can be filled in. This makes it possible to group points of sale for which the values, origin and destination city and the pricing class are the same.

- **Period:**
  For the period it is possible to add a period of time for which the value of the PNV, the origin city and the pricing class are the same. For instance for the tickets with departure date from October 1 until October 31.

- **Frequency:**
  The frequency consist of seven 0/1-digits which represent the days of the week on which the data point is specified. For example, the frequency 1010000 stands for all Mondays and Wednesdays. Adding a frequency makes it possible to combine data points when the value, the origin and destination city and pricing class are the same for one or more days of the week.

In the beginning of this chapter an example was given for a PNV:

*The PNV for a ticket from Amsterdam to London, in pricing class J, when the ticket is bought in the Netherlands, is €418,00 for tickets with departure date October 1, 2011.*

This PNV can be translated to one data point that can be sent to the reservation system:

| €418 | Amsterdam | London | J | The Netherlands | from 01-10-11 until 01-10-11 | 0000010 |

We can see that the frequency for this data point is 0000010, this is because 1st October 2011 falls on a Saturday.
2.3.2 Example: Reduction of PNVs to Altea

As discussed above: When two or more data points are the same, except for the point of sale, it is possible to put these two data points together. Take for example the next three data points, where we use NL for the Netherlands, UK for the United Kingdom and US for the United States as different points of sale.

Data point 1: Point of Sale The Netherlands

\[ \begin{array}{|c|c|c|c|c|} 
\hline 
€418 & Amsterdam & London & J & NL \\
\hline 
\text{from 01-10-11 until 01-10-11} & 0000010 \\
\hline 
\end{array} \]

Data point 2: Point of Sale The United Kingdom

\[ \begin{array}{|c|c|c|c|c|} 
\hline 
€418 & Amsterdam & London & J & UK \\
\hline 
\text{from 01-10-11 until 01-10-11} & 0000010 \\
\hline 
\end{array} \]

Data point 3: Point of Sale The United States

\[ \begin{array}{|c|c|c|c|c|} 
\hline 
€418 & Amsterdam & London & J & US \\
\hline 
\text{from 01-10-11 until 01-10-11} & 0000010 \\
\hline 
\end{array} \]

Because these data points only differ in the points of sale, it is possible combine them into one data point:

\[ \begin{array}{|c|c|c|c|c|} 
\hline 
€418 & Amsterdam & London & J & NL, UK, US \\
\hline 
\text{from 01-10-11 until 01-10-11} & 0000010 \\
\hline 
\end{array} \]

Combining data points with the same points of sale is not used by KLM very often at this moment, but could reduce the amount of data considerably.

In a same way we can reduce data by combining data points with the same period or frequency. An example is as follows. Suppose that the PNVs for the route from Amsterdam to London, in pricing class J, have the same value of €418,00 at the 1st of October, 2nd of October and 4th of October. This gives us the next three data points:

Data point 1: Date 01-10-11

\[ \begin{array}{|c|c|c|c|c|} 
\hline 
€418 & Amsterdam & London & J & NL \\
\hline 
\text{from 01-10-11 until 01-10-11} & 0000010 \\
\hline 
\end{array} \]

Data point 2: Date 02-10-11

\[ \begin{array}{|c|c|c|c|c|} 
\hline 
€418 & Amsterdam & London & J & NL \\
\hline 
\text{from 02-10-11 until 02-10-11} & 0000001 \\
\hline 
\end{array} \]

Data point 3: Date 04-10-11

\[ \begin{array}{|c|c|c|c|c|} 
\hline 
€418 & Amsterdam & London & J & NL \\
\hline 
\text{from 04-10-11 until 04-10-11} & 0100000 \\
\hline 
\end{array} \]

The above three data points can be combined by merging the frequencies and the period of time.

\[ \begin{array}{|c|c|c|c|c|} 
\hline 
€418 & Amsterdam & London & J & NL \\
\hline 
\text{from 01-10-11 until 04-10-11} & 0100011 \\
\hline 
\end{array} \]

Again, when such data points are the same for more than one point of sale, it is possible to combine them as well. One then obtains:

\[ \begin{array}{|c|c|c|c|c|} 
\hline 
€418 & Amsterdam & London & J & NL, UK, US \\
\hline 
\text{from 01-10-11 until 04-10-11} & 0100011 \\
\hline 
\end{array} \]

13
2.3.3 Sending a Closure to Altea

For a Closure we also introduce the format of one data point to Altea:

<table>
<thead>
<tr>
<th>Closure</th>
<th>Origin City</th>
<th>Destination City</th>
<th>Route</th>
<th>Pricing Class</th>
<th>Points of Sale</th>
<th>Period</th>
<th>Frequency</th>
</tr>
</thead>
</table>

Most of the items that need to be filled in, are the same for the data points for PNVs and Closures. There are two differences:

- Instead of sending a value of the PNV, a *Closure Flag* is sent. A Closure Flag means that it is not possible to buy tickets for that specific flight in the given pricing class.
- The PNVs are determined for all flights from origin to destination city for certain dates, whereas the Closure is set on a route, e.g. a flight number.

The example at the beginning of the chapter was:

*The tickets from Amsterdam to London on the flight KL1000, in pricing class T, when the ticket is bought in the Netherlands, are closed for tickets with departure date October 1, 2011.*

This can be translated into:

| Closed | Amsterdam | London | KL1000 | T | NL | from 01-10-11 until 01-10-11 | 0000010 |
2.3.4 Example: Reduction of Closures to Altea

In the same way as for the PNVs, it is possible to merge data points. We look at the next three data points:

Datapoint 1: Point of Sale The Netherlands

| Closed | Amsterdam | London | KL1000 | T | NL | from 01-10-11 until 01-10-11 | 0000010 |

Data point 2: Point of Sale The United Kingdom

| Closed | Amsterdam | London | KL1000 | T | UK | from 01-10-11 until 01-10-11 | 0000010 |

Data point 3: Point of Sale The United States

| Closed | Amsterdam | London | KL1000 | T | US | from 01-10-11 until 01-10-11 | 0000010 |

The above three data points for Closures can now be put together as:

| Closed | Amsterdam | London | KL1000 | T | NL, UK, US | from 01-10-11 until 01-10-11 | 0000010 |

This works the same way for the frequency and the period of time.

