Personnel Scheduling for a Global Tour Company

Mr Thomas K Gregson

Honours Thesis in Operations Research

Supervisors: Natasha Boland & Heng-Soon Gan
Second Reader: Moshe Sniedovich

© Thomas K Gregson, November 2006.
Personnel Scheduling for a Global Tour Company

Thomas K Gregson

SUPERVISORS: NATASHIA BOLAND & HENG-SOON GAN

Submitted November 3, 2006
Contents

1 Introduction ............................................. 4
   1.1 Related Problems ..................................... 7
   1.2 The Network ......................................... 8

2 Related Models ........................................ 10
   2.1 Linear Programming and Network Flows ................. 10
   2.2 Set Covering ......................................... 11
   2.3 Set Packing and Partitioning .......................... 13
   2.4 Auxiliary Variables .................................. 15

3 Techniques Used To Solve Models ....................... 17
   3.1 Simplex Algorithm .................................... 17
   3.2 Branch and Bound .................................... 19
   3.3 Column Generation & Dantzig-Wolfe Decomposition .... 21

4 Complete Model ......................................... 24
   4.1 Network Model ....................................... 25
   4.2 The Complete Model ................................... 26
      4.2.1 Transfer Cost .................................... 27
      4.2.2 Training Cost .................................... 28
      4.2.3 Work Load Constraints ............................ 29
      4.2.4 Auxiliary Variables to Constrain Consecutive Days Worked ................................ 30
   4.3 Computational Analysis of Complete Model ............ 32
      4.3.1 Computational Analysis of MAXGAP ............... 33
      4.3.2 Computational Analysis of Training Cost ........ 34
      4.3.3 Computational Analysis of Work Load ............ 35
      4.3.4 Computational Analysis of BigM ................. 36
      4.3.5 Computational Analysis of Auxiliary Variables .. 37
   4.4 Computational Infeasibility of Complete Model ....... 37

5 A Route Based Model & Heuristic ....................... 38
   5.1 The Route Based Model ............................... 39
      5.1.1 Training Cost .................................... 39
      5.1.2 Preferred Work Loads ............................. 41
   5.2 Initialization Heuristic .............................. 43
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.1</td>
<td>Network Model</td>
<td>44</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Transfer Cost</td>
<td>44</td>
</tr>
<tr>
<td>5.2.3</td>
<td>StartCost Model</td>
<td>47</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Work Load, Approximation</td>
<td>47</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Work Load</td>
<td>52</td>
</tr>
<tr>
<td>5.2.6</td>
<td>Auxiliary Variables to Constrain Consecutive Days Worked</td>
<td>54</td>
</tr>
<tr>
<td>5.2.7</td>
<td>Interaction Heuristics</td>
<td>55</td>
</tr>
<tr>
<td>5.3</td>
<td>Computational Analysis of Route Based Model</td>
<td>58</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Computational Analysis of penalty</td>
<td>58</td>
</tr>
<tr>
<td>5.4</td>
<td>Computational Analysis of Heuristic Methods</td>
<td>59</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Computational Analysis of StartCost</td>
<td>59</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Computational Analysis of BigM'</td>
<td>60</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Computational Analysis of BigM''</td>
<td>61</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Computational Analysis of pen</td>
<td>61</td>
</tr>
<tr>
<td>5.4.5</td>
<td>Computational Analysis of MINGAP</td>
<td>62</td>
</tr>
<tr>
<td>5.4.6</td>
<td>Computational Analysis of Average</td>
<td>62</td>
</tr>
<tr>
<td>5.4.7</td>
<td>Comparison of Heuristic Models</td>
<td>62</td>
</tr>
<tr>
<td>5.5</td>
<td>Comparison of Complete Model and Route Based Model</td>
<td>63</td>
</tr>
<tr>
<td>6</td>
<td>Column Generation Model</td>
<td>65</td>
</tr>
<tr>
<td>6.1</td>
<td>Master Problem</td>
<td>66</td>
</tr>
<tr>
<td>6.2</td>
<td>Subproblem</td>
<td>67</td>
</tr>
<tr>
<td>6.3</td>
<td>Computational Analysis of Column Generation</td>
<td>68</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Comparison of Column Generation and Route Based Model</td>
<td>68</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Comparison of Column Generation and Complete Model</td>
<td>71</td>
</tr>
<tr>
<td>7</td>
<td>Concluding Remarks</td>
<td>72</td>
</tr>
<tr>
<td>7.1</td>
<td>Results and Outcomes</td>
<td>72</td>
</tr>
<tr>
<td>7.2</td>
<td>Future Research</td>
<td>73</td>
</tr>
<tr>
<td>A</td>
<td>Implementation and Interfaces</td>
<td>76</td>
</tr>
<tr>
<td>B</td>
<td>Miscellaneous Code &amp; Figures</td>
<td>81</td>
</tr>
<tr>
<td>C</td>
<td>Sorting Algorithm</td>
<td>85</td>
</tr>
<tr>
<td>D</td>
<td>BigM' Algorithm</td>
<td>87</td>
</tr>
<tr>
<td>E</td>
<td>BigM'' Algorithm</td>
<td>88</td>
</tr>
<tr>
<td>F</td>
<td>Complete Model Code</td>
<td>89</td>
</tr>
<tr>
<td>G</td>
<td>Route Based Model Code</td>
<td>95</td>
</tr>
<tr>
<td>H</td>
<td>Column Generation Code</td>
<td>106</td>
</tr>
</tbody>
</table>
Acknowledgments

I am indebted to Dr Heng-Soon Gan who provided many helpful suggestions towards the thesis, particularly the mosel code. Heng-Soon’s time and clever ideas were an integral part of this work. I am very grateful to my supervisor Assoc/Prof Natasha Boland who was extremely approachable; Natasha’s wealth of experience in Operations Research was invaluable with many technical details. The client was a constant source of information and I thank them for allowing me the opportunity to do research with the company. Thanks to Mr. Tom Beadle for the time he spent on the project particularly in compiling data. Lastly to my fellow students who I have learnt a great deal from, it has been a pleasure to work with everyone.
Chapter 1

Introduction

In this thesis we introduce The Tour Leader Scheduling Problem (TSLP) a new combinatorial problem, closely related to existing problems such as Home Health Care, Vehicle Routing, Set Covering, Set Packing and others. It is essentially a network flow combinatorial problem. The motivation for this research is to optimize human resource planning operations for a company that operates holiday tours around the world. The associated personnel, tour leaders, are considered the most expensive resource for the company. Effective labor scheduling can reduce the cost of the human resources, improve customer service and increase employee satisfaction. Numerous approaches have been published in the literature for dealing with different versions of workforce scheduling problems; a lot of important techniques were gained from the literature review. Nevertheless no approach has dealt specifically with the TLSP, thus most of the constraints and costs of the problem have been developed without any specific references.

The client is a tour guide operator operating tours (trips) on a global scale, however the focus of this research is on the European operations, covering regions of Italy, France and Eastern Europe. In theory the optimization models developed could be applied to other continents.

The problem is to create a schedule for the season. A schedule is a complete list of each trip offered by the client with an assigned leader to guide that trip. The trips are set a-priori. A trip has a start city and an end city. The trips are uniquely categorized by a code, for example AKR060513, which embeds the trip type (AKR) and start date (13/05/06). The trip type uniquely specifies the start and end cities. More than one trip of the same trip type is completed during the season. The start date uniquely defines a particular trip, AKR060513 starts on the 13th of May 2006. There are no two trips with the same trip type that start on the same day.
The hard constraints on the problem are that a leader must be qualified to take a particular trip. To be qualified the leader must have previous experience of the trip or be trained pre season. Once a leader is assigned a particular trip they must finish it. Every trip must be covered by exactly one leader. The work load defined as \( \frac{\text{Days worked on a route}}{\text{Total days elapsed on a route}} \) cannot exceed some given percentage which may be mandated by the company or stipulated by the individual tour leader. A route is an ordered collection of trips done by a single leader.

The soft constraints are that a personal desired work load for each leader should be met. The number of days worked in a row is bounded (block time). The number of days worked in a row is defined as the days worked on a route such that all breaks are less than some given number of days. For operational reasons the number of leaders used in a schedule should be minimized.

The optimization is to determine a schedule at minimum cost for the company subject to the specified hard and soft constraints. A transfer cost is incurred to the company if a leader is assigned a trip that starts in a different city from where they finished the last trip. A training cost is incurred to the company if a leader is assigned a trip type in a schedule that they are not qualified for.

The current practice of the client is to use basic heuristic methods and experiences from the prior year’s schedules. This thesis produces an optimal cost schedule subject to the constraints by applying integer programming techniques. An integer program computational solution is deemed infeasible if the computation time to optimal is larger than the time needed by a skilled person to create a schedule; estimated to be about 24 hours.

A recurring theme throughout this thesis is that of iteration. First the problem is modeled incorporating all factors in a complete model, and then consideration is given to the computational efficiency of such a model. The problem is then decomposed using the idea of Dantzig-Wolfe Decomposition and solved using heuristic methods. An analysis of how sub-optimal the heuristic methods were is found by using column generation, and comparing small time horizons of a heuristic schedule to the complete model.

Similar model requirements are used in the Airline Crew Rostering (ACR) problem as in the TLSP. Figure 1.1 presents some of the factors involved in producing a schedule. A similar diagram can be found in [17].
Figure 1.1: Factors of the Tour Leader Scheduling problem
1.1 Related Problems

Nurse Scheduling and Home Health Care

An exhaustive literature search was done to find any relevant problems previously studied with the same properties as the TLSP. No papers were found that had treated this exact problem. However, the closely related problem of Home Health Care (HHC) was discovered (the problem is $\mathcal{NP}$-Hard [8]). The problem is described in [5]. There exists a set of nurses and a set of patients who are to be visited in their home. The problem is to determine the best route for each nurse to visit a subset of patients such that each patient is visited.

The problem has both a vehicle routing component and a number of hard and soft constraints between assigning nurses to patients. The vehicle routing problem arises since many different routes are possible which cover all patients. The important component is that each patient must be visited exactly once. This is analogous to the tour leader scheduling problem where each trip must be visited exactly once. A sample of soft constraints in the HHC problem from [5] are that patients prefer certain time intervals for being served, the right chemistry between patients and nurses should be ensured and the staff satisfaction concerning work load and work time should be maximized. Work load with penalty constraints are used extensively in the TLSP. The major difference between the TLSP and HHC is that in the TLSP no consideration is given to any preference between the items to be visited and the visitors.

Nurse Scheduling is described in [12] as assigning nurses to shifts subject to the skill level of nurses working each shift, working relationships, personal schedule preferences, avoidance of shift patterns that might adversely affect health and the arrangements of shifts from the previous scheduling period. The nurse scheduling problem could be used as an extension of TLSP since it accommodates nurses being able to train during a schedule [12].

Rostering

Personnel rostering concerns the best way to assign the human resources to jobs over some scheduling period. This is researched extensively in the literature in the area of ACR. Human resources are important because airlines are usually dealing with large sums of money, so a small fractional increase in efficiency results in a relatively large saving to the bottom line of the company. The ACR problem is closely related to Tour Scheduling, [2], that deals with both work hours of the
day and work days of the week when assigning personnel to jobs. In [17] the components of a crew rostering system for the airline industry are rest times, days off, qualification rules, costs, crew bids, robustness, training and many other considerations. The TLSP involves all of these components. The ACR problem was used as a guide as to how large integer programs of this nature might be solved. Most papers reviewed used some form of column generation.

To solve the ACR problem, column generation is often used. In the master problem a set partitioning type problem is solved to select exactly one roster for each crew member such that the demands of the activities are met and the objective (minimize cost) is optimized [17]. This is analogous to the TSLP where leaders are assigned to at most one route such that every trip is covered. Since it is not possible to enumerate all possible rosters, the master problem is always defined on a subset of all possible rosters. A subproblem is solved where a large number of legal rosters are generated. This idea is used in the TLSP with the subproblem generating legal routes. The important idea for the TLSP is that of column generation and the iteration between the subproblem to make feasible routes and the master problem; assigning leaders to routes.

1.2 The Network

Figure 1.2 shows how the TLSP was modeled. Each trip is represented by a node. Feasible trip pairs are shown by the arcs. Each arc has a weight consisting of a transfer cost. Each node has a weight corresponding to the training cost. The network arcs are created as in Section 4.1. The network exhibits the following properties; it is directed, acyclic and conservative (all weights are positive). The days off between successive trips can be visualized as the arc from WSK060508 to WSU060525 where there is a break of 4 days. An optimal cost schedule for this small set of trips is shown in bold. Table 1.1 presents information on the optimal routes created.