The Closures differ from the PNVs because they are set on the route from Amsterdam to London. In our example we have seen that for the route a flight number was filled in. It is also possible to look at on all flights of the route. In this case not one flight number is filled in, but KLANY is used to denote all the flights, for example:

| Closed | Amsterdam | London | KLANY | T | NL | from 01-10-11 until 01-10-11 | 0000010 |
3 Problem Definition and Approach

In the Introduction I described my main goal for the project as:

*Reduce the amount of yield data sent to the reservation system in order to satisfy the restrictions of the reservation system*

Altea sets a restriction of the amount of data at 100 million yield data points that can be sent to the reservation at once. The yield data points are the data points of the Closures and PNVs together. At the moment both KLM and Air France send about 70 million yield data points for their network. When the networks are combined the number of routes will increase significantly and will thereby exceed the 100 million. We would like to reduce the amount of data by at least 30%.

In order to reduce the amount of yield data that is sent to Altea, we consider the following subproblems:

1. Reduce the amount of Closures sent to the reservation system
2. Reduce the amount of PNVs sent to the reservation system

The problem can be seen as a mathematical *set cover problem*. In Chapter 4 we discuss the set cover problem. In Chapter 5 we start with the first subproblem, and try to reduce the amount of Closures as much as possible. Because this problem only looks at the possibility of a route being open or closed for sale, this is the easier problem to start with. In Chapter 5 we first describe and evaluate the mathematical approach. We then shortly discuss an approach to solving it. A couple of tests are done and the results are shown in the last section of the chapter. For the second subproblem the same outline is followed in Chapter 6.


4 Set Cover Problem

In this chapter more information is given about the set cover problem, because our problem is a specific form of such a problem. First the structure of a set cover problem is described. Another way of modeling the problem is as the set partition problem. The set partition problem will be described as well, and the two problems will be compared. Then the applications and the characteristics of the problem are looked at, and at the end of the chapter some ways of approaching the problem are described. In Chapters 5 and 6 of the thesis two ways of determining solutions to the problem are described: the Greedy algorithm, which is a heuristic, and with the use of the program CPLEX, which contains an implementation of the optimization algorithm Branch-and-Bound. Therefore some extra attention will be given to these two solution methods at the end of this chapter.

4.1 The Set Cover Problem

The set cover problem is defined as follows:

Given a universe $U$ and a family $S$ of subsets $S_i \ (i = 1, \ldots, K)$ of $U$, a cover is a subfamily $C \subseteq S$ of sets whose union is $U$. In the basic (minimum cardinality) set cover optimization problem, the task is to find a cover that uses the fewest subsets with input $(U, S)$.

In the minimum weighted set cover problem, each subset $S_i$ is given a weight/cost $c(S_i)$ and the goal is to determine a set cover with minimum total weight. This version of the problem can be formulated as the following integer linear problem:

Let

$$x_{S_i} = \begin{cases} 1 & \text{if } S_i \in C \\ 0 & \text{otherwise} \end{cases}$$

$$z_{OPT} = \text{minimize } \sum_{S_i \in S} c(S_i) \cdot x_{S_i} \quad (1)$$

subject to

$$\sum_{S_i \subseteq S} x_{S_i} \geq 1 \quad \text{for all } e \in U \quad (2)$$

$$x_{S_i} \in \{0, 1\} \quad \text{for all } S_i \in S \quad (3)$$

In the case of minimum cardinality set cover, $c(S_i)$ is equal to one for each $S_i$.

The objective function (1) minimizes the total costs and constraint (2) ensures that each element in $U$ is covered. Constraint (3) requires that a set $S_i \in S$ can either be in the cover $C$ or not. This makes the set cover problem an integer problem, and more specifically, a binary integer optimization problem.

For example, when:

$U = \{1, 2, 3, 4, 5\}$
$S = \{\{1, 2, 3\}, \{2, 3\}, \{3, 4\}, \{3, 4, 5\}\}$

when $c(S_i) = 1$ for all $S_i \in S$, then
\( C = \{\{1, 2, 3\}, \{3, 4, 5\}\} \) is an optimal solution

The above problem describes the set cover problem, but when the inequality constraint (2) is replaced by an equality sign then the problem is called a set partition problem. We can see that in our example the set partition problem has no feasible solution, because element \{3\} is in each subset and we need at least two subsets to cover the universe. Therefore it is not possible to find a cover where all elements appear only once.

### 4.2 Applications of the Set Cover Problem

The set cover problem and the set partition problem have lots of applications. Routing, scheduling and location problems are often described as a set cover problem.

We find an example of the set cover problem at General Motors. General Motors needs to buy materials and they had different offers for combinations of materials. In this case one wants to find a set cover, such that all the materials are bought, while minimizing the costs.

An example of the set partition problem can be found in the airline industry for crew scheduling. One wants to find a cover where every flight of an airline is assigned to exactly one cockpit crew [1].

### 4.3 Characteristics of the Set Cover Problem

The question whether there exists a solution to the set cover problem \((U, S)\) with at most \(k\) subsets, is called the decision version of the set cover problem as opposed to the optimization version (1)-(3). The decision problem is NP-complete. NP stands for non-deterministic polynomial time and an NP problem is a decision problem that can be solved in polynomial time by a non-deterministic Turing machine. No polynomial time algorithm for solving an NP-complete problem has been found so far and it is conjectured that such an algorithm will never be found. One could, however, hope for a good quality approximation of the optimal solution.

The optimization version problem of the set cover problem (1)-(3) is an NP-hard problem. An NP-hard problem is a problem that is as least as hard as any problem in NP [9].

### 4.4 Solution Methods

For set cover instances that are relatively small, it is possible to solve the problem to optimality by linear programming based Branch-and-Bound. When the problem size increases, the worst-case solution time increases exponentially, due to the increase in the size of the branching tree. Because of the many large applications of set cover, lots of research is done in finding better/other ways of solving or approximating the problem.

For many problems heuristics give satisfying results. The Greedy Heuristic is an example of a heuristic that lends itself for the set cover problem. This heuristic does not always find the optimal solution, but it is a fast and easy algorithm. We will use this algorithm to find a solution for KLM’s problem. Therefore the algorithm will be discussed in the next section.
Also algorithms are combined in order to determine an exact solution to the problem. First a heuristic is used to find an upper bound on the optimal value and after that optimization (exact) algorithms can be used to decrease the size of the branching tree.

CPLEX is a software package from IBM that uses the exact algorithm Branch-and-Bound to find a solution. This software package will be used to find an optimal solution to the answer of my problem. At the end of this chapter some more information will be given about CPLEX.