<table>
<thead>
<tr>
<th>Leader</th>
<th>Route</th>
<th>Work Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daniel</td>
<td>WSW060114 WSW060128 WSW060211 ASK060302</td>
<td>89%</td>
</tr>
<tr>
<td>Katalin</td>
<td>ARK060422 WSY060507 ASP060528 WRI060605 WRI060619 ASP060703</td>
<td>87%</td>
</tr>
<tr>
<td>Ashley</td>
<td>WSK060508 WSU060525 ZSS060603</td>
<td>88%</td>
</tr>
</tbody>
</table>

Table 1.1: The optimal schedule for the trips shown in Figure 1.2

1 An extension of the models presented here would be to include crew bidding.
Figure 1.2: Example network used to model the TLSP
Chapter 2

Related Models

2.1 Linear Programming and Network Flows

The Integer Programming model of the TLSP is based on a network flow. [4] provides the foundations to the underlying structure of such a model, in particular the conservation of flow in a network when finding a shortest path (minimum cost for the TLSP) corresponding to one route. The TLSP must cover all trips. This is done by creating paths in the network. The optimal solution is to have the routes done at minimum cost, subject to hard and soft constraints. The basic flow equations are defined in [4] to be

\[
\min \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} X_{ij}
\]

subject to

\[
\sum_{j=1}^{m} X_{ij} - \sum_{i=1}^{m} X_{ij} = \begin{cases} 
1, & \text{if } i = s \\
0, & \text{if } i \neq 1 \text{ or } m \\
-1, & \text{if } i = m
\end{cases}
\]

where \( s \) is the first node and \( m \) is the last node of a shortest path, and \( X_{ij} = 1 \) if arc \( i \to j \) is included in the path, 0 otherwise and \( c_{ij} \) is cost on arc \((i, j)\). This model is equivalent to sending one unit of flow through the network at minimum cost. In the TLSP there is a cost associated with travel between any pair of nodes, \( c_{ij} \).
Due to the structure of the network which is acyclic in the TSLP there must be more than one route created to cover all trips. The arcs of the network are formed when it is possible for a leader to do consecutive trips. Obviously a leader cannot do a trip that starts before another trip ends due to the constraint that once a leader starts a trip they must end that trip.

The integer model has some interesting properties involving the use of the simplex algorithm, refer to Section 3.1. The matrix $A$ in the linear program is the node-arc incidence matrix [4], it can be shown that this matrix is unimodular. Unimodality implies that if the constraint $X_{ij} = 0$ or 1 were replaced by $X_{ij} \geq 0$ and if an optimal solution exists then the simplex method would still obtain an integer basic feasible solution where the value of each variable $X_{ij}$ is 0 or 1.

2.2 Set Covering

Set covering is the term used to describe a mathematical model where the aim is to find a set of sets whose union has all members of the union of all sets [6]. Usually the set covering problem is to find a set at minimum cost. A visualization of the problem is shown in Figure 2.1. Each region $\{1, \ldots, 13\}$ can contain a center, the constraint is that each region must be covered by a center. In this example the covering rule is if a center is located in a region it can cover all regions adjacent to it. Moreover there is some cost function incurred when a center is created in a region. A plausible set covering for this example is to create centers in regions $\{2, 9, 12\}$. Set covering can be thought of as a pure combinatorial problem [24]. Given a set $S$ of regions and a set of potential centers $P$, let $A_j \subseteq S$ be the regions that can be covered by a center at $j \in P$ with $c_j$ the cost incurred for operating that center. The combinatorial problem is then

$$
\min_{T \subseteq P} \left\{ \sum_{j \in T} c_j : \bigcup_{j \in T} A_j = S \right\}
$$

This problem can also be modeled as an Integer Program. The incidence matrix is $a_{ij} = 1$ if $i \in A_j$ and 0 otherwise. The decision variables are $x_j = 1$ if center $j$ is selected and 0 otherwise. The constraints are that at least one center must service region $i$ creating constraints with the RHS $\geq 1$. The integer program is

$$
\min \sum_{j \in P} c_j x_j
$$
Figure 2.1: Visualisation of a Set Covering Problem
\[
\begin{align*}
\text{s.t.} & \quad \sum_{j \in \mathcal{P}} a_{ij} x_j \geq 1 \quad \forall i \in \mathcal{S} \\
& \quad x_j \in \{0, 1\} \quad \forall j \in \mathcal{P}
\end{align*}
\]

The combinatorial flavor of the set covering problem leads to an important result. The number of subsets to consider is \(2^{|\mathcal{P}|}\) from [24]. In Figure 2.1, suppose that a center could be created in each region then the number of possible subsets is 8192. There is not yet known an efficient algorithm to solve the set covering problem. The set covering problem is proved to be \(NP\)-complete in [11]. There are however a number of heuristics developed for this problem most commonly the greedy heuristic [9].

In the TLSP set covering constraints are used in the master problem of column generation, Chapter 6. Where the trips are analogous to the set \(\mathcal{S}\), the leaders to set \(\mathcal{P}\) and \(A_j\) is the set of trips that can be covered by leader \(j \in \mathcal{P}\) with cost \(c_j\). Since the objective function is to be minimized this will drive down the number of leaders assigned to a route to be 1. The constraint is relaxed so that the reduced cost can be easily calculated.

### 2.3 Set Packing and Partitioning

The set packing problem is very like set covering but in this case each set element must appear in at most one subset. In this case, the constraints of a linear program have a RHS of \(\leq 1\). Set packing problems arise in partitioning applications, where the need is to partition elements under strong constraints on what is an allowable partition. The key feature of partitioning problems is that no elements are permitted to be covered by more than one set. A combinatorial optimization problem which is \(NP\)-hard, is to find the largest number of mutually disjoint subsets that cover a given set of sets [6].

To model set packing, let some universal set consist of all edges \(E\) of a graph \(G = (V, E)\), and \(S_i \subseteq E\) consist of all edges incident on vertex \(V_i\). Any set packing corresponds to a set of vertices \(v\) with no edge in common, in other words, an independent set \(\bigcap_{i \in v} V_i = \emptyset\). In Figure 2.2 the set of vertices \(V = \{1, 4, 6\}\) gives a set packing of the edges in the graph. In the TLSP the set packing constraint arises often. In the complete model there is a constraint that specifies each leader can do one first trip on a route or none at all. The set of leaders used is analogous to the set \(v\) and the set of first trips (only one) is the arc \(S_i\), the universal set \(E\) is the set of all trips. In the decomposition model a set packing constraint is that each trip must be entered once or not at all.
Figure 2.2: A graph $G$ depicting the use of set packing
The set partitioning problem arises when each set element must appear in exactly one subset. In this case, the constraints in the linear program have a RHS of $= 1$. A combinatorial definition of set partitioning is consider some set $S$. There exists a set of sets $P$ such that $X_i \subseteq S$ and $P = \{X_1, \ldots, X_i\}$. Then $P$ is a partition if

$$X_i \neq \emptyset \quad \forall i$$

$$\bigcup_{i} X_i = S$$

$$X_i \cap X_j = \emptyset \quad \forall i, j$$

The set partitioning problem is known to be $NP$-complete and there is unlikely to exist an efficient algorithm [15]. However there are heuristics that exist and can be found in [15]. Another approach is the use of column generation [19].

This form of constraint arises often in the TLSP. In the complete model there is a constraint that specifies every trip must be visited by any leader exactly once. In the decomposition model every trip must be left exactly once in the subproblem and every route created must be assigned to exactly one leader.

### 2.4 Auxiliary Variables

Auxiliary variables are often used to keep a record of some computation, and, thereby, allow reasoning over the entire computation history [20]. In [23] auxiliary variables are used in the Traveling Salesman Problem (TSP) to eliminate sub tours. They are employed to count the number of times a city has being visited on a tour. For each city $i \in \mathcal{N}$ there is an auxiliary variable $t_i$ which counts the number of cities visited up to city $i$. The constraint is defined such that each time the tour visits a city the counter is updated. Using a decision variable $x_{ij} = 1$ if city $j$ is visited directly after city $i$, 0 otherwise and $M$ a suitably large constant, we can model the counter increment using the following linear constraint.

$$t_j \geq t_i + 1 - M (1 - x_{ij}) \quad \forall i \in \mathcal{N}, \ j \in \mathcal{N}$$

The same idea is used in the TLSP. The constraint is not to eliminate sub tours but rather uses the auxiliary variables to eliminate too many days worked in a row by a leader (block time). The constraint counts how many days have been worked in a row without encountering a break of 6 or
more consecutive days, and records this value in an auxiliary variable, then constrains the auxiliary variable to be less than 70.
3.1 Simplex Algorithm

The simplex algorithm is a method for solving problems in linear programming. It is researched since it is used in almost all commercial solvers including Xpress-MP that incorporate branch and bound. A more efficient algorithm is the interior point algorithm but is not usually used in conjunction with branch and bound. The simplex algorithm was invented by George Dantzig in 1947, and tests adjacent vertices of the feasible set (which is a polytope) in sequence so that at each new vertex the objective function improves or is unchanged. The complexity of the simplex algorithm in practice is very efficient, generally taking $2m$ to $3m$ iterations at most (where $m$ is the number of equality constraints), and converging in expected polynomial time, [21]. However, its worst-case complexity is exponential, as can be demonstrated in [16].

A linear program in standard form is

$$
\begin{align*}
\text{optimize} & \quad c^T x \\
\text{s.t.} & \quad Ax = b
\end{align*}
$$

(3.1)

where $A \in M_{m \times n} (\mathbb{R}), c \in \mathbb{R}^n, b \in \mathbb{R}^m$ and $x$ is an $n$ dimensional vector of variables.

A basic solution to the set of constraints $Ax = b$ is obtained by setting $n - m$ variables to 0 and solving for the remaining $m$ variables. Any basic solution in which all variables are nonnegative is defined to be a basic feasible solution.
From \( n \) variables a set of \( m \) basic variables can be chosen in \( \binom{n}{m} \) ways. Since it is possible that all basic solutions may be basic feasible solutions the maximum number of basic feasible solutions is \( \binom{n}{m} \).

Some of the important theorems from linear programming are now given.

**Theorem 1 (From [4])** The collection of extreme points of the polytope corresponds to the collection of basic feasible solutions, and both are nonempty, provided that the feasible region is not empty.

**Theorem 2 (From [4])** Assume that the feasible region is nonempty. Then a finite optimal solution exists iff \( cd_j \geq 0 \) for \( j = 1, 2, \ldots, l \) where \( d_1, \ldots, d_j \) are the extreme directions of the feasible region. Otherwise the optimal solution value is unbounded.

**Theorem 3 (From [4])** If an optimal solution exists, then an optimal extreme point (or equivalently an optimal basic feasible solution) exists.

**Theorem 4 (From [4])** A linear programming problem having a finite optimal value has an optimal solution at an extreme point.

Using Theorems (1)-(4) it is only necessary to enumerate all possible basic feasible solutions and choose the one having the optimal objective value. This is not satisfactory since the number of possible basic feasible solutions for small problems is very large since \( \binom{n}{m} = \frac{n!}{m!(n-m)!} \). This approach also does not tell us if the problem is unbounded.

The simplex algorithm only searches a subset of the extreme points and tells us if the objective value is unbounded. A summary of the procedures used in the simplex algorithm for a maximization problem in standard form Equation (3.1) as outlined by [23] is as follows:

**Step 1** If all nonbasic variables have nonnegative coefficients in row 0 (the objective function), then the current basic feasible solution is optimal, leave the algorithm.

**Remark:** This coefficient is known as the reduced cost. It is the amount by which the objective value will change by increasing the nonbasic variable by 1. Formerly in linear programming the reduced cost of a variable \( x_j \) is \( c_j - \Pi^T A_j \) where \( \Pi \) is the vector of duals corresponding to each constraint [14].
Step 2 Choose the variable with the most negative reduced cost in row 0 to enter the basis. This is called the entering variable.

Step 3 For each constraint in which the entering variable has a positive coefficient, compute the ratio \( \frac{\text{Right Hand of Constraint}}{\text{Coefficient of entering variable}} \). Any constraint attaining the smallest ratio is the winner of the ratio test.

Step 4 Use elementary row operations or similar to make the entering variable the basic variable in any row that wins the ratio test.

Remark: This is called pivoting on the row that wins the ratio test.

Step 5 Return to Step 1

The key to the simplex method is recognizing the optimality of a given extreme point based on local considerations without having to explicitly enumerate all possible basic feasible solutions [4]. In some special cases the optimal solution obtained by simplex corresponds to an optimal integer solution, however most often this is not the case, and simply rounding variables up and down does not necessarily correspond to the optimal integer solution.

### 3.2 Branch and Bound

The branch and bound algorithm is the exact method for finding solutions of an integer program. It is used in conjunction with the simplex algorithm to find the optimal integer solution. It is researched since most commercial solvers use this method including Xpress-MP. The first paper presenting a branch and bound algorithm to an integer program problem was [18].

Consider a maximization problem in Equation (3.1) with additional constraints that \( x \in \mathbb{Z}^n \) from [24].

Step 1 Solve the linear relaxation of the problem using the simplex method to obtain an upper bound on the integer solution \( \bar{z} \) and a solution \( \bar{x}_n \). If the solution is integer \( (x \in \mathbb{Z}^n) \), then the optimal relaxation is integer and we are done. Otherwise set \( \bar{z} = -\infty \)

Step 2 Now we need to create two new subproblems by branching on a single fractional variable. The choice of which variable to choose is arbitrary; [24] suggests taking the most fractional variable
in the set $C$ of variables.

$$\arg \max_{j \in C} \min \{ f_j, 1 - f_j \} \quad (3.2)$$

where $f_j = x_j^* - \lfloor x_j^* \rfloor \quad (3.3)$

so that a fractional value of $\frac{1}{2}$ is best.

Then if $x_j = \pi_j \notin Z^1$ split the problem into two subproblems with the following constraints

$$S_1 = S \cap \{ x : x_j \leq \lfloor \pi_j \rfloor \} \quad (3.4)$$

$$S_2 = S \cap \{ x : x_j \geq \lceil \pi_j \rceil \} \quad (3.5)$$

Then $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$.

**Step 3** We now have two more subproblems currently active. A choice must be made as to which active subproblem to choose. Again [24] provides some insight. Choices are Depth First Search and Best-Node First.

**Remark:** In most commercial solvers a mixture of both these ideas is used. A default strategy for a problem is already known by a solver, it does this by testing on hundreds of different problem instances [24].

The problem now is to re-optimize a chosen subproblem $i$, with the added constraint of $S_i$. Since we have only added one upper or lower constraint the optimal basis remains dual feasible. [24] says that typically only a few pivots are needed to find the new optimal solution $\pi^i$, $\pi^i_n$. At this stage pruning of the subproblem can take place.

- If $\pi^i$ with corresponding solution $x \in Z^n$ is larger than $\bar{z}$ then reset $\bar{z} = \pi^i$. Otherwise prune by optimality.

- If $\pi^i$ with corresponding solution $x \notin Z^n$ is larger than $\bar{z}$ then keep subproblem $i$ as active. Otherwise prune by bounding arguments.

- If the feasible region for subproblem $i$ is empty, prune by infeasibility.
**Step 4** Choose a new active subproblem and repeat branching until there are no more active subproblems.

Branch and bound is a very effective way of finding integer solutions. However in very large integer programming problems like the TLSP, the system can easily run out of memory keeping track of all the possible active subproblems. Also branch and bound algorithms can fail because the bounds obtained from the linear relaxations are too weak [24]. So the tightening of the formulation is crucial. This leads to column generation which *convexifies* part of the feasible region.

### 3.3 Column Generation & Dantzig-Wolfe Decomposition

Column generation was devised by Gilmore and Gomory in 1961 to solve large scale LP’s [23]. The classical example of a column generation scheme is the cutting stock problem. Additionally, column generation has been applied to many problems such as crew scheduling and vehicle routing. Consider a standard integer program formulation

\[
\begin{align*}
\min & \quad cx \\
\text{s.t.} & \quad Ax \geq b \\
& \quad x \in X \\
& \quad x \text{ integer} \\
\end{align*}
\]  

At the heart of column generation is the idea that the set \(X^* = \{x \in X : x \text{ integer}\}\) is represented by a finite set of vectors [3]. Moreover if \(X\) is bounded then \(X^*\) is just a finite set of points, say \(X^* = \{x^i | i \in I\}\). Further if \(x\) is binary then \(X^*\) coincides with the extreme points of its convex hull, \(\text{conv}(X^*)\). Since representing a bounded polyhedron by its extreme points is the basic premise of the Dantzig-Wolfe decomposition, the integer column generation approach is closely related [3].

Given \(X^* = \{x^i | i \in I\}\) and the fact that any point \(x \in X^*\) can be represented as

\[
x = \sum_{i \in I} x_i \lambda_i
\]

\[
\text{s.t. } \sum_{i \in I} \lambda_i = 1
\]

\[
\lambda_i \in \{0, 1\}
\]

21
Then the Dantzig-Wolfe decomposition of the problem in (3.6) is

\[
\begin{align*}
\text{min} & \quad \sum_{i \in I} (cx^i) \lambda_i \\
\text{s.t.} & \quad \sum_{i \in I} (Ax^i) \lambda_i \geq b \\
& \quad \sum_{i \in I} \lambda_i = 1 \\
& \quad \lambda_i \in \{0, 1\}
\end{align*}
\] (3.10)

The essential difference between (3.6) and (3.10) is that \(X^*\) has been replaced by a finite set of points. The Dantzig-Wolfe Decomposition in (3.10) is now called the Global Master Problem. The problem with this formulation is that the set \(I\) is very large.

**Remark:** In the TLSP the set \(I\) contains all possible feasible routes, which is \(O(10^{203})\). It is thus impractical to enumerate all routes.

Formulation (3.10) can however now be solved by Branch & Price where at each node the linear relaxation of the problem is solved by column generation. This then creates a new formulation where the master problem is not taken over all columns, but some subset that is favorable. This is called the restricted master. The restricted master is defined at iteration \(k\) to be as follows where \(I' \subset I\).

\[
\begin{align*}
\text{min} & \quad \sum_{i \in I'} (cx^i) \lambda_i \\
\text{s.t.} & \quad \sum_{i \in I'} (Ax^i) \lambda_i \geq b \\
& \quad \sum_{i \in I'} \lambda_i = 1 \\
& \quad \lambda_i \in \{0, 1\}
\end{align*}
\] (3.11)

When solving the LP relaxation of the global master problem, one first solves a restricted master problem and then seeks a favorable column to be entered into the basis. The favorable column is determined by the largest reduced cost (negative in a minimization problem). The reduced cost is defined as \((c - \Pi_1 A)x^k - \Pi_2\) which involves the duals of the master problem constraints. The subproblem is to solve

\[
\min_{i \in I} (c - \Pi_1 A)x^i = \min_{x \in X} (c - \Pi_1 A)x
\]
where $\Pi_1$ is the dual value associated with the joint constraints of the restricted master. This will produce an optimal solution $x^k$, the column generation process is then to append solution $k$ to $I'$ if $(c - \Pi_1 A)x^k < \Pi_2$ where $\Pi_2$ is the dual value associated with the convexity constraints [24].

A stopping criterion for a minimization problem is if the best reduced cost returned is $\geq 0$

The master problem must be initialized with some feasible solution. This is done depending on the problem. In the TLSP this is done by using a simple decomposition of the problem to find a feasible set of route solutions to enter into the master problem.
Chapter 4

Complete Model

The complete model is the full mathematical description of the problem. The ideal situation would involve the model solving. However, TLSP is hard and thus will only provide a platform to use as a comparison to other models. In Section 4.3 we discuss the numerical results from the model, although mathematically correct the model will ultimately fail due to the massive number of decision variables that are created.

The underlying principles of the model are set covering, set packing, and shortest paths all of which have been reviewed in Chapter 2. The structure used to model the TLSP is a network flow, as shown in Figure 1.2. The network is constructed in which every node represents a trip and every arc represents a possible pair of trips done by any leader. Thus the theoretical worst case node-arc incidence matrix [4] for the network has dimensions $M_{O(10^2) \times O(10^4)}$. The network arcs built to model the problem due to practical constraints are discussed in Section 4.1. To produce a single route in the network, a single unit of flow is pushed from the first node to the last node, as in Chapter 2.

The problem is to find a set of routes that cover every trip only once at minimum cost, subject to hard and soft constraints as discussed in Chapter 1. The number of trips offered by the client is 675 the number of available leaders (routes) is 51.
4.1 Network Model

Parameters

\[ s_i = \text{Start date of trip } i \]
\[ e_i = \text{End date of trip } i \]
\[ t_{ij} = \text{Number of days between end trip } i \text{ and start of trip } j \ (= s_j - e_i) \]
\[ k_{ij} = \text{Transfer time needed between trip } i \text{ and trip } j \]
\[ \text{MAXGAP} = \text{Maximum allowable days between any trip pair} \]

The network is built on the basis of two hard constraints. The first is a practical constraint Equation (4.1) that stipulates the transfer time needed between the end of trip \( i \) and the start of trip \( j \) is at least \( k_{ij} \), where obviously \( k_{ij} \geq 0 \). This produces the acyclic nature of the network which is utilized in many of the models. The second is stipulated by the client as a maximum number of days off between successive trips (trip pairs). Equation (4.2) stipulates that the days off must be less than some \( \text{MAXGAP} \).

\[ t_{ij} \geq k_{ij} \quad (4.1) \]
\[ t_{ij} \leq \text{MAXGAP} \quad (4.2) \]

The client has supplied data on the two constraints (refer to Appendix A for a discussion on the data used.). The maximum allowable days off between any two trips is 14 days, this is the current practice of the client. It is so that the leaders are not under worked. This is an important parameter in the model, since it directly affects the solution, but more importantly the feasibility of solution. For example if the \( \text{MAXGAP} \) is set too low in comparison to the length of the season then it would be impossible for all the trips to be covered by the limited number of leaders available making an infeasible problem.

The transfer time \( k_{ij} \) needed between any two trip pairs is due to the travel time it takes to transport a leader between cities in Europe. The transfer time needed between any city pair is part of the data set. The transfer time of any trip pair is dependent on the transfer time between the end city of one trip and the start city of the next trip. For each trip code there is a start and end date, embedded in the trip code is a trip type code, within each trip type code there is a start and end city.
The network is constructed such that a set \( \mathcal{A} \) is created that possesses all possible trip pairs in the network, (all arcs in the network).

Sets

- \( \mathcal{T} \) is the set of all Trips
- \( \mathcal{A} \) is the set of all possible Trip Pairs

The set \( \mathcal{A} \) is defined by

\[
\mathcal{A} = \{(i, j) \in \mathcal{T} \times \mathcal{T} : k_{ij} \leq t_{ij} \leq \text{MAXGAP}\} \tag{4.3}
\]

4.2 The Complete Model

Sets

- \( \mathcal{L} \) is the set of all Leaders
- \( \mathcal{H} \) is the set of all Trip Types

Variables

\[
X_{lij} = \begin{cases} 
1, & \text{if leader } l \text{ does trip } i \text{ followed by trip } j \\
0, & \text{Otherwise}
\end{cases}
\]

\[
Y_{li} = \begin{cases} 
1, & \text{if leader } l \text{ does trip } i \\
0, & \text{Otherwise}
\end{cases}
\]

\[
Z_{li} = \begin{cases} 
1, & \text{if leader } l \text{ does trip } i \text{ as the first trip in their route} \\
0, & \text{Otherwise}
\end{cases}
\]

\[
\hat{Z}_{li} = \begin{cases} 
1, & \text{if leader } l \text{ does trip } i \text{ as the last trip in their route} \\
0, & \text{Otherwise}
\end{cases}
\]

Parameters

\[
c_{ij} = \text{Transfer cost between end trip } i \text{ and start of trip } j
\]
4.2.1 Transfer Cost

Objective Function

\[
\min \sum_{l \in L} \sum_{(i,j) \in A} c_{ij} X_{lij} \tag{4.4}
\]

The objective function is to be minimized. The transfer cost involves the transportation of a leader from the end of one trip to the start of the next trip. The cost arises when a leader ends a trip in a city that is not the start city of their next trip, thus a cost will be incurred by the client to transport that leader to the next trip’s start city. The make up of the cost between any two cities will include one or more of air, rail and bus tickets as well as other miscellaneous travel expenses specific to city pairs. Some cities are much more expensive to fly out of than into; so the data includes information on travel costs such as Rome to Venice and Venice to Rome. The cost \(c_{ij}\) can be visualized as the cost on the arcs of the network.

Constraints

The objective function is to be minimized subject to the following hard logic constraints.

If a leader \(l\) does a trip pair \(X_{lij} = 1\) then \(Y_{li} = 1\) and \(Y_{lj} = 1\) however if \(X_{lij} = 0\) then it is still possible for a leader to have done trip \(i\) or trip \(j\), it just means that the leader has not done them in a trip pair.

\[
X_{lij} \leq Y_{li} \quad \forall \ l \in L \ (i, j) \in A \tag{4.5}
\]

\[
X_{lij} \leq Y_{lj} \quad \forall \ l \in L \ (i, j) \in A \tag{4.6}
\]

Every trip in the network must be done by one leader exactly. This is a set partitioning constraint. In the network this means that each node must be visited exactly once only by any route.

\[
\sum_{i \in T} Y_{lk} = 1 \quad \forall \ k \in T \tag{4.7}
\]

Every leader can only start and finish on one particular trip. If a leader does not start a route at all then there will be no first and last trip for that leader. This is the case when there are excess leaders.

\[
\sum_{i \in T} Z_{li} \leq 1 \quad \forall \ l \in L \tag{4.8}
\]
\[ \sum_{i \in T} \hat{Z}_{li} \leq 1 \quad \forall l \in \mathcal{L} \quad (4.9) \]
\[ \sum_{i \in T} Z_{li} = \sum_{i \in T} \hat{Z}_{li} \quad \forall l \in \mathcal{L} \quad (4.10) \]

Every trip must be entered unless it is the first trip for a leader and every trip must be left unless it is the last trip for a leader. This constraint enforces the conservation of one unit of flow through a node in the network.

\[ Y_{lj} = \sum_{i \in T} X_{lij} + Z_{lj} \quad \forall l \in \mathcal{L}, j \in \mathcal{T} \quad (4.11) \]
\[ Y_{li} = \sum_{j \in T} X_{lij} + \hat{Z}_{li} \quad \forall l \in \mathcal{L}, i \in \mathcal{T} \quad (4.12) \]

### 4.2.2 Training Cost

The leaders have their own individual set of trip types that they are qualified for. This model utilizes that information to implement a training cost incurred for a leader to do a particular route. The cost is only incurred once for each trip type on a route, even if more than one trip type is done. The criterion for qualification of a leader is discussed in Chapter 1. In the instance where a leader is assigned a trip type and they are qualified for that trip type then no cost is incurred. A cost specific to each trip type is incurred if a leader is not qualified pre season to do that trip type. This is a cost incurred by the client pre season to train a leader to forfill the created schedule. This model naturally favors leaders who are well qualified, by seeking to assign them more trips.

In this model a new binary variable is needed. It captures the logic that a route can have many repeats of a trip type, but for a particular leader the training cost should only be incurred once for that trip type if they are not qualified. This formulation is stronger than penalizing every trip done since it is more realistic.