4.4.1 Greedy Algorithm

The Greedy algorithm is a method, that iteratively chooses the subset with the smallest costs per element. Let $C$ be the set of subsets of $S$ that are already in the cover. The union of the elements of the subsets in $C$ will at termination cover the elements of the universe. Also, let $\alpha$ be the average cost per newly covered element. The algorithm looks like follows:

1. $C = \emptyset$
2. While $C \neq U$ do
   Find the subset $S_i \in S$ for which the costs per element is the lowest. Let $i' = \arg\min_{S_i : S_i \notin C} \{\alpha(S_i)\}$ with $\alpha(S_i) = \frac{c(S_i)}{|S_i - C|}$
   $C = C \cup S_i$
3. The sets chosen in Step 2 of the algorithm form the feasible solution $C$.

For example, $U = \{1, 2, 3, 4, 5, 6\}$
$S = \{\{1, 2, 3, 4\}, \{3, 4, 5\}, \{1, 2, 6\}\}$
and $c(S_1) = 3, c(S_2) = 5, c(S_3) = 2$

We start with $C = \emptyset$.

When we follow the algorithm we calculate $\alpha(S_i)$ for the three subsets. In the first iteration we find that the smallest value $\alpha(S_i)$ is attained for $S_3$:

$$\frac{c(S_3)}{|S_3 - C|} = \frac{2}{3}$$

$C := S_3 = \{1, 2, 6\}$

In the second iteration the algorithm the smallest value $\alpha(S_i)$ is attained for $S_1$:

$$\frac{c(S_1)}{|S_1 - C|} = \frac{3}{2} = 1$$

and sets $C := C \cup S_1 = \{\{1, 2, 3, 4\}, \{1, 2, 6\}\}$.

And because the universe is not fully covered at the third and last iteration the set $S_2$ is added:
which gives: $C := C \cup S_2 = \{\{1, 2, 3, 4\}, \{1, 2, 6\}, \{3, 4, 5\}\}$. The set $C$ of subsets covers all the elements of the universe and is therefore the solution of the algorithm.

The Greedy Algorithm does not always find the optimal solution, and the example above shows this [10]. The Greedy algorithm finds three subsets in the solution, where the optimal solution has two subsets, namely $S_2$ and $S_3$.

The Greedy Algorithm always finds a solution $C$ such that the ratio $\rho = \frac{|C|}{\text{OPT}}$ is always less or equal to $ln(n)$, where $n$ is the number of elements in the universe. Feige [11] proved that no polynomial time algorithm can give a better approximation ratio, that is, $\rho \geq ln(n)$ for set cover.

### 4.4.2 CPLEX

The IBM ILOG CPLEX Optimization Studio, or shortly CPLEX, is an optimization software package. The package is named after the simplex method, but it now provides other methods for mathematical programming. CPLEX solves large linear programming problems, quadratic programming problems and also integer programming problems, such as the set cover problem. For integer optimization CPLEX uses linear programming based Branch-and-Bound together with various heuristics [7].

**Branch-and-Bound and Cutting Planes**

The quality of the linear relaxation is improved by the use of “cutting planes”. The terms Branch-and-Bound and Cutting Planes will be explained in the following subsection.

**Branch-and-Bound** is an algorithm that is used for solving integer optimization problems. With the use of a so-called branching tree, all possible solution candidates are considered (branching). By checking upper and lower bounds of the nodes in the branching tree, large subsets of solutions are eliminated when one knows that these subsets will never be candidates for an optimal solution (bounding).

We look at the following linear integer problem (IP):

$$z_{OPT} = \min \quad c^T x$$
subject to $\quad Ax \leq b$
$x \geq 0, x \in \mathbb{Z}^n$

The linear relaxation of (IP) is the problem that is obtained by ignoring the restriction $x \in \mathbb{Z}^n$. The optimal value of the linear relaxation is a lower bound $\hat{z}$ on the optimal value $z_{OPT}$. The value of a feasible solution to (IP) is an upper bound $\hat{z}$ on $z_{OPT}$. These values are used recursively in the Branch-and-Bound tree to find the optimal value $z_{OPT}$. 

20
How the algorithm works is explained by looking at a possible node of the branching tree. We look at a node $k$ of the branching tree and let $x_j$ be a variable with value $f$, which is not an integer. We branch on $x_j$ and obtain two subproblems by respectively adding the constraints $x_j \leq \lfloor f \rfloor$ and $x_j \geq \lceil f \rceil$:

With the above method of branching, all possible solutions can implicitly be considered. Because it is desirable not to calculate all possible nodes, the tree is pruned when one of the next three situations occur:

1. The LP-relaxation is infeasible at node $l$.
2. The LP-solution is integral.
3. If $z_{LP}^l \geq \overline{z}$, where $\overline{z}$ is the value of the current best (smallest) upper bound. We prune this situation since the value of a solution in the subtree rooted at node $l$ can never be better than $z_{LP}^l$.

When all nodes are either calculated or pruned, the optimal solution of the linear integer problem is found. (Note that other ways of branching may be possible.)
A cutting plane for an integer problem is a new functional constraint that reduces the feasible region for the LP-relaxation without eliminating any feasible integer solutions. Since the linear relaxation is tightened by the addition of cutting planes, the lower bound from the relaxation is improved. In this way we hope for a smaller branching tree. For more details, see [12].

Figure 11: A possible cutting plane
5 Reduction of Closures

In this chapter we consider the problem:

Reduce the amount of Closures sent to the reservation system

At first a mathematical formulation is introduced in terms of the set cover problem, then the way the problem is programmed is explained and finally some results are shown.

5.1 Mathematical Formulation

A data point for a Closure that is sent to Altea looks like:

<table>
<thead>
<tr>
<th>Closure</th>
<th>Origin City</th>
<th>Destination City</th>
<th>Route</th>
<th>Pricing Class</th>
<th>Points of Sale</th>
<th>Period</th>
<th>Frequency</th>
</tr>
</thead>
</table>

In order to satisfy to the restrictions of Altea, we would like to put our problem in the form of the set cover problem. To do this, first we put our information into a matrix form. We have seen that for the origin and destination cities, the route and the pricing class, only a single value can be specified to Altea. Therefore we split the problem into a set of matrices, one matrix for each unique origin and destination pair, route and pricing class combination.

For the points of sale, period of time and the frequency, there are more possibilities. In the matrix, rows represent the points of sale and columns the number of days before departure. An element of the matrix that equals 1 represents a Closure.