**Variables**

\[ P_{lk} = \begin{cases} 
1, & \text{if leader } l \text{ does trip type } k \text{ once or more on a route} \\
0, & \text{Otherwise} 
\end{cases} \quad (4.13) \]

**Parameters**

\[ T_k = \text{Training cost incurred for trip type } k \]
\[ Q_l = \text{The set of all trip types that leader } l \text{ is qualified for} \]

\[ \theta_k = \text{Trip type of the trip } k \]

**Objective Function**

\[
\min \sum_{l \in L} \sum_{(i,j) \in A} c_{ij}X_{tij} + \sum_{l \in L} \sum_{k \in H \setminus Q_l} T_k P_{lk}
\]  

(4.14)

**Constraints**

\[ P_{\theta_l} \geq Y_{li} \quad \forall \ i \in T \ \forall \ l \in L \]  

(4.15)

### 4.2.3 Work Load Constraints

The desired work load stipulated by a leader is considered as an upper bound on a route work load for a leader \( l \). This is modeled as a hard constraint, since the client is bound to produce no routes with a work load of \( \geq 90\% \). The client stated that each leader should be able to request their own work load pre season. It was decided not to impose a linear penalty on the difference between the desired work load and the actual created work load for a leader. The work load preference is simply a preference and not actually a real cost to the client. Moreover if a penalty was introduced then the optimization would be making a compromise between actual costs incurred and a penalty term.

A hard constraint does have implications on the infeasibility of solution. This was particularly prevalent with small \( |T| \) because the likelihood decreases that a trip pair exists with a suitably lengthy break to achieve a work load on route less than 90%.

**Parameters**

\[ d_l = \text{Individual desired work load for leader } l \]

**Constraints**

The work load for a leader \( l \) is defined as

\[
\frac{\text{Days Worked on a route}}{\text{Total days elapsed on a route}}
\]
and we require that the work load is \( \leq d_l \). This can be expressed in another form since

\[
\text{Days Worked on a route} = \text{Total days elapsed on a route} - \text{Days Off on a route}
\]

then this gives a new expression for the work load as

\[
\text{Total days elapsed on a route} - \text{Days Off on a route} \leq d_l \times (\text{Total days elapsed on a route})
\]

so another way of constraining the work load is

\[
\frac{\text{Days Off on a route}}{\text{Total days elapsed on a route}} \geq (1 - d_l).
\]

The total days elapsed on a route is the end date of the last trip for that leader minus the start of the first trip for that leader. The ratio of days off to days elapsed is then multiplied out so as to become a linear constraint. The term \( \hat{Z}_{lk}e_k - Z_{lk}s_k \) is the total days elapsed on a route for a leader \( l \).

\[
\sum_{(i,j) \in A} X_{ij}t_{ij} \geq (1 - d_l)(\sum_{k \in T} \hat{Z}_{lk}e_k - \sum_{k \in T} Z_{lk}s_k) \quad \forall \ l \in \mathcal{L}
\]  

(4.16)

**4.2.4 Auxiliary Variables to Constrain Consecutive Days Worked**

Part of the scope on the problem was to ensure that no leader worked more than 70 consecutive days in a row. The definition of consecutive days worked in a row is such that there is no break of more than 6 days. This is a constraint imposed by the client due to occupational health and safety issues. This constraint is modeled by creating auxiliary variables that capture the days worked without a suitably lengthy break for each leader. The parameters here were determined from discussions with the client to be 70 days consecutively and then a break minimum of 6 days. The idea of using auxiliary variables is taken from [23] in the TSP problem, and was reviewed in Chapter 2.

**Variables**

\[ E_{li} = \text{Days accumulated without a suitable length break for leader} \ l \ \text{at end of trip} \ i \]  

(4.17)

**Parameters**

\begin{align*}
\text{DaysLimit} &= \text{Maximum number of days allowed accumulated} \quad 70 \\
\text{Break} &= \text{Length of suitable break} \quad 6 \text{ days}
\end{align*}

(4.18)  

(4.19)
The variable $E_{li}$ is defined so that an upper limit can be set for any leader $l$ at any trip $i$ to have worked $DaysLimit$ consecutive days. If the binary variable $X_{lij} = 1$ and creating the trip pair $(i \rightarrow j)$ for that leader $l$ would make $E_{lj} \geq DaysLimit$ then there is a contradiction between Constraints (4.20) and (4.21). The value of $E_{ij}$ is the sum of all the days worked in a row (including breaks) up until the end of trip $j$ such that there has been no break of $\geq 6$ days in the previous 70 days. This is an analogous constraint used in the elimination of sub tours in the TSP. The TSP problem using linear constraints is known to be very hard to solve (it is actually $NP$-Hard [24] refer to Chapter 2) thus it is predicted that this constraint will not give adequate computation time as discussed in Chapter 1.

A suitable size for the value of $BigM$ was chosen noting the following properties of the problem. $E_{li}$ can never be larger than 70 from Constraint (4.21), the largest possible break from the network structure is 14 days and the longest trip possible is 21 days. Thus a good first approximation is $BigM = 70 + 14 + 21 = 105$. The value of $BigM$ however could be tightened to a specific trip pair $(i,j)$, giving $BigM_{ij}$. The values of $E_{li}$ for trips that are in the first 70 days of the season will never be more than 70 days so the constraint is redundant for this subset of trips. The days defined by $e_j - e_i$ in Constraint (4.20) are the days between the end of trip $j$ and the end of trip $i$. The lower bound on the value of $E_{ij}$ is the length of trip $j$. Thus $BigM_{ij}$ was set to

$$BigM_{ij} = DaysLimit + (s_j - e_i)$$

(4.23)

So that when $X_{lij} = 0$ a lower bound on $E_{ij}$ is obtained as the value of $E_{li}$, which can only ever be a maximum of 70, plus the length of the trip $j$, Constraint (4.20) was improved to

$$E_{ij} \geq E_{li} + e_j - e_i - (70 + s_j - e_i)$$

$$= E_{li} - 70 + e_j - s_j$$

(4.24)

The computational effect of tightening the value of $BigM$ is tested in Section 4.3.4
The Complete Model

The mixed integer program for the complete model can now be defined as:

**Objective Function**

\[
\min \sum_{l \in L} \sum_{(i,j) \in A} c_{ij} X_{lij} + \sum_{l \in L} \sum_{k \in H \setminus \{q_l\}} T_k P_{lk} 
\] (4.25)

**Constraints**

\[
X_{lij} \leq Y_{li} \quad \forall \; l \in L, \; (i,j) \in A
\]
\[
X_{lij} \leq Y_{lj} \quad \forall \; l \in L, \; (i,j) \in A
\]
\[
\sum_{l \in L} Y_{lk} = 1 \quad \forall \; k \in T
\]
\[
\sum_{i \in T} Z_{li} \leq 1 \quad \forall \; l \in L
\]
\[
\sum_{i \in T} \hat{Z}_{li} \leq 1 \quad \forall \; l \in L
\]
\[
\sum_{i \in T} Z_{li} = \sum_{i \in T} \hat{Z}_{li} \quad \forall \; l \in L
\]
\[
Y_{ij} = \sum_{i \in T} X_{lij} + Z_{lj} \quad \forall \; l \in L, \; j \in T
\] (4.26)
\[
Y_{li} = \sum_{j \in T} X_{lij} + \hat{Z}_{li} \quad \forall \; l \in L, \; i \in T
\]
\[
P_{li} \geq Y_{li} \quad \forall \; i \in T, \; l \in L
\]
\[
\sum_{(i,j) \in A} X_{lij} \geq (1 - d_l)(\sum_{k \in T} \hat{Z}_{lk} e_k - \sum_{k \in T} Z_{lk} s_k) \quad \forall \; l \in L
\]
\[
E_{ij} \geq E_{li} + (e_j - e_i) - \text{BigM} (1 - X_{lij}) \quad \forall \; l \in L, \; \forall \; (i,j) \in A \quad \text{s.t.} \; \sum_{(i,j) \in A} (s_j - e_i) \leq \text{Break}
\]
\[
E_{li} \leq \text{DaysLimit} \quad \forall \; i \in T, \; l \in L
\]
\[
E_{li} \geq e_i - s_i \quad \forall \; i \in T, \; l \in L
\]

**4.3 Computational Analysis of Complete Model**

The computational results are carried out using the data sets provided by the client. The data is discussed in Appendix A. The computational output statistics from Xpress-MP such as the Gap are also discussed in Appendix A. The cardinality of the set $T$ used, $x = |T|$ in each computational analysis is the first $x$ trips offered in the season, ordered by their start dates. Table 4.1 defines each of the models tested.
### 4.2.1 Objective Function Equation

<table>
<thead>
<tr>
<th>Model</th>
<th>Objective Function Equation</th>
<th>Constraint Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.1</td>
<td>(4.4)</td>
<td>(4.5)-(4.12)</td>
</tr>
<tr>
<td>4.2.2</td>
<td>(4.14)</td>
<td>(4.5)-(4.12),(4.15)</td>
</tr>
<tr>
<td>4.2.3</td>
<td>(4.14)</td>
<td>(4.5)-(4.12),(4.15),(4.16)</td>
</tr>
<tr>
<td>4.2.4</td>
<td>(4.14)</td>
<td>(4.5)-(4.12),(4.15),(4.20)-(4.22)</td>
</tr>
</tbody>
</table>

Table 4.1: Definition of the models tested

### 4.3.1 Computational Analysis of MAXGAP

Since there are 675 trips (nodes in the network) the maximum number of possible trip pairs (arcs in the network) \(|\mathcal{A}| = O(10^5)\). Then there are 51 leaders who can take any combination of the possible trip pairs. So the total number of variables \(X_{ij}\) created is \(O(10^6)\) which is too large to solve in Xpress-MP. Refer to Appendix B for the mosel code used to create variables.

The first experiment with this model was to test what effect the value of MAXGAP had on the solution and the amount of variables created. This would also determine a lower bound on the value of MAXGAP so that the problem was still computationally feasible. A summary of results can be found in Table 4.2. Where 'Start Optimization' is 'Yes' if the problem’s linear program matrices can be loaded into Xpress-MP and 'No' otherwise. The 'Feasible Solution' is 'Yes' if the problem loads and can find a solution and 'Empty Memory' if the problem uses all the available memory on the computer without finding a solution. The only costs and constraints modeled in this experiment were the transfer costs.

<table>
<thead>
<tr>
<th>MAXGAP</th>
<th>Number Variables</th>
<th>Start Optimization</th>
<th>Feasible Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>283,509</td>
<td>Yes</td>
<td>Empty Memory</td>
</tr>
<tr>
<td>2</td>
<td>749,190</td>
<td>Yes</td>
<td>Empty Memory</td>
</tr>
<tr>
<td>4</td>
<td>1,055,190</td>
<td>Yes</td>
<td>Empty Memory</td>
</tr>
<tr>
<td>7</td>
<td>(\approx 1,739,100)</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>(\approx 2,328,300)</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>(\approx 3,131,000)</td>
<td>No</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.2: Computational results for varying MAXGAP using the full data set

From the construction of the network arcs (preprocessing of binary variables) it was established that a solution cannot be found by simply decreasing the MAXGAP to a certain level. This is because even when the level is decreased enough that the computer can actually create all the variables, the optimization uses all the available memory. It is predicted however that at a small level of MAXGAP the solution would probably be infeasible due to the fact that there are not enough leaders to do all the routes that would need to be created. Using this experiment it was
decided that since a solution cannot be gained for the whole data set for a $MAXGAP$ of 0, the best case would be to consider a smaller subset of trips and reset $MAXGAP = 14$.

We now test the cardinality of $\mathcal{T}$ that can be used to achieve a solution. The 'Optimal Solution' is the minimum cost that is required for the set $\mathcal{T}$ to be covered only considering transfer costs.

| $|\mathcal{T}|$ | Number Variables/Constraints | Start Optimization | Run Time (sec) | Optimal Solution |
|----------|-----------------------------|---------------------|----------------|-----------------|
| 10       | 2040/13628                  | Yes                 | 0.3            | 0               |
| 20       | 4335/47298                  | Yes                 | 1.0            | 0               |
| 30       | 8568/101368                 | Yes                 | 6.2            | 0               |
| 50       | 22797/250102                | Yes                 | 59.3           | 0               |
| 60       | 35904/361292                | Yes                 | 121            | 0               |
| 80       | 73134/661193                | Yes                 | 498.8          | 0               |
| 100      | $\approx 103,820/1,361,000$| No                  | Empty Memory   | -               |

Table 4.3: Computational results from varying $|\mathcal{T}|$

Table 4.3 shows that the maximum number of trips that can be considered at any one time using model 4.2.1 is of order 90 which is 13% of the total amount of trips offered in a season. The number of variables created in the program is within the working limit of Xpress-MP however the number of constraints grow very large and outside the capacity of Xpress-MP to optimize.

### 4.3.2 Computational Analysis of Training Cost

The addition of the training cost constraints obviously does not improve the computational performance of the program since more variables and constraints are being added. The performance is shown in Table 4.4. In this particular case the run time is highly dependent on the choice of the trips in $\mathcal{T}$, since a good data set (in terms of computational efficiency) will include a lot of trips that the leaders are collectively trained for, thus making it easy to assign trips to leaders. To overcome this potential anomaly the same data sets were used as in Table 4.3.

The poor computation time is due to Constraint (4.15). The variable $P_{lk}$ has no constraint that is forcing its value to 0, in other words it is unconstrained to take the value of 1. When $P_{lk} = 1$ then a training cost is potentially incurred to the objective function. Since none of the training costs are 0 this will increase the objective function if that leader is not qualified. Thus it is part of the optimization of the objective function to drive $P_{lj}$ down to 0 when leader $l$ does not do trip type $j$. The Constraint (4.15) must be constructed with this logic since we are using the variable $P_{lk}$ to keep some memory of what has happened on a leaders route. The simplex dual algorithm implemented by Xpress-MP finds a linear relaxation solution to the problem very efficiently, in the
order of a few seconds, however the integer solution using branch and bound is what increased the computational time.

| $|T|$ | Number Variables/Constraints | Run Time (sec) | Objective Value |
|-----|-----------------------------|----------------|-----------------|
| 10  | 3773/14189                  | 0.3            | 0               |
| 20  | 6164/48369                  | 1.4            | 0               |
| 30  | 10164/102949                | 3050.6 (Gap 26%) | 189           |
| 50  | 25556/262853                | 3340.4 (Gap 88%) | 693           |

Table 4.4: Analysis of varying $|T|$ for Model 4.2.2

Table 4.4 shows that for any $|T|$ the addition of a training cost to the model increases the run time. Constraint (4.15) has the effect of increasing the computational time by $\approx 40\%$ when the optimal objective value is 0. This is because the small $|T|$ includes no trip which a leader is not qualified for, thus the routes are done at zero cost. However when the data set $T$ is increased to include trips that no leader is trained for then the optimal solution will include training costs.