More formally, we introduce matrix \( A = a_{ij} \), for every combination of origin and destination city, route and a pricing class. An element of matrix \( A \) stands for:

\[
a_{ij} = \begin{cases} 
1 & \text{if points of sale } i \text{ is closed } j \text{ days before departure} \\
0 & \text{otherwise}
\end{cases}
\]

We look at an example for a matrix \( A \) with three different points of sale, namely the Netherlands, the United States and the United Kingdom and look at Closures for a flight with flight number KL1000 from Amsterdam to London for pricing class T one week ahead, from October 1, 2011. Suppose that the flight is closed two days before departure for point of sale the United States, and three days before departure for the other two points of sale. This looks like:

<table>
<thead>
<tr>
<th></th>
<th>01/10/2011</th>
<th>02/10/2011</th>
<th>03/10/2011</th>
<th>04/10/2011</th>
<th>05/10/2011</th>
<th>06/10/2011</th>
<th>07/10/2011</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>closed</td>
<td>closed</td>
<td>closed</td>
<td>closed</td>
<td>open</td>
<td>open</td>
<td>open</td>
</tr>
<tr>
<td>UK</td>
<td>closed</td>
<td>closed</td>
<td>closed</td>
<td>closed</td>
<td>open</td>
<td>open</td>
<td>open</td>
</tr>
<tr>
<td>US</td>
<td>closed</td>
<td>closed</td>
<td>closed</td>
<td>open</td>
<td>open</td>
<td>open</td>
<td>open</td>
</tr>
</tbody>
</table>

Matrix \( A \) looks like:

\[
A = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0
\end{pmatrix}
\]

With the information of matrix \( A \) we can construct the universe \( U \).
The size of the universe $\mathcal{U}$ is the number of Closures that needs to be covered in matrix $A$ and the elements correspond to the place of the Closures in matrix $A$. This is done as follows:

When $a_{ij} = 1$, then element $\{n \ast (i - 1) + j\}$ is in $\mathcal{U}$.

with $n$ the number of columns in matrix $A$.

For the example $\mathcal{U}$ consists of the elements:

$\mathcal{U} = \{1, 2, 3, 4, 8, 9, 10, 11, 15, 16, 17\}$

If an element in matrix $A$ is equal to zero, we want this element not to be in the cover. Therefore we construct a complement of the universe $\mathcal{U}^c$, with the elements that should not be in the cover. This complement looks like:

When $a_{ij} = 0$, then element $\{n \ast (i - 1) + j\}$ is in $\mathcal{U}^c$.

For our example $\mathcal{U}^c$ consist of the elements:

$\mathcal{U}^c = \{5, 6, 7, 12, 13, 14, 18, 19, 20, 21\}$

Now we look at how the subsets are defined. In section 5.2 an algorithm is introduced that generates all the subsets. To understand how we can formulate the set cover problem, in this section I will show how possible subsets can be generated for our example.

We begin by looking at a possible data point to Altea. Such a data point has to be specified for a period of time (begin and end date), one or more points of sale and a frequency. For the example we can introduce the next datapoint:

• Points of Sale: NL, UK and US
• Start date: 2 October 2011
• End date: 4 October 2011
• Frequency: 1000001

In the format of a data point to Altea, this example looks like:

<table>
<thead>
<tr>
<th>Closure</th>
<th>Amsterdam</th>
<th>London</th>
<th>KL1000</th>
<th>T</th>
<th>NL, UK, US</th>
<th>from 02-10-11 until 04-10-11</th>
<th>1000001</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/10/2011</td>
<td>02/10/2011</td>
<td>03/10/2011</td>
<td>04/10/2011</td>
<td>05/10/2011</td>
<td>06/10/2011</td>
<td>07/10/2011</td>
<td></td>
</tr>
<tr>
<td>NL</td>
<td>closed</td>
<td>closed</td>
<td>closed</td>
<td>open</td>
<td>open</td>
<td>open</td>
<td></td>
</tr>
<tr>
<td>UK</td>
<td>closed</td>
<td>closed</td>
<td>closed</td>
<td>open</td>
<td>open</td>
<td>open</td>
<td></td>
</tr>
<tr>
<td>US</td>
<td>closed</td>
<td>closed</td>
<td>closed</td>
<td>open</td>
<td>open</td>
<td>open</td>
<td></td>
</tr>
</tbody>
</table>

Because 3 October is on a Monday in 2011, this frequency means that the route is closed on Sunday 2 October and Monday 3 October and that the route is open on Tuesday 4 October for Points of Sale NL, UK and US. This corresponds to the following table:
We see that this data point is not specified for all days. This data point keeps the routes open for the days for which it is not specified on, and therefore the above table can be seen as:

<table>
<thead>
<tr>
<th></th>
<th>01/10/2011</th>
<th>02/10/2011</th>
<th>03/10/2011</th>
<th>04/10/2011</th>
<th>05/10/2011</th>
<th>06/10/2011</th>
<th>07/10/2011</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>open</td>
<td>closed</td>
<td>closed</td>
<td>open</td>
<td>open</td>
<td>open</td>
<td>open</td>
</tr>
<tr>
<td>UK</td>
<td>open</td>
<td>closed</td>
<td>closed</td>
<td>open</td>
<td>open</td>
<td>open</td>
<td>open</td>
</tr>
<tr>
<td>US</td>
<td>open</td>
<td>closed</td>
<td>closed</td>
<td>open</td>
<td>open</td>
<td>open</td>
<td>open</td>
</tr>
</tbody>
</table>

This table corresponds to the next matrix:

\[
S_1' = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}
\]

The subsets \(S_k\) can be constructed from matrices \(S_k'\) as follows:

\[S_k' = [s_{ij}], \text{ when } s_{ij} = 1, \text{ then element } \{n \ast (i - 1) + j\} \text{ is in } S_k.\]

For the example \(S_1\) consists of the elements:

\[S_1 = \{2, 3, 9, 10, 16, 17\}\]

In a same way we can find a second subset \(S_2\) for the data point:

<table>
<thead>
<tr>
<th>Closure</th>
<th>Amsterdam</th>
<th>London</th>
<th>KL1000</th>
<th>T</th>
<th>NL, UK, US</th>
<th>from 01-10-11 until 03-10-11</th>
<th>1000011</th>
</tr>
</thead>
</table>

where \(S_2 = \{1, 2, 3, 8, 9, 10, 17, 18, 19\}\)

And a third subset \(S_3\) for the data point:

<table>
<thead>
<tr>
<th>Closure</th>
<th>Amsterdam</th>
<th>London</th>
<th>KL1000</th>
<th>T</th>
<th>NL, UK</th>
<th>from 01-10-11 until 04-10-11</th>
<th>1100011</th>
</tr>
</thead>
</table>

where \(S_3 = \{1, 2, 3, 4, 8, 9, 10, 11\}\).

For this example we can see that \(S_2\) and \(S_3\) cover the universe and are therefore a solution to the set cover problem.