The solution obtained by model 4.2.2 with an increased data set had a significantly increased computational time $\approx 5000\%$. The $|T|$ that can be handled by Xpress-MP with the training cost included is, as stated before, determined by the actual trips that are in $T$. The full data set will certainly contain trips that some leaders will have to be trained pre season in order to take.

This computation involves no constraint on the work load of a route created. The optimal solution in model 4.2.1 is to assign a single trip to each leader making the transfer cost 0, when $|T| \leq 51$. The optimization in model 4.2.2 is making decisions on the relative importance of the travel cost incurred by a leader to the training cost incurred by a leader, such that all trips are covered. The optimal value obtained from the addition of the training cost is increased. This means that the training cost is an important parameter of the problem and should not be excluded in further models.

4.3.3 Computational Analysis of Work Load

To achieve a lower bound on the solution each leader’s work load preference was set to 0.9 the maximum allowable work load. This was done to increase the solution space to maximum, thus decreasing the chance of infeasibility of solution.

To analyze the difference in computation time between model 4.2.2 and 4.2.3 the exact same elements of $T$ must be used, and they should produce a feasible solution to get good results of the
difference in run time. Table 4.5 presents some computational results from varying $|T|$. 

| $|T|$ | Number Variables/Constraints | Run Time (sec) | Model 4.2.2 | Model 4.2.3 |
|-----|-------------------------------|---------------|-------------|-------------|
| 10  | 3676/11944                    | 0.3           | 1           |
| 20  | 6436/39952                    | 2.3           | 162.3       |
| 28  | 9364/84484                    | 3000 (Gap 26%)| 3000 (Gap 36%)|
| 40  | 13761/161352                  | 5000 (Gap 72%)| 5000 (Gap 84%)|

Table 4.5: Analysis of varying $|T|$ for Model 4.2.3

The computational results showed that the addition of the work load constraint significantly increases the run time. For $|T| = 28$ the Gap for model 4.2.3 is much larger than the Gap for model 4.2.2 at the same run time. The same is true for $|T| = 40$. This means that the addition of the work load constraint has a large impact on the run time of the program and consideration was given to whether this was a vital constraint. The infeasibility of the model in terms of run time had implications for the client as to how binding the work load constraint and preference system was.

The addition of the auxiliary variable which constrains the days worked in a row could provide good results on work loads created, without using Constraint (4.16) and lowering the risk of infeasibility.

### 4.3.4 Computational Analysis of BigM

The effect of tightening $\text{BigM}$ to $\text{BigM}_{ij}$ was tested in Table 4.6. It was determined that to get a good comparison between $\text{BigM}$ and $\text{BigM}_{ij}$ that a limit of 1500 sec be set for the run time of the program. If an optimal solution was not found then a comparison of computational efficiency was done using the value of Gap.

| $|T|$ | Season Length | # Variables/Constraints | Run Time (sec) | Objective Value |
|-----|---------------|-------------------------|----------------|----------------|
| 7   | 112           | 3287/6484               | 0.2            | 40             |
| 10  | 38            | 4672/8390               | 0.5            | 0              |
| 11  | 151           | 15750/164412            | 1500 (Gap 54%) | 518            |
| 16  | 175           | 6354/29647              | 1.8            | 69             |
| 25  | 197           | 9568/69283              | 11.6           | 89             |
| 35  | 214           | 13557/118914            | 1500 (Gap 58%) | 322            |
| 40  | 263           | 15750/164412            | 1500 (Gap 56%) | 496            |

Table 4.6: Analysis of tightening the value of $\text{BigM}$ for varying $|T|$ for Model 4.2.4

This showed that if the season length is less than 70 days then the computational time between using
BigM and $BigM_{ij}$ is the same, confirming that the constraint is redundant. The computational times for increased $|T|$ between $BigM$ and $BigM_{ij}$ is significant, confirming that tightening the bound on $BigM$ does decrease the computational time.

### 4.3.5 Computational Analysis of Auxiliary Variables

The effect of using model 4.2.4 to try and constrain the work load, produced better computational results than using model 4.2.3. For $|T| = 40$ model 4.2.3 ran for 5000 sec with a Gap of 84%, compared to model 4.2.4 which ran for 1500 sec with a Gap of 41%. Using this information and a visual check of the work loads created it was deemed that a suitable model in terms of work loads is achieved by using model 4.2.4. The trade off is that a leader could no longer specify their own personal work load.

### 4.4 Computational Infeasibility of Complete Model

The complete model had proven to be too large to solve for the full data set in Xpress-MP. The mathematical formulations in models 4.2.1 to 4.2.4 have covered all possible costs and constraints that are to be imposed on the problem by the client. A solution to the TLSP was sought by applying the idea of decomposition. The key to the decomposition of the problem is that it considered the leaders as being homogeneous. Decomposition and iteration is expanded further in Chapter 6.
Chapter 5

A Route Based Model & Heuristic

The optimization problem to create a schedule was decomposed as discussed in Chapter 3. This was to enable the optimization to take place with a reduced number of variables making the problem solvable. The decomposition involves independently creating routes and then assigning routes to leaders.

The route based model optimization best assigns the leaders to a given set of feasible routes by considering the leaders preferred work load and training cost. The first approach was to use heuristic methods to find good routes, thus making a schedule.

The use of route generating heuristics will not necessarily produce the optimal schedule that would have been found by the complete model. The lower limit (of the objective value) on any optimal solution found by the route based model is equal to the optimal value found by the complete model. This is because the heuristic routes are created with no consideration given to the leader’s qualifications or desired work loads, since the problem has being decomposed. So the heuristic routes cannot form a schedule with objective value strictly less than that found by the complete model.

An objective is to determine how sub-optimal the heuristic methods are in developing feasible routes. This is done by a comparison of small time horizons between the complete model and the route based model. At larger time horizons column generation is used. In Chapter 6 column generation is used to determine the optimal routes in a subproblem such that the master problem prices out favorably in terms of costs. Thus the master problem and the subproblem are connected and in theory could find the optimal objective value that would have been found by the complete
5.1 The Route Based Model

Given that we have some set of feasible routes the problem is to assign the leaders to the routes, completing the schedule. This is a trivial problem if the leaders are considered homogeneous in terms of their qualifications and desired work load. However the leaders in this model are considered qualified for a set of trip types and have a desired work load \( (d_l) \). The route based model uses the following notation.

**Sets**

- \( R \) is the set of all routes possible
- \( R_j \) is the set of trips on route \( j \)
- \( S_j \) is the set of trip types on route \( j \)

**Variables**

\[
U_{lj} = \begin{cases} 
1, & \text{if leader } l \text{ is assigned route } j \\
0, & \text{Otherwise} 
\end{cases} \tag{5.1}
\]

The set \( R \) is the set of all possible routes that can be created, this is a very large set of routes as discussed in Section 3.3, \( |R| = O(10^{203}) \). On each route there could contain in theory a maximum of \( O(10^2) \) trips. The set \( R_j \) is the set of trips that are on route \( j \in R \).

The following properties are created from the set of routes \( R \).

**Parameters**

\[
G_j = \text{Work load on created route } j
\]

5.1.1 Training Cost

If a leader is not qualified to take a particular trip type but is assigned that trip type on a route then there is a training cost incurred. Refer to Section 4.2.2 for a description of the training costs.
Parameters

\[ \hat{T}_{lj} = \text{Training cost for leader } l \text{ to do route } j \]

The parameter \( \hat{T}_{lj} \) is created using the following algorithm

```
Algorithm (5.1)
for \( l \in \mathcal{L} \) do
    for \( j \in \mathcal{R} \) do
        set \( \hat{T}_{lj} := 0 \)
        for \( k \in S_j \setminus Q_l \) do
            \( \hat{T}_{lj} := \hat{T}_{lj} + T_k \)
        end
    end
end
```

Refer to Appendix B for the mosel code implemented in Xpress-MP.

Objective Function

\[
\min \sum_{l \in \mathcal{L}} \sum_{j \in \mathcal{R}} \hat{T}_{lj} U_{lj} \quad (5.2)
\]

The objective function is linear since the parameter \( \hat{T}_{lj} \) is created outside the objective function.

Constraints

Every route created must be covered by one leader only

\[
\sum_{j \in \mathcal{R}} \sum_{l \in \mathcal{L}} U_{lj} \geq 1 \quad \forall \ t \in \mathcal{T} \quad (5.3)
\]

Every leader must cover one route or none at all

\[
- \sum_{j \in \mathcal{R}} U_{lj} \geq -1 \quad \forall \ l \in \mathcal{L} \quad (5.4)
\]
5.1.2 Preferred Work Loads

The leaders have a desired work load $d_l$. The model seeks to fit the leaders to their desired work load by using a soft constraint. This was done, since part of the scope of the problem was to create a schedule where leaders could have some input into their own work load level. It was predicted that it would be quite unlikely that every leader would be assigned to a route with exactly their preferred work load.

The penalty on the objective function for assigning leaders to routes not equal to their desired work load is subject to a certain tolerance level. This is the amount the leaders would accept as a fair deviation from their desired work load. In consultation with the client the tolerance was set to be $\pm 10\%$. This means that a leaders preference is actually a domain of preferences $10\%$ each side of $d_l$. The size of the penalty incurred to the objective function if a leader is not assigned to a route with work load within $d_l \pm 10\%$ is not a physical cost. Testing was done to determine the optimal level in Section 5.3. The soft constraint can be visualized in Figure 5.1.

Parameters

\[
\begin{align*}
P_{lj} &= \text{The amount by which } G_j > d_l + 0.1 \\
\hat{P}_{lj} &= \text{The amount by which } G_j < d_l - 0.1 \\
\text{penalty} &= \text{Penalty level applied to the distances } P_{lj}, \hat{P}_{lj}
\end{align*}
\]

Objective Function

The penalty for the work load can be added to the objective function in Equation (5.2).

\[
\min \sum_{l \in L} \sum_{j \in R} \hat{T}_{lj} U_{lj} + \text{penalty} \sum_{l \in L} \sum_{j \in R} ((G_j - (d_l + 0.1))^+ + ((d_l - 0.1) - G_j)^+) U_{lj}
\]

Where $(G_j - (d_l + 0.1))^+$ takes the maximum of $(0, (G_j - (d_l + 0.1)))$

A new parameter is introduced to capture the penalty for a leader to do a route.

Parameters

\[
\begin{align*}
B_{lj} &= \text{Work load penalty incurred when leader } l \text{ is assigned to route } j
\end{align*}
\]