The set cover problem is then defined as:

- Let \(\mathcal{S} = \{S_1, ..., S_n\}\) be the set of feasible subsets based on matrices \(S_k' (k = 1, ..., n)\), where each subset represents a data point that can be sent to Altea.
- Let \(\mathcal{U}\) be the elements of matrix \(A\) that should be covered.
- Let \(\mathcal{U}^c\) be the elements of matrix \(A\) that should not be covered.
The specific set cover problem looks like:

\[
\begin{align*}
\text{minimize} & \quad \sum_{S_i \in S} c(S_i) \cdot x_{S_i} \\
\text{subject to} & \quad \sum_{S_i : e \in S_i} x_{S_i} \geq 1 \quad \text{for } e \in U \\
& \quad \sum_{S_i : e \in S_i} x_{S_i} = 0 \quad \text{for } e \in U^c \\
& \quad x_{S_i} \in \{0, 1\} \quad \text{for all } S_i \in S
\end{align*}
\]

The solution that we have found for the set cover problem, is not an answer to the set partition problem. Because there are elements that appear in the cover more than once. The set partition problem is defined as:

- Let \( S = \{S_1, \ldots, S_n\} \) be the set of feasible subsets, where each subset is a data point that can be sent to Altea.
- Let \( U \) be the elements of matrix \( A \) that should be covered only once by the subsets \( S \).
- Let \( U^c \) be the elements of matrix \( A \) that should not be covered.

And the set partition problem the problem can be formulated as:

\[
\begin{align*}
\text{minimize} & \quad \sum_{S \in S} c(S) \cdot x_S \\
\text{subject to} & \quad \sum_{S : e \in S} x_S = 1 \quad \text{for } e \in U \text{ with } A'_e = 1 \\
& \quad \sum_{S : e \in S} x_S = 0 \quad \text{for } e \in U \text{ with } A'_e = 0 \\
& \quad x_S \in \{0, 1\} \quad \text{for all } S \in S
\end{align*}
\]

With the next two subsets the set partition problem can be solved:

Where \( S_1 = \{1, 2, 3, 8, 9, 10, 17, 18, 19\} \), which corresponds to the data point to Altea:

<table>
<thead>
<tr>
<th>Closure</th>
<th>Amsterdam</th>
<th>London</th>
<th>KL1000</th>
<th>T</th>
<th>NL, UK, US</th>
<th>from 01-10-11 until 03-10-11</th>
<th>1000011</th>
</tr>
</thead>
</table>

and \( S_2 = \{4, 11\} \) which corresponds to the data point to Altea:

| Closure | Amsterdam | London | KL1000 | T | NL, UK | from 04-10-11 until 04-10-11 | 0100000 |
5.2 The Solution and Programming Part

Now a mathematical formulation of the problem is found, it is possible to derive a solution approach. For all possible combinations of routes on pricing classes between two cities, a matrix is generated with the information about Closures put into matrix form that looks like matrix $A$ of the previous chapter. For all these matrices a number of data points that can be sent to Altea is found by using the following two steps:

- **Step 1:** Generate feasible subsets in order to cover all the Closures, where the non-Closures should not be covered.
  
  For both the set cover and set partition problem a recursion method is used to generate the feasible subsets. As we shall see later, we need to generate substantially more subsets in the partition case compared to the cover case.

- **Step 2:** Use a set cover and set partition algorithm to get an (optimal) solution. Where the (optimal) solution is the minimum amount of Closures for which all the Closures are covered.
  
  The problem is approached in two ways, with the Greedy Algorithm and with CPLEX.

In the figure below the program structure is shown graphically:

![Diagram](image)

**Figure 12:** Structure of the program for reducing the amount of Closures

The graph shows an arrow from the Greedy Algorithm to CPLEX. The Greedy Algorithm finds a feasible solution for the set cover problem and this solution could be used as an upper bound for CPLEX. At the end of this chapter, after we have found some results for the problem, this option will be evaluated.

In this section of the report the recursion methods are explained, as well as the two ways of solving the problem.
5.2.1 Step 1: A recursion method is used to generate feasible subsets

By using a recursion method, feasible subsets are found as input of the subsets $S_i$ of the set cover and partition problem. Both methods use the matrices $A$ from the mathematical formulation and the characteristics of the data that can be sent to Altea.

With the next algorithm the subsets are generated:

1. Look at all combinations of rows and find the common Closures.

2. For each combination of rows do the following:
   (a) Take the first Closure in the array, this is the start date of the subset.
   (b) Make all combinations of frequencies for the array.
   (c) The end date of the subset is determined by looking at how long the subset can be lengthened and still matches the frequency.
   (d) Take the next Closure in the array (this is the new start date) and return to step 2b.

Example of the algorithm

We look at an example for a matrix $A$:

\[
A = \begin{pmatrix}
1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1
\end{pmatrix}
\]

and we follow the algorithm step by step.

1. Look at all combinations of rows and find the common Closures

In the first step we look at a combination of rows. Because we have two rows, three combinations are possible, namely row 1, row 2 and the combination of both rows. For our example we look at the combination of row 1 and row 2. The common Closures are put into one array:

\[
\begin{pmatrix}
1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1
\end{pmatrix}
\]

Note that taking two (or more) rows together corresponds to combining two (or more) points of sale.

2a. Take the first Closure in the array, this is the start date of the subset

Now we look at the first Closure and choose this as our start date. In our example the first element of the array is a Closure and therefore the start date is 0. Because we constructed our matrix $A$ where the column $i$ represents the tickets that can be sold $i$ days before departure, the start date corresponds to the actual date of the departure of the flight.

2b. Make all combinations of frequencies for the array

Now we make all possible frequencies with start date 0. As we saw in the previous section a frequency are seven numbers reflecting the seven days of the week, that can either be 0 or 1. For our combined rows 1& 2 we find 4 possible frequencies:
2c. The end date of the subset is determined by looking at how long the subset can be lengthened and still matches the frequency

The end date is determined by extrapolating the subset and checking whether it still matches the array. For the four frequencies we find four subsets $S$. These four subsets below can be translated back to the elements of matrix $A$ and from there on the corresponding elements $e \in \mathcal{U}$ can be found:

Combined rows 1 & 2, Start date = 0; End date = 7, with frequency

$$[1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1]$$

(combset 1)

Combined rows 1 & 2, Start date = 0; End date = 9, with frequency

$$[1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1]$$

(subset 2)

Combined rows 1 & 2, Start date = 0; End date = 7, with frequency

$$[1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1]$$

(subset 3)

Combined rows 1 & 2, Start date = 0; End date = 9, with frequency

$$[1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1]$$

(subset 4)

2d. Take the next Closure in the array and return to step 2b

Now all combinations of subsets can be found for the next Closure in the array, so for start date = 2.