The parameter $B_{lj}$ is calculated in the same way as the penalty distance is defined in the objective.
Figure 5.1: Visualization of Work Load Penalty
function, Equation (5.5). The algorithm determines the cost incurred to the objective function for a leader \( l \) to do a route \( j \). The algorithm is shown below. Refer to Appendix B for the mosel code used in Xpress-MP.

```
Algorithm (5.2)
begin
for \( l \in L \) do
    for \( j \in R \) do
        if \( d_l + 0.1 \geq G_j \) and \( d_l - 0.1 \leq G_j \) then
            \( B_{lj} := 0 \)
        else
            if \( d_l + 0.1 \leq G_j \) then
                \( B_{lj} := \text{penalty} (G_j - d_l - 0.1) \)
            else
                \( B_{lj} := \text{penalty} (d_l - 0.01 - G_j) \)
            end if
        end if
    end
end

Objective Function

We have now defined the route based model to be

\[
\min \sum_{l \in L} \sum_{j \in R} \tilde{T}_{lj} U_{lj} + \sum_{l \in L} \sum_{j \in R} B_{lj} U_{lj}
\]

subject to the constraint Equations (5.3) and (5.4). The problem now is to initialize the master problem with a set of feasible routes. The first attempt to do this is the use of heuristic methods.

5.2 Initialization Heuristic

Heuristic methods were used to generate a set of feasible routes such that the route based model was initialized. The key to the heuristic models is that they consider every leader to be homogeneous in terms of qualification and desired work load. If the leaders are considered homogeneous, then the problem is to simply create shortest routes (minimum transfer cost) in the network such that every
trip is visited. The heuristic method cannot capture the training cost since that cost is specific to each leader. Constraints on the properties of the routes, for example, work load can be modeled. Creating routes in the network was done using a different set of variables to that used in Section 4. The network which encapsulates the problem is also redefined.

5.2.1 Network Model

The network is created under the same conditions as in Chapter 4 with the added component that an arc exists from every trip in \( T \) to an \( ENDNODE \) (a trip that does not physically exist but must be entered and not left, an absorbing node). This produces a new set of possible trip pairs \( A' \). The maximum theoretical cardinality of the new set is \(|A'| = O(10^5)\), however after the network is created with \( MAXGAP = 14 \), the exact \(|A'| = 31,369\) which is quite feasible to be solved in Xpress-MP. The set \( A' \) is defined by

\[
A' = A \cup \{(i, ENDNODE) \in T \times ENDNODE\}
\]  

(5.7)

The only actual costs considered are the transfer costs. An example network for this model is shown in Figure 5.2. The arcs represent the feasible paths that leaders can take such that \( T \) is covered.

The optimal routes (minimum transfer cost) for this subset of the trips \( T \) is shown in Table 5.1

<table>
<thead>
<tr>
<th>Trip 1</th>
<th>Start Date</th>
<th>Trip 1</th>
<th>End Date</th>
<th>Trip 1</th>
<th>Trip 2</th>
<th>...</th>
<th>Work Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSW060211</td>
<td>02/11/06</td>
<td>02/24/06</td>
<td>ASK060302</td>
<td>ENDNODE</td>
<td>79%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WR060605</td>
<td>06/05/06</td>
<td>06/15/06</td>
<td>WR060619</td>
<td>ENDNODE</td>
<td>74%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WSU060525</td>
<td>05/25/06</td>
<td>06/03/06</td>
<td>ZSS060306</td>
<td>ENDNODE</td>
<td>100%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Optimal routes produced from Figure 5.2

5.2.2 Transfer Cost

Sets

- \( E \) is the set of all Trips \( T \cup \{ENDNODE\} \)

Variables

\[
X_{ij} = \begin{cases} 
1, & \text{if trip } i \text{ is followed by trip } j \\
0, & \text{Otherwise} 
\end{cases}
\]
Figure 5.2: Example network used in the decomposition model
Objective Function

\[ \min \sum_{(i,j) \in A} c_{ij}X_{ij} \quad (5.8) \]

The problem is modeled using different logic to that of Chapter 4. In this model the routes are created such that there is a unique start trip for every route, but every route must now finish at an arbitrary ENDNODE.

When optimizing the routes the only physical cost incurred for model 5.2 is the transfer cost. The optimal solution would be to have every leader do one trip, since this would create no transfer cost. However the number of leaders available has an upper limit set by the client equal to \(|L|\). The reason that this maximum is set is because the client predicts there would be a loss of quality of service if the number increased. This would in turn impact on future revenue of the company.\(^1\) It was necessary to model the network using an ENDNODE so that the number of routes created can be limited. Since \(\sum_{i \in T} X_i \text{ENDNODE} \) will give the number of routes created, (equivalent to the number of leaders used.)

Constraints

The number of routes created cannot exceed the number of leaders available. The number of routes created is determined by summing all arcs into the ENDNODE.

\[ \sum_{i \in T} X_{i \text{ENDNODE}} \leq |L| \quad (5.9) \]

Every trip in \(T\) must be left once only. Any trip in \(T\) can have a trip pair with any trip in \(E\)

\[ \sum_{j \in E} X_{ij} = 1 \quad \forall \ i \in T \quad (5.10) \]

Every trip in \(T\) must be entered once from a trip in \(T\), except the first trip of a route which will not be entered at all.

\[ \sum_{i \in T} X_{ij} \leq 1 \quad \forall \ j \in T \quad (5.11) \]

These three constraints are all that is needed to completely model the problem of creating routes in the network, at minimum cost such that every trip is covered and every route terminates in the

\(^1\)Incorporating this more precisely into a model is a possible future project
There is no need for a flow balance constraint in this network. This is because the network is acyclic and every node in $T$ must be left, thus sub tours are never possible.

### 5.2.3 StartCost Model

Since $|T| < |L|$ (in Figure 5.2) then obviously the optimal way to cover this problem is to assign a single leader to each trip, so that no transfer cost is incurred. For operational reasons, it was suggested by the client that as well as minimizing cost, the number of leaders used in a season should be minimized. To model this in the heuristic model a cost was used to penalize the objective function when a leader was used, or equivalently a route created.

**Parameters**

$$StartCost = \text{Penalty incurred if a route is created} \quad (5.12)$$

**Objective Function**

$$\min \sum_{(i,j) \in A} c_{ij}X_{ij} + \sum_{i \in T} (X_{i, \text{ENDNODE}}) \text{StartCost} \quad (5.13)$$

Since every route created must end on ENDNODE, to constrain the number of leaders used the penalty $StartCost$ is applied to any arc in the network that is done between a trip and ENDNODE. The value of the $StartCost$ was tested on to see the effects that it had on the number of leaders used. This was done in Section 5.4.

### 5.2.4 Work Load, Approximation

The routes are to be created such that the work load is no greater than 90%. As a first approximation to constraining the work load a new definition of work load was introduced

$$\frac{\text{Days worked on route}}{\text{Season length}}.$$ 

A work load limit for all routes was set to be 90%. The season length could be found by taking $\max_{i \in T} (e_i) - \min_{i \in T} (s_i)$. The new definition was used as it was suspected that the new constraints introduced would increase the computation time significantly.

The variable $W_{ij}$ accumulates the days worked on a route. Then the total days worked on any route $i$ is given by evaluating $W_{i, \text{ENDNODE}}$. Since every trip is done once, Constraint (5.10) ensures there can never be more than one route with the same trip pair ($i \rightarrow \text{ENDNODE}$).
Variables

- $W_{ij} =$ Days worked up to trip $i$ when next trip is trip $j$

The variable $W_{ij}$ is defined for all feasible trip pairs in the set $A'$.

Constraints

\[
\sum_{k \in \mathcal{E}} W_{jk} = \sum_{i \in \mathcal{T}} W_{ij} + (e_j - s_j) \quad \forall \ j \in \mathcal{T} \tag{5.14}
\]

\[
W_{ij} \leq BigM' X_{ij} \quad \forall \ (i, j) \in A' \tag{5.15}
\]

Constraint (5.16) is used to limit the work load on every route created to be less than 90% using the approximate definition of work load.

\[
W_{i \text{ ENNDNODE}} \leq \left( \max_{j \in \mathcal{T}} (e_j) - \min_{j \in \mathcal{T}} (s_j) \right) \times 0.9 \quad \forall \ i \in \mathcal{T} \tag{5.16}
\]

The Constraint (5.14) is defined so that the days worked are accumulated for the node $j$, Then when the variable $W_{ij}$ is evaluated at $W_{i \text{ ENNDNODE}}$ with Equation (5.15), the days worked will be in reference to node $i$. Constraint (5.15) allows the value of $W_{ij}$ to be less than some suitably large number $BigM'$ when $X_{ij} = 1$. The value of $BigM'$ is the upper bound for the maximum possible number of days worked up until and including node (trip) $i$. As a first approximation to the worst possible case $BigM'$ is set to be as large as the number of days in the season $BigM' = \max_{i \in \mathcal{T}} (e_i) - \min_{i \in \mathcal{T}} (s_i)$.

Constraints (5.14),(5.15) work by the fact that the constraints modeling how the routes are created (variable $X_{ij}$) ensure that every node is visited exactly once except for the $\text{ENDNODE}$. An example follows to show the logic.

Consider the set of nodes shown in Figure 5.3 with the bold arcs representing the optimal solution, and apply the Constraints (5.14) and (5.15).

Constraint (5.14):

\[
\begin{align*}
W_{11} + W_{12} + W_{13} + W_{14} + W_{15} + W_{16} + W_{1 \text{ ENNDNODE}} &= W_{11} + W_{21} + W_{31} + W_{41} + W_{51} + W_{61} + (e_1 - s_1) \\
W_{21} + W_{22} + W_{23} + W_{24} + W_{25} + W_{26} + W_{2 \text{ ENNDNODE}} &= W_{12} + W_{22} + W_{32} + W_{42} + W_{52} + W_{62} + (e_2 - s_2) \\
W_{31} + W_{32} + W_{33} + W_{34} + W_{35} + W_{36} + W_{3 \text{ ENNDNODE}} &= W_{13} + W_{23} + W_{33} + W_{43} + W_{53} + W_{63} + (e_3 - s_3) \\
\vdots
\end{align*}
\]
Figure 5.3: Example network to demonstrate Constraints (5.14), (5.15)
By using the acyclic nature of the network, and the constraints on variable $X_{ij}$, every trip must be done only once, and there are no cycles present. The optimal solution in Figure 5.3 can be applied with Constraint (5.15)

Constraint (5.15):

\[
W_{11} = W_{13} = W_{14} = W_{15} = W_{16} = W_1_{ENDNODE} \leq 0
\]

\[
W_{12} \leq BigM'
\]

\[
W_{21} = W_{22} = W_{23} = W_{24} = W_{25} = W_{26} \leq 0
\]

\[
W_2_{ENDNODE} \leq BigM'
\]

\[
W_{31} = W_{32} = W_{33} = W_{35} = W_{36} = W_3_{ENDNODE} \leq 0
\]

\[
W_{34} \leq BigM'
\]

\[
\vdots
\]

The constraints then accumulate the days worked along a particular route.

\[
W_{2_{ENDNODE}} = W_{12} + (e_2 - s_2)
\]

\[
W_{12} = (e_1 - s_1)
\]

\[
W_{5_{ENDNODE}} = W_{45} + W_{34} + (e_5 - s_5)
\]

\[
W_{45} = W_{34} + (e_4 - s_4)
\]

\[
W_{34} = (e_3 - s_3)
\]

\[
W_{6_{ENDNODE}} = (e_6 - s_6)
\]

It was proposed that because the network had the advantages of being directed, acyclic and conservative (refer to Chapter 1) that the largest possible value of $BigM'$ could be tightened, to correspond to an individual node, $BigM'_i \forall i \in T$. The vector $BigM'_i$ is considered as a parameter of the problem since there is no decision variables involved.

**$BigM'_i$ Algorithm**

The tightening of the $BigM'$ constraint should improve the run time of the program as in Section 4.2.4, since the solution space is decreasing for the variable $W_{ij}$. Constraint (5.15) could then be improved to Constraint (5.17). The value of $BigM'$ is only in reference to node $i$ since the definition
of $W_{ij}$ accumulates the days worked up to node $i$ in Constraint (5.15).

$$W_{ij} \leq \text{Big}M'_{ij}X_{ij} \quad \forall (i,j) \in A'$$ (5.17)

The value of $\text{Big}M'_i$ for any $i$ can be calculated by the recursive formula

$$\text{Big}M'_i = \max_{p \in T \setminus \{i\}, (p,i) \in A} \left( \text{Big}M'_p \right) + e_i - s_i$$ (5.18)

The algorithm to compute $\text{Big}M'_i$ works by considering the nodes $i$ sequentially, starting from the trip corresponding to the min$_{i \in T} (s_i)$. Thus the set $T$ must be sorted in increasing size of $s_i$. A choice of the Shell sort algorithm which has complexity $O(n^2)$ [10] was used, since it was easy to implement in Xpress-MP. Refer to Appendix C for an example of the sorting algorithm used. Since the nature of the network is acyclic the algorithm to find the longest possible path to a node $i$ is defined as follows. Refer to Appendix D for the mosel code implemented in Xpress-MP.

Algorithm (5.3)
\begin{verbatim}
begin
Order the set $T$ according to trip start times from smallest to largest
for $i \in T$ do
  set $\text{Big}M'_i = e_i - s_i$
  for $(p, i) \in A$ do
    if $\text{Big}M'_i < \text{Big}M'_p + e_i - s_i$ then
      set $\text{Big}M'_i = \text{Big}M'_p + e_i - s_i$
    end if
  end for
end for
end
\end{verbatim}

The algorithm relies on the Principle of Optimality from [22]. The algorithm works by considering the ordered set $T$. At node $i$, $\text{Big}M'_i = e_i - s_i$. Then every element $(p, i)$ in the set $A$, (obviously none for the first trip in the season,) is checked. When $\text{Big}M'_i < \text{Big}M'_p + e_i - s_i$ the value of $\text{Big}M'_i$ is updated to be $\text{Big}M'_i = \text{Big}M'_p + e_i - s_i$. The set $T$ needs to be ordered in terms of $s_i$ so that when the comparison is done on adjacent nodes, the maximum of $\text{Big}M'_p + e_i - s_i$ will definitely give the longest path, since the network is acyclic. The value at node $i$ completely captures all possible maximum length routes back through the network to the start of the season.

**Theorem 5** The algorithm for $\text{Big}M'_i$ has complexity $O(nm)$.
**Proof:** The algorithm at the first for loop has 1 operation for each of the elements in $T$, $|T| = n$. Then within this for loop another for loop goes through each element in $A$, $|A| = m$. Within this for loop there is 1 comparison, 4 additions, and 1 operation. Thus the complexity is $n(6m) = O(nm)$.

The tightening of $BigM'$ bound had no effect on the optimal objective value found, however the run time was predicted to improved. The computational results were carried out in Section 5.4.

The definition of the work load in model 5.2.4 using a season length, is not the actual work load experienced by the leaders. A leader could start their route into the season, thus it is not an accurate representation of the real work load. There is no variable in the heuristic model which captures a created routes first and last trip as in the complete model. So to calculate the actual number of days in a route a new variable was created which counted the days off on a route.

5.2.5 Work Load

To model the correct definition of work load, the days off on a route are accumulated. With the number of days off on a route known the correct definition of work load can be imposed to constrain the work load. The days off were accumulated with a new variable $D_{ij}$, created under the same conditions as variable $W_{ij}$ with essentially the same constraints.

**Variables**

$$D_{ij} = \text{Accumulated days off for trip } i \text{ when next trip is any trip } j$$

**Constraints**

The variable is created such that it is valid for every trip pair in $A'$. Equation (5.19) is used to model the work load constraint on every route created.

$$W_{i \text{ ENDNODE}} \leq (W_{i \text{ ENDNODE}} + D_{i \text{ ENDNODE}}) \times 0.9 \quad \forall \ i \in T \quad (5.19)$$

Constraints needed to capture the logic of $D_{ij}$ are

$$\sum_{k \in \mathcal{E}} D_{jk} = \sum_{i \in \mathcal{T}} D_{ij} + (s_j - e_i) X_{ij} \quad \forall \ j \in \mathcal{T} \quad (5.20)$$

$$D_{ij} \leq BigM'' X_{ij} \quad \forall \ (i,j) \in \mathcal{A}' \quad (5.21)$$

52
The Constraint (5.20) is defined so that the days off are accumulated for the node $j$. However when applying Constraint (5.21) the variable $D_{ij}$ when evaluated at $D_{i \text{ ENDNODE}}$ the days off will be in reference to node $i$. Constraint (5.20) works by using the value of $X_{ij}$ to determine the amount of days off between the preceding trip pair $(i \rightarrow j)$ to the current trip $j$.

The Constraint (5.21) is the same as Constraint (5.15). The worst case bound is again $BigM'' = \max_{i \in T} (e_i) - \min_{i \in T} (s_i)$. However the same principle applies of tightening the $BigM''$ bound to $BigM''_i$ predicted to improve the run time of the program. The Constraint (5.21) is modified to Constraint (5.22). The value of $BigM''$ is again in reference to node $i$ since the definition of $D_{ij}$ acclimates the days off up to node $i$ in Constraint (5.21).

\[
D_{ij} \leq BigM''_i X_{ij} \quad \forall (i, j) \in A' \tag{5.22}
\]

$BigM''_i$ Algorithm

The value of $BigM''_i$ for any $i$ can be calculated by the recursive formula

\[
BigM''_i = \max_{p \in T} \left( \max_{(p, i) \in A} \left( BigM''_p + s_i - e_p \right) \right) \tag{5.23}
\]

The algorithm is used to find the maximum number of days off that could possibly be accumulated up to a particular trip $i$. Refer to Appendix E for the Xpress-MP code to implement the algorithm for $BigM''_i$.