Until now the algorithms are the same for both methods, but now a distinction can be made between two recursion methods, one for the set cover problem and one for the set partition problem.

**Recursion Method for the Set Cover Problem**

When a subset $S_i$ is found that contains at least the same information as another subset $S_j$, we say that subset $S_i$ dominates subset $S_j$. For the set cover problem only the dominating subsets have to be used to solve the problem. Therefore we look at the subsets that are generated by the algorithm above and drop all the subsets that are dominated by another subset.

In the example subsets 1, 2 and 3 are dominated by subset 4, so only subset 4 has to be taken into account. By using the recursion method for set cover only one subset is sent to **Step 2** of the programming part for the combination of the two rows, namely:

Rows 1 & 2, Start date = 0; End date = 9, with frequency

$$[1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1]$$

**Recursion Method for the Set Partition Problem**

For the set partition problem, all feasible subsets have to be generated and to be taken into account.
Otherwise it could be that the set partition problem can is infeasible, as we saw in Chapter 4. The method therefore does not eliminate subsets that are dominated by another subset. Instead the method splits the subsets that are found with the same strategy as before, into all possible other subsets by repeatedly removing the last Closure in the array. When for instance the next subset is found:

Rows 1 & 2, Start date = 0; End date = 9, with frequency

\[
\begin{bmatrix}
1 & 0 & 1 & 1 & 0 & 0 & 0
\end{bmatrix}
\]

Then also the next four subsets are generated:
Rows 1 & 2, Start date = 0; End date = 4, with frequency

\[
\begin{bmatrix}
1 & 0 & 1 & 1 & 0 & 0 & 0
\end{bmatrix}
\]

Rows 1 & 2, Start date = 0; End date = 3, with frequency

\[
\begin{bmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

Rows 1 & 2, Start date = 0; End date = 0, with frequency

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

For the set partition problem many more data points are generated and used as input for the second step of the program. An advantage of solving the set partition problem instead of the set cover problem is that no information is sent to Altea more than once, which could make it easier for the user. A disadvantage is that many more data points are generated and the run time will increase a lot for solving the set partition problem.

5.2.2 Step 2: The problem is solved

In the second step, the set cover problem is solved by using the subsets that are generated, together with the matrix $A$ to find a solution for the problem. Two ways of approaching the problem are used: A Greedy Algorithm and Branch-and-Bound as implemented in CPLEX.

As we saw in Chapter 4 the Greedy Algorithm always takes the step with the lowest cost per element. In our problem all the subsets have the same costs. Therefore we take the subset that has the largest amount of Closures at every step of the algorithm.

As for CPLEX, we programmed the set cover problem as the linear integer problem as defined in Chapter 5.
5.3 Results

For all origin and destination pairs, routes and pricing classes, matrices are generated with the information about the Closures. There was a total of 201,617 matrices generated with a total of 556,770 rows. These matrices represent 13,491,488 Closures.

We only used the recursion method for the set cover problem, because the other method took too much time to run. After running the recursion method for the set cover problem both CPLEX and the Greedy Algorithm were used to approach the problem. We found the next results:

<table>
<thead>
<tr>
<th></th>
<th>Without Optimization</th>
<th>CPLEX</th>
<th>Greedy Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total data points sent to Altea</td>
<td>≈ 13,000,000</td>
<td>589,269</td>
<td>600,831</td>
</tr>
<tr>
<td>Run time (min)</td>
<td></td>
<td>221</td>
<td>352</td>
</tr>
</tbody>
</table>

The number of 13 million data point without using an optimization method is an average number of data points that is sent to Altea.

There are a couple of remarkable observations to make:

- With both solution methods it is possible to reduce the number of data points by a lot. In the problem definition we noticed that it would be useful to reduce the amount of data points by about 30%, but in this first subproblem it became possible to reduce the number of data points by over 95%.

- CPLEX did a better job at the reduction, because it uses an exact way of finding a solution. By using the Greedy Algorithm we found 11,562 additional data points, this is about 2% more data points than the optimal solution. The difference is not significantly high.

- As we can observe, Greedy requires longer computing times. This is probably due to a combination of implementation issues and efficiency of CPLEX. It is of course a very positive outcome, and slightly unexpected, that it is faster to compute the optimal solution, than the approximate solution by Greedy.

- In Chapter 5.2 we discussed the possibility of using the approximate solution of Greedy as an upper bound for CPLEX. Because of the shorter computing time of CPLEX, this possibility is not useful.

- We see that there is a very small difference between the total amount of Closures in the matrices (13,491,488) and the amount of Closures that KLM sends to Altea at this moment. This means that at this moment Closures are not handled efficiently by KLM. An explanation may be that for example the feature that it is possible to merge rows is not used at this moment.
6 Reduction of PNVs

In this chapter the reduction of the PNVs is discussed, where the following subproblem is considered:

Reduce the amount of PNVs sent to the reservation system

A similar outline is followed as for the previous chapter. At first a mathematical formulation is given in terms of the set cover problem, then the way the problem is solved is discussed, and finally some results are shown.

6.1 Mathematical Formulation

For the PNVs we introduced a format for how one data point can be sent to Altea:

<table>
<thead>
<tr>
<th>Value</th>
<th>Origin City</th>
<th>Destination City</th>
<th>Pricing Class</th>
<th>Points of Sale</th>
<th>Period</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>PNV</td>
<td>City</td>
<td>City</td>
<td>Class of Sale</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Again we see that for the value of the PNV, the origin and destination city and the pricing class, only one value can be specified per data point. But there is some more freedom in the points of sale, the period and the frequency. The period and the frequency can be determined by looking at the date of the flight. Therefore we look at matrices for one origin city, destination city and pricing class, where the rows represent the points of sale and the columns the number of days before departure.

As an example we look at a the city pair from Amsterdam to London, from October 1, 2011. We look at the value for the PNVs one week ahead for three points of sale: the Netherlands, the United States and the United Kingdom:

<table>
<thead>
<tr>
<th></th>
<th>01/10/2011</th>
<th>02/10/2011</th>
<th>03/10/2011</th>
<th>04/10/2011</th>
<th>05/10/2011</th>
<th>06/10/2011</th>
<th>07/10/2011</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>450</td>
<td>450</td>
<td>450</td>
<td>400</td>
<td>300</td>
<td>400</td>
<td>350</td>
</tr>
<tr>
<td>UK</td>
<td>450</td>
<td>500</td>
<td>350</td>
<td>450</td>
<td>350</td>
<td>400</td>
<td>300</td>
</tr>
<tr>
<td>US</td>
<td>500</td>
<td>500</td>
<td>450</td>
<td>400</td>
<td>450</td>
<td>400</td>
<td>400</td>
</tr>
</tbody>
</table>

A feature of Altea is used to get a better reduction. When two data points are sent to Altea with different values, the lowest value of the PNV is chosen. So if we look at our example and we want to make subsets for PNVs with a price of €400.00, the elements of the matrix with the value 400 should at least be sent to Altea, but the value 400 may be sent to Altea for lower valued elements as well.

In the table below the bold values show which elements should at least be sent to Altea, these are exactly the PNVs with value 400. The italic values show which elements can also be sent to Altea, these are the PNVs with a value lower than 400.

<table>
<thead>
<tr>
<th></th>
<th>01/10/2011</th>
<th>02/10/2011</th>
<th>03/10/2011</th>
<th>04/10/2011</th>
<th>05/10/2011</th>
<th>06/10/2011</th>
<th>07/10/2011</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>450</td>
<td>450</td>
<td>450</td>
<td>400</td>
<td>300</td>
<td>400</td>
<td>350</td>
</tr>
<tr>
<td>UK</td>
<td>450</td>
<td>500</td>
<td>350</td>
<td>450</td>
<td>350</td>
<td>400</td>
<td>300</td>
</tr>
<tr>
<td>US</td>
<td>500</td>
<td>500</td>
<td>450</td>
<td>400</td>
<td>450</td>
<td>400</td>
<td>400</td>
</tr>
</tbody>
</table>
For every PNV-value $v$ in the above table we construct two 0-1 matrices, matrix $A$ and matrix $B$.

An element of matrix $A$ stands for:

$$a_{ij} = \begin{cases} 
1 & \text{if the value of the PNV is equal to } v \text{ for point of sale } i \text{ at } j \text{ days before departure} \\
0 & \text{otherwise} 
\end{cases}$$

For our example with a PNV value of 400, matrix $A$ looks like:

$$A = \begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}$$

Note that the elements in matrix $A$ that are equal to 1, correspond to the bold values in the table above.

An element for matrix $B$ stands for:

$$b_{ij} = \begin{cases} 
1 & \text{if the value of the PNV is equal to or smaller than } v \text{ for point of sale } i \text{ at } j \text{ days before departure} \\
0 & \text{otherwise} 
\end{cases}$$

For our example for the PNV value of 400, matrix $B$ looks like:

$$B = \begin{pmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}$$

Note that the elements in matrix $B$ that equal to 1, correspond to the bold and italic values in the table above.

When an element in matrix $A$ is equal to one, this element should be covered by at least one of the subsets and should therefore be in the universe. Thus,

When $a_{ij} = 1$, then element $\{n \times (i - 1) + j\}$ is in $U$.

When an element in matrix $B$ is equal to zero, this element should not be covered in any of the subsets and should therefore be in the complement of the universe. Thus,

When $b_{ij} = 0$, then element $\{n \times (i - 1) + j\}$ is in $U^c$.

The set cover problem is then defined as:

- Let $S = \{S_1, ..., S_n\}$ be the set of feasible subsets, where each subset is a data point that can be sent to Altea.
- Let $U$ be the elements of matrix $A$ that should be covered.
- Let $U^c$ be the elements of matrix $B$ that should not be covered.
This specific set cover problem can be formulated as the following integer linear problem:

\[
\begin{align*}
\text{minimize} & \quad \sum_{S_i \in S} c(S_i) \cdot x_{S_i} \\
\text{subject to} & \quad \sum_{S_i \in S} x_{S_i} \geq 1 \quad \text{for } e \in U \\
& \quad \sum_{S_i \in S} x_{S_i} = 0 \quad \text{for } e \in U^c \\
& \quad x_{S_i} \in \{0, 1\} \quad \text{for all } S_i \in S
\end{align*}
\]  

(12)  

(13)  

(14)  

(15)

Again the costs for \(c(S) = 1\) for all subsets, because all subsets are constructed such that they correspond to exactly one data point that can be sent to Altea.

For the above example with PNV-value 400, our universe \(U\) consists of:

\[U = \{4, 6, 13, 18, 20, 21\}\]

And \(U^c\) consists of:

\[U^c = \{1, 2, 3, 8, 9, 11, 15, 16, 17, 19\}\]

With the information of matrix \(B\) we can construct subsets in a similar way as for the subsets in Chapter 5. How all the subsets are generated will be explained in the next section of this chapter.

For the example we construct two subsets \(S_1\) and \(S_2\), where:

\[S_1 = \{6, 7, 13, 14, 20, 21\}\]

This subset corresponds to the data point to Altea:

\[
\begin{array}{cccccc}
400 & \text{Amsterdam} & \text{London} & T & \text{NL, UK, US} & \text{from 06-10-11 until 07-10-11} & 0001100
\end{array}
\]

and we construct \(S_2\) as:

\[S_2 = \{4, 6, 7, 18, 20, 21\}\]

This subset corresponds to the data point to Altea:

\[
\begin{array}{cccccc}
400 & \text{Amsterdam} & \text{London} & T & \text{NL, US} & \text{from 05-10-11 until 07-10-11} & 0011100
\end{array}
\]

These two subsets cover \(U\) and none of the elements of \(U^c\) and therefore we found a solution to this set cover problem.

Note that if we would not have used the information of matrix \(B\), but only the information of matrix \(A\), we needed at least 3 subsets to cover the universe. The feature that also lower PNV-value can be added, therefore reduces the amount of data points that need to be sent to Altea.

The set partition problem is formalized as the set cover problem, except from constraint (13). The inequality constraint is replaced by an equality constraint.
6.2 The Solving and Programming Part

For the PNVs, at first files are generated for every combination of origin and destination city for all subclasses. The files contain information about all PNV-values for every point of sale and 365 days before departure. For each file the matrices $A$ and $B$ are constructed.

- **Step 1:** Generate feasible subsets of matrix $B$ for the set cover problem.
  
  Because we saw in the previous chapter that the recursion method for the set partition problem took too much time, we only programmed the set cover problem for this subproblem.

- **Step 2:** Use a set cover algorithm to get an (optimal) solution, where the (optimal) solution is the minimum amount of PNVs for which all the PNVs are covered.

  The problem is solved in two ways, with the Greedy Algorithm and with CPLEX.

In the figure below the program structure is shown graphically:

![Figure 13: Structure of the program for reducing the amount of PNVs](image)

6.2.1 Step 1: A recursion method is used to generate feasible subsets

Again a recursion method is used to generate feasible subsets $S_i$ as input for the set cover problem. The recursion method uses matrices $B$ from the mathematical formulation and the characteristics of the data that can be sent to Altea.