```
Algorithm (5.4)
begin
Order the set $T$ according to trip start times from smallest to largest
for $i \in T$ do
  set $BigM''_i = 0$
  for $(p, i) \in A$ do
    if $BigM''_i < BigM''_p + s_i - e_p$ then
      set $BigM''_i = BigM''_p + s_i - e_p$
    end if
  end
end
end
```

The algorithm works on the same principles as in Section 5.2.4. However the accumulation of days
is in reference to the days off between successive trips \((s_i - e_p)\). Using Theorem 5 this algorithm has complexity of \(O(nm)\). The computational results between \(BigM''\) and \(BigM''\) are in Section 5.4.

The main purpose of the heuristic is to generate feasible routes. As is discussed in Chapter 6, the heuristic problem is turned into a subproblem using column generation. For this reason it is preferable that the routes are created in a short amount of computational time. Auxiliary variables as well as satisfying the soft constraint of consecutive days worked could also be used as an alternative to the work load constraints modeled with variables \(W_{ij}\) and \(D_{ij}\).

### 5.2.6 Auxiliary Variables to Constrain Consecutive Days Worked

The soft constraint that specifies the number of consecutive days worked should be less than \(DaysLimit\) as was modeled in Chapter 4. As in Chapter 4 when the auxiliary variable constraint used it is hoped that the run time will be suitably small and that the constraints needed to limit the route work loads to be less than 90% are redundant.

The variable \(E_i\) is created for each trip \(i\) as apposed to Section 4.2.4 where the variable is created for each leader \(l\) and trip \(i\). The tightest bound on the value of \(BigM\) is the same as used in Section 4.2.4, \(BigM_{ij}\).

**Variables**

\[
E_i = \text{Number of days accumulated without a suitable length break at trip } i \tag{5.24}
\]

**Constraints**

\[
E_j \geq E_i + (e_j - e_i) - BigM_{ij} (1 - X_{ij}) \quad \forall (i, j) \in A
\]

\[
s.t. \, (s_j - e_i) \leq \text{Break} \tag{5.25}
\]

\[
E_i \leq DaysLimit \quad \forall i \in T \tag{5.26}
\]

\[
E_i \geq e_i - s_i \quad \forall i \in T \tag{5.27}
\]

The variable \(E_i\) uses the same logic as used in Section 4.2.4. If the binary variable \(X_{ij} = 1\) and creating that trip pair makes \(E_i \geq DaysLimit\) then there is a contradiction in the constraints. The computational and work load results are tested in Section 5.4
5.2.7 Interaction Heuristics

The heuristics so far to create routes have been done independently of any information on the leaders desired work loads. An optimal way to do this is the use of column generation Chapter 6. One way to try and create routes that might be favorable in the route based model is to create routes with some input information about the leaders desired work loads \( (d_l) \). A soft constraint was imposed between the work load on each route created in the heuristic and a value that incorporated the input average and variance of the parameter \( d_l \).

The idea is that if the variance of the leader’s desired work loads is large then the spread of the created routes work loads in the heuristic models should also be large. Conversely, if the variance is small then the created work loads should have small spread. The average and variance of the parameter \( d_l \) can be easily calculated in Xpress-MP. Note that this method can only be applied with constraint Equations (5.14), (5.15), (5.20), (5.21) since a knowledge of the work load created on routes is needed. The input average and input variance for the parameter \( d_l \) is given in Equations (5.28) and (5.29) respectively.

\[
\text{Input Average} = \frac{\sum_{l \in \mathcal{L}} d_l}{|\mathcal{L}|} \tag{5.28}
\]
\[
\text{Input Variance} = \frac{\sum_{l \in \mathcal{L}} \left( d_l - \frac{\sum_{l \in \mathcal{L}} d_l}{|\mathcal{L}|} \right)^2}{|\mathcal{L}| - 1} \tag{5.29}
\]

Variables

Variables are needed to determine the amount by which a created route is outside the soft constraint domain.

\[
L_i = \text{The amount } G_i > (\text{Input Average} + \text{Input Variance} + 0.1) \tag{5.30}
\]
\[
\hat{L}_i = \text{The amount } G_i < (\text{Input Average} - \text{Input Variance} - 0.1) \tag{5.31}
\]

Constraints

The constraint can be visualized in Figure 5.4: The soft constraint will give no penalty or extra cost to the objective function if the created work load is within the domain of

\[
[\text{Input Average} - \text{Input Variance} - 0.1, \text{Input Average} + \text{Input Variance} + 0.1]
\]

55
Figure 5.4: Visualization of the soft constraint on route work loads
At the ends of this domain the constraints needed are

\[
\frac{W_{i\text{ ENDNODE}}}{(W_{i\text{ ENDNODE}} + D_{i\text{ ENDNODE}})} \leq (\text{Input Average} + \text{Input Variance} + 0.1)
\]

\[
\frac{W_{i\text{ ENDNODE}}}{(W_{i\text{ ENDNODE}} + D_{i\text{ ENDNODE}})} \geq (\text{Input Average} - \text{Input Variance} - 0.1)
\]

so then

\[
W_{i\text{ ENDNODE}} - (W_{i\text{ ENDNODE}} + D_{i\text{ ENDNODE}}) \times
\]

\[
(\text{Input Average} + \text{Input Variance} + 0.1) \leq 0
\]

\[
W_{i\text{ ENDNODE}} - (W_{i\text{ ENDNODE}} + D_{i\text{ ENDNODE}}) \times
\]

\[
(\text{Input Average} - \text{Input Variance} - 0.1) \geq 0
\]

Thus the constraints used to capture this logic are

\[
L_i \geq W_{i\text{ ENDNODE}} - (W_{i\text{ ENDNODE}} + D_{i\text{ ENDNODE}}) \times
\]

\[
(\text{Input Average} + \text{Input Variance} + 0.1) \forall i \in T \quad (5.32)
\]

\[
L_i \geq 0 \forall i \in T \quad (5.33)
\]

\[
\hat{L}_i \leq W_{i\text{ ENDNODE}} - (W_{i\text{ ENDNODE}} + D_{i\text{ ENDNODE}}) \times
\]

\[
(\text{Input Average} - \text{Input Variance} - 0.1) \forall i \in T \quad (5.34)
\]

\[
\hat{L}_i \leq 0 \forall i \in T \quad (5.35)
\]

**Objective Function**

The objective function from Section 5.2.3 with the added penalty term is Equation (5.36). The penalty factor multiplied by the distance the work load of the created route is outside the specified domain is pen.

\[
\min \quad \sum_{(i,j) \in A} c_{ij}X_{ij} + \sum_{i \in T} X_{i\text{ENDNODE}}\text{StartCost} + \sum_{i \in T} \left(L_i - \hat{L}_i\right)\text{pen} \quad (5.36)
\]

The value of pen is an artificial cost and the value was tested to determine the optimal level of penalty incurred. This is done in Section 5.4.

A network heuristic method was designed that does not need to accumulate the work load for a route, so as constraint Equations (5.14), (5.15), (5.20), (5.21) were not needed. The proposal was

57
to implement some simple heuristic ideas when creating the network. The first approach was to further constrain the feasible set of trips \((i, j)\) such that there must exists at least \(MINGAP\) days between the end of trip \(i\) and start of trip \(j\), \((s_j - e_i \geq MINGAP)\). This would then ensure that the work load for a particular leader would always be \(< 100\%\) (if a leader was assigned to more than one trip.) The computational results are shown in Section 5.4.

The previous constraint produced an optimal objective value which was relatively large. A softer constraint was imposed when creating the network which only creates trip pairs \((i, j)\) when the trips, if considered consecutively give a work load of less than some \(Average\) bound. Expressed mathematically as:

\[
\frac{e_i - s_i + e_j - s_j}{e_j - s_i} \leq Average
\]  

(5.37)

The computational results are shown in Section 5.4.

### 5.3 Computational Analysis of Route Based Model

Table 5.2 defines each of the models tested.

<table>
<thead>
<tr>
<th>Model</th>
<th>Objective Function Equation</th>
<th>Constraint Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.1</td>
<td>(5.2)</td>
<td>(5.3),(5.4)</td>
</tr>
<tr>
<td>5.1.2</td>
<td>(5.6)</td>
<td>(5.3),(5.4)</td>
</tr>
<tr>
<td>5.2.2</td>
<td>(5.8)</td>
<td>(5.9)-(5.11)</td>
</tr>
<tr>
<td>5.2.3</td>
<td>(5.13)</td>
<td>(5.9)-(5.11)</td>
</tr>
<tr>
<td>5.2.4</td>
<td>(5.13)</td>
<td>(5.9)-(5.11),(5.14)-(5.16)</td>
</tr>
<tr>
<td>5.2.5</td>
<td>(5.13)</td>
<td>(5.9)-(5.11),(5.14),(5.15),(5.19)-(5.21)</td>
</tr>
<tr>
<td>5.2.6</td>
<td>(5.13)</td>
<td>(5.9)-(5.11),(5.14),(5.15),(5.19),(5.20),(5.25)-(5.27)</td>
</tr>
<tr>
<td>5.2.7</td>
<td>(5.36)</td>
<td>(5.9)-(5.11),(5.14),(5.15),(5.19),(5.20),(5.32)-(5.35)</td>
</tr>
</tbody>
</table>

Table 5.2: Definition of the models tested

#### 5.3.1 Computational Analysis of \(penalty\)

The only other cost incurred in the objective function for model 5.1.2 is that of the training cost. As a starting point the training cost was considered to be homogeneous and set to \(T_k = 1\). Then the ratio of \(\frac{Training \ Cost}{penalty}\) is the relative weighting of the importance of the two costs. If the ratio is \(< 1\) the routes will be assigned to leaders with a bias on the work load preference. It was decided
that since the work load preference is not actually a cost incurred to company directly then the ratio should be of order 2. The penalty parameter was tested on in Table 5.3.

The penalty value was tested on in Table 5.3 first using the homogeneous data then tested on the actual data set to determine the optimal penalty level.

<table>
<thead>
<tr>
<th>penalty</th>
<th>Training Cost</th>
<th>Objective Value of Model 5.1.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>1</td>
<td>285.248</td>
</tr>
<tr>
<td>0.8</td>
<td>1</td>
<td>280.511</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>262.661</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>280.413</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>276.518</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>305.822</td>
</tr>
<tr>
<td>0.2</td>
<td>Actual Data Set</td>
<td>15478.1</td>
</tr>
<tr>
<td>1</td>
<td>Actual Data Set</td>
<td>14830.7</td>
</tr>
<tr>
<td>20</td>
<td>Actual Data Set</td>
<td>15904.6</td>
</tr>
<tr>
<td>50</td>
<td>Actual Data Set</td>
<td>15559</td>
</tr>
<tr>
<td>100</td>
<td>Actual Data Set</td>
<td>15613.3</td>
</tr>
<tr>
<td>150</td>
<td>Actual Data Set</td>
<td>15270.9</td>
</tr>
</tbody>
</table>

Table 5.3: Summary of varying penalty

Table 5.3 shows that the penalty value is highly dependent on the routes created in the heuristic route generating method. For this reason and by a visual inspection of the routes created, the optimal value of penalty was chosen to be 1. Model 5.1.2 had a computation time of 4.0 sec.

5.4 Computational Analysis of Heuristic Methods

5.4.1 Computational Analysis of StartCost

The first experiment was to set all the data to be homogeneous thus determining the lower bound on |L| so that every trip can be covered in the season. To do this every city pair cost was set to 10, except the same cities which were set to be 0. A very large start cost was applied so that the effects of the transfer costs were negated. The maximum number of trip pairs done in a season is 675 − 1 = 674 thus the maximum the transfer cost can be is 6740. So any start cost greater than 6740 will certainly negate the transfer costs. Then the StartCost was cross tested against the actual data set. A summary of StartCost results is shown in Table 5.4 using model 5.2.3.

As the results show the lower bound on the number of leaders needed to run all trips is 44. When
<table>
<thead>
<tr>
<th>StartCost</th>
<th>Transfer Cost</th>
<th>Number of Routes Created</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Homogeneous $10</td>
<td>54</td>
</tr>
<tr>
<td>50</td>
<td>Homogeneous $10</td>
<td>54</td>
</tr>
<tr>
<td>100</td>
<td>Homogeneous $10</td>
<td>44</td>
</tr>
<tr>
<td>1000</td>
<td>Homogeneous $10</td>
<td>44</td>
</tr>
<tr>
<td>1000000</td>
<td>Homogeneous $10</td>
<td>44</td>
</tr>
<tr>
<td>1</td>
<td>Actual Data Set</td>
<td>54</td>
</tr>
<tr>
<td>50</td>
<td>Actual Data Set</td>
<td>54</td>
</tr>
<tr>
<td>100</td>
<td>Actual Data Set</td>
<td>54</td>
</tr>
<tr>
<td>200</td>
<td>Actual Data Set</td>
<td>49</td>
</tr>
<tr>
<td>250</td>
<td>Actual Data Set</td>
<td>47</td>
</tr>
<tr>
<td>350</td>
<td>Actual Data Set</td>
<td>46</td>
</tr>
<tr>
<td>450</td>
<td>Actual Data Set</td>
<td>45</td>
</tr>
<tr>
<td>490</td>
<td>Actual Data Set</td>
<td>45</td>
</tr>
<tr>
<td>500</td>
<td>Actual Data Set</td>
<td>44</td>
</tr>
</tbody>
</table>

Table 5.4: Summary of varying StartCost

considering homogeneous costs the best StartCost to use, is \( \approx 10 \) times larger than the transfer cost. The actual data set showed that the StartCost which gave the optimal number of leaders used (minimum) was 500. Excellent run times were exhibited when producing routes; however the routes had no constraint on the work loads created.

5.4.2 Computational Analysis of \( \text{BigM}' \)

The computational results for the tightening of \( \text{BigM}' \) are shown in Table 5.5. As in Chapter 4 the cardinality of the set \( T \) used \( x = |T| \) is the first \( x \) trips offered in the season. The \( \text{BigM}' \) constraint is tested with model 5.2.4. The addition of the algorithm to tighten Constraint (5.15)