In the algorithm, the subsets were initially generated in the same way as for the reduction of Closures. But because for some origin and destination cities the PNVs were defined on more than 20 points of sale, the algorithm took too much time. When all combinations of points of sale are calculated, over 16 million combinations are looked at for every value that is in the matrix. I decided not to look at all
possible combinations, but at a selection.

Three combinations of rows of matrix $B$ are chosen to look at when programming the subsets:

- All rows are checked separately to at least cover every element that needs to be covered.
- The combination of all rows together, because often one value is the same for a certain day before departure at all points of sale.
- For all combinations of rows it is checked whether these rows are exactly the same. I found out that it occurs frequently that two or more rows are exactly the same in a matrix. If these two rows can be put together, this reduces the total amount of data point for PNVs a lot.

The rest of the algorithm stays the same including the removing of the dominative subsets for the set cover problem.

6.2.2 Step 2: The problem is solved

The problem is solved with matrix $A$ and the datasets $S_i$ as input. Again the Greedy Algorithm and CPLEX are used to solve the problem. This part is programmed the same way is in Chapter 5.

Note that in this case for most matrices an optimal solution is not found. This is because not all possible subsets are generated, only a selection of combinations of rows are chosen.
6.3 Results

For all origin and destination pairs and pricing classes, matrices are generated with the information about the PNVs. For the 206 cities that KLM flies on, a total of 1,013,520 matrices were generated.

The recursion method for generating subsets for the set cover problem was runned and again the Greedy Algorithm and CPLEX were used to approach the problem. We found the next results:

<table>
<thead>
<tr>
<th></th>
<th>Without Optimization</th>
<th>CPLEX</th>
<th>Greedy Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total data points to Altea</td>
<td>$\approx 57,000,000$</td>
<td>7,807,136</td>
<td>7,837,430</td>
</tr>
<tr>
<td>Run time (min)</td>
<td>1130</td>
<td>2306</td>
<td></td>
</tr>
</tbody>
</table>

The number of 57 million data points without using an optimization method is an average number of data points that is sent to Altea for the PNVs.

When we look at the results, some remarkable observations can be made:

- After optimizing the set cover problem, the total amount of yield data that has to be sent to Altea is reduced to some more than 7.8 million data points. The goal for the problem was to reduce the amount of data points with about 30%, where for this subproblem a reduction of 86% is reached.

- Because CPLEX uses an exact way of finding a solution, a better result is found for CPLEX. By using the Greedy Algorithm an additional 30,294 data points were found. This is about 0.3% more data points than CPLEX, which is a very small difference.

- In the recursion method, not all combinations of rows are checked and therefore not all possible subsets are generated. Therefore the above results are not the optimal solution for the problem, but it is a very good solution. If one wants to find a better reduction, it is possible to look at more combinations of rows.

- We found that CPLEX finds the solution faster than the Greedy Algorithm. For this problem the runtime of CPLEX is about twice as fast. CPLEX probably uses some smart algorithms to find a good feasible solution quickly. Again we found a positive outcome, where the optimal solution can be determined faster than the approximate solution.

- The runtime of the program is about 18 hours for CPLEX. Because it is possible to run the program in parallel, the program is fast enough to reach the time limit. The data points for each of the 206 cities can be found in parallel and therefore an average of 6 minutes per city will be enough to find the data points. It would of course still be interesting and useful to decrease the computing time, and therefore we give some recommendations in Chapter 8 that might lead to such a decrease.
7 Conclusions

In this chapter some conclusions are made based on the results that are found during the project for reducing the amount of yield data for KLM.

In Chapter 5.2 two recursion methods are introduced, one for the set cover problem and one for the set partition problem. Because the second method took too much time to run, we only looked at the set cover problem. The set cover problem may send data points to Altea more than once. This might be confusing for the user, and therefore this disadvantage should be taken into account.

The yield data points of KLM were reduced from about 70 million data points to less than 8.5 million data points for the Closures and PNVs together. The first goal was to reach a reduction of about 30%, instead a reduction of 88% is reached. With this incredibly high reduction the restrictions to Altea will no longer be a problem. Also it will be possible for KLM and AirFrance to extend the revenue management system and the amount of data to Altea in the future.

During the project a Greedy Algorithm is compared to the optimization program CPLEX. Because CPLEX uses exact algorithm, a better reduction is found than with the Greedy Algorithm.

Also the runtime of the program is a lot faster when using CPLEX. For the PNV subproblem, the runtime is almost twice as long for the Greedy Algorithm than for CPLEX. This is probably because CPLEX uses some smart and efficient heuristics to find a good solution.

Because CPLEX finds a faster and better solution that the Greedy Algorithm, we can conclude that this is a better way of approaching KLM’s problem.

The total runtime of the program is pretty long, with a total of 21 hours for CPLEX. Because the program can be runned in parallel, the time limit can still be reached. It could be useful to decrease the runtime, therefore some recommendations are made in Chapter 8.
8 Recommendations

In this chapter some recommendations will be given based on the results that are found in the project.

As we saw in the conclusion it is possible to reduce the amount of data to 8.5 million data points. It is thereby possible to reduce the problem in such a way that the restrictions of Altea can be satisfied. Even if KLM and Air France want to send more information/data points to Altea, the restrictions will still be satisfied. A possible increase of the amount of data points would be if KLM introduces new flights.

During the project I mainly looked at how the amount of data that is sent to Altea can be reduced. But because the data needs to be sent to Altea at least once a day, it is also important to look at the runtime of generating the data points. This is something I did not focus on in the programming part and therefore I will give some recommendations to decrease the runtime:

• It is possible to decrease the runtime a lot by not generating all possible subsets, but sufficient subsets to find a cover. An example was found in Chapter 6, where not all combinations of rows were looked at, but only a smart selection.

• In Chapter 2.1 we saw that KLM uses two types of Closures, Delay Out Closures and Seats Sold Closures. For both types of Closures, KLM uses strategies to generate the Closures. By using the information of the strategies, it could be possible to generate the subsets faster.

• Because of the huge amount of matrices that had to be generated, this took a lot of time. The matrices were derived from a database and it could be a lot faster if these matrices were generated directly when the data is produced by the Revenue Management department.

In this project we only looked at the reduction of the data points to Altea. The database of KLM itself contains all the information about the PNVs and Closures as well. The data points are not saved in the database. KLM could save the information in the same format as they send it to Altea, to decrease the size of the database. This could also make the generation of the subsets a lot faster.
References