| \(|T|\) | Run Time (sec) | \( \text{BigM}' \) | \( \text{BigM}'_i \) | Objective Value |
|--------|----------------|---------------------|---------------------|-----------------|
| 200    | 16.3           | 15.2                | 11742               |
| 400    | 630.6          | 581.9               | 24294               |
| Full Data | 2297.7          | 1628.7              | 30865               |

Table 5.5: Effect of improving \( \text{BigM}' \)

decreases the computation time for the full data set by \( \approx 29\% \). This is due to the refined solution space. The implications are that the algorithm for \( \text{BigM}'_i \) will be implemented when using this model.
5.4.3 Computational Analysis of $BigM''$

The computational results between $BigM''$ and $BigM_i''$ are shown in Table 5.6. The same run time was used for each cardinality of $T$ experimented with. The $BigM''$ constraint is tested with model 5.2.5. Table 5.6 shows that the computational time needed to use the definition of work load in Section 5.2.5 is not efficient. However the run time for the full data set had a solution with Gap= 0.06%. This was achieved in a time less than the required maximum computation time, stated in Chapter 1. This implied that depending on how sub-optimal the heuristic model is (refer to Chapter 6) then this model could be used. The effect of tightening the $BigM''$ constraint gave an $\approx 0.3\%$ decrease in computation time. The relatively small improvement might be because the computation time is no longer dominated by the solution space but rather the use of two $BigM$ constraints.

| $|T|$ | Run Time (sec) | $BigM''$ | $BigM_i''$ |
|-----|----------------|---------|-----------|
| 45  | 0.1            | 0.1     |           |
| 74  | 495.1          | 380.0   |           |
| 133 | 1700 (Gap 3.53%) | 1700 (Gap 8.9%) |           |
| Full Data Set | 63000 (Gap 0.71%) | 63000 (Gap 0.06%) |           |

Table 5.6: Effect of improving $BigM''$

5.4.4 Computational Analysis of $pen$

This was a general approach to creating good routes thus the penalty should not out weigh the minimization of the transfer costs. An approximate way of checking this was to make sure the number of routes created before the constraint was introduced is the same after. The program run time was tested to see the effect that the new constraints have. The computational results are shown in Table 5.7.

The optimal value of $pen$ used is in the order of 20. This gave the best results in terms of penalizing routes created outside of the tolerance range and the least impact on the number of routes created. The statistics do improve the routes created, however a significant increase in computational time was observed, and for the full data set no integer solution was found at extremely large run times.
| $|\mathcal{T}|$ | pen | Run Time (sec) | Number routes created |
|---|---|---|---|
| | | Model 5.2.5 | Model 5.2.7 | Model 5.2.5 | Model 5.2.7 |
| 72 | 0 | 0.3 | - | 27 | - |
| 72 | 20 | 0.3 | 1.2 | 27 | 27 |
| 72 | 50 | 0.3 | 1.2 | 27 | 28 |
| 72 | 100 | 0.3 | 1.6 | 27 | 39 |
| 72 | 150 | 0.3 | 1.5 | 27 | 40 |
| 253 | 0 | 1158.5 | - | 44 | - |
| 253 | 20 | 1493.7 | 18500 (Gap 6%) | 44 | 47 |
| 320 | 0 | 2853.6 (Gap 14%) | - | 44 | - |
| 320 | 20 | 20000 (Gap 12%) | 20000 (Gap 16%) | 44 | 48 |
| 358 | 0 | 434.1 | - | 44 | - |
| 358 | 20 | 434.1 | 3556.3 (Gap 2.7%) | 44 | 45 |
| Full Data Set | 0 | 62893.5 (Gap 0.6%) | - | 44 | - |
| Full Data Set | 50 | 500000 (Gap $\infty$) | 500000 (Gap $\infty$) | - | - |

Table 5.7: Summary of varying pen and between model 5.2.5 and model 5.2.7

5.4.5 Computational Analysis of MINGAP

The computational results of the program show that this provided a feasible solution when $MINGAP = 1$. Anything greater than 1 and the solution was infeasible because of the constraint on $|\mathcal{L}|$. When implemented the network heuristic performed very well and solved in 2.0 sec total time. However this approach gave an objective value 31% larger in the route based model, compared with $MINGAP = 0$. This was considered to be unacceptable and better improvements could be made. However a visual check of the schedule did show that all work loads were below 90%

5.4.6 Computational Analysis of Average

The computational results showed that the Average must be $\geq 0.945$ to obtain a feasible solution. This is because $|\mathcal{L}|$ was not large enough to cover the amount of routes need to be created to cover all trips. It was noted that the routes created from using Average = 0.945 were identical to the routes created when the $MINGAP = 1$. This network heuristic is also unsatisfactory since the total objective value for the route based model was 31% larger. This provided no better approximation than constraining the $MINGAP$ to be 1.

5.4.7 Comparison of Heuristic Models

In Sections 5.2.2 to 5.2.6 all factors that are able to be modeled in a heuristic problem are done. A suitable heuristic model could be chosen that best suits the scope of the problem. The client has
stipulated that the work load of any route should not exceed 90%. Also in practice the run time of the program should not exceed the estimated time that a skilled person could construct a schedule; this is estimated to be about 24 hours of work time. Comparison of the models is shown in Table 5.8.

<table>
<thead>
<tr>
<th></th>
<th>Run Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Model 5.2.3</td>
</tr>
<tr>
<td>98</td>
<td>0.1</td>
</tr>
<tr>
<td>253</td>
<td>0.4</td>
</tr>
<tr>
<td>320</td>
<td>0.5</td>
</tr>
<tr>
<td>Full Data</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Table 5.8: Computational performance of each route generating model

Using the computations in Table 5.8 it was assessed that the $\textit{BigM}$ constraints were increasing the run time of the program significantly. The run times needed to compute the optimal routes using the $\textit{BigM}$ constraints were outweighing any benefit gained in terms of restricting the work load to be less than 90%. This was assessed from a visual check of the routes created. Moreover since the decomposition of the problem leads itself to a column generation solution, it was decided that the best model is that presented in Section 5.2.3, since the run time was very small. If the heuristic method for finding routes proves to find a solution close to the global optimal that would have being found by the complete model (refer to Chapter 6), then heuristic model 5.2.6 would be satisfactory to determine a schedule for the company that considers all factors of the problem.

## 5.5 Comparison of Complete Model and Route Based Model

The complete model was mathematically complete, however it did not solve for the full data set due to the large amount of variables and constraints generated. The decomposition using heuristic methods to find feasible routes was built using all factors of the problem possible. To achieve this the computation time was significant, but still within the limit of 24 hours. To test whether the actual solution generated by the heuristic model was close to the optimal solution that would have been found with the complete model, small time horizon instances were tested in Table 5.9. The LP relaxation objective values are compared. The route based model is tested with heuristic model 5.2.3 since it does not contain work load constraints\(^2\). The route based model used is 5.1.1 since it does not penalize route work loads, since we are only comparing the routes in terms of actual costs

\(^2\)It was hard to guarantee that a problem was feasible in terms of work load requirements with small $|\mathcal{T}|$
incurred to the company. The complete model that was used in this comparison is model 4.2.2 with the modified objective function Equation (4.14) + \( \sum_{i \in T} \sum_{j \in T} StartCostZ_{ij} \) so as the costs incurred to the objective function are the same.

| \(|T|\) | Complete Model | Model 5.2.3 |
|-------|----------------|-------------|
|       | LP Relaxation  | LP Relaxation | # Routes | # Routes | % Difference |
| 20    | 2881           | 2881        | 14       | 14       | 0            |
| 30    | 4027.17        | 4097        | 17       | 19       | 1.7          |
| 40    | 5359.73        | 5551        | 24       | 25       | 3.6          |

Table 5.9: Computational comparison of the complete model 4.2.2 and the route based model 5.1.1.

The results show that for small cardinality of \( T \) the heuristic is within \( \approx 3\% \). It is noted that the difference in the objective value seems to be growing as \(|T|\) increases.

To further test just how sub-optimal the heuristic was with a larger cardinality of \( T \) or equivalently a larger time horizon, column generation was used. Since in theory column generation can find the optimal routes (as discussed in Chapter 3.)
Chapter 6

Column Generation Model

The theory of column generation was reviewed in Chapter 2. Column generation seeks to find optimal routes in a subproblem such that the master problem will price out favorably. Column generation works by initializing the master problem with some feasible set of routes. This was done with heuristic model 5.2.3. The costs considered in the first attempt at column generation were transfer costs, start costs, and training costs\(^1\).

The column generation procedure for the TSLP initially creates a variable between each created route from the initialization and each leader for a first pass at the master objective function. Column generation is an iterative process that once initialized loops back to find routes and leader combinations that will price out favorably in the master objective function. This process will enable a comparison of just how optimal the route based model 5.1.1 was, and the heuristic methods in creating routes with no regard to the leader’s properties. It will also determine how optimal the column generation procedure is by a comparison to the complete model solved with small \(|T|\).

Column generation for the TLSP can be essentially characterized by the following. At each iteration of the column generation procedure a subproblem is executed for each leader to find a route at minimum reduced cost. When the route is found for that leader if its reduced cost is less than some tolerance then the master objective function and master constraints are appended with the updated route properties, and the variable created between that route and that leader. The stopping criterion is if no variables are added to the master problem for a whole iteration. This is equivalent to no route and leader combinations existing that price out favorably in the master objective function (reduced cost < tolerance). Then a global integer search is executed on the

\(^1\)This is a relaxation of the problem, future work could involve including the work load penalty function.
master objective function using the updated created variables.

6.1 Master Problem

The master problem is as defined in Section 5.1 with the added relaxation of the work load penalty. The set $A_j$ is the set of all trip pairs done on a route $j \in \mathcal{R}$.

Sets

- $A_j$ is the set of all trip pairs done on route $j$

Objective Function

The objective function for the master problem uses the previously defined costs. For a leader $l$ to be assigned a route $j$ there is a training cost, transfer cost and a start cost.

$$\min \sum_{l \in \mathcal{L}} \sum_{j \in \mathcal{R}} \left( \tilde{T}_{lj} + \sum_{(t_1, t_2) \in A_j} c_{t_1 t_2} + \text{StartCost} \right) U_{lj} \quad (6.1)$$

The constraint on the master objective function Equation (6.2) implies that any trip $t \in \mathcal{T}$ can be covered by more than one leader, however the minimization statement will drive this down such that any trip will only be covered by one leader exactly. Equation (6.3) stipulates that each leader must be only assigned at most 1 route. The constraints in this format are in the standard form to be applied to column generation and in particular a form in which the reduced cost can be easily determined. Column generation as explained in Chapter 2 is closely related to the dual problem. For this reason Constraints (6.2) and (6.3) are given the dual variables $\Pi^1_l$ and $\Pi^2_l$ respectively.

Constraints

$$\sum_{j \in \mathcal{R}} \sum_{l \in \mathcal{L} \text{ s.t. } i \in R_j} U_{lj} \geq 1 \quad \forall t \in \mathcal{T} \quad \left( \Pi^1_l \right) \quad (6.2)$$

$$- \sum_{j \in \mathcal{R}} U_{lj} \geq -1 \quad \forall l \in \mathcal{L} \quad \left( \Pi^2_l \right) \quad (6.3)$$

Reduced Cost
The reduced cost of any variable in the master problem is defined in Chapter 2. The cost coefficient of variable \( U_{lj} \) is \( \hat{T}_{lj} + \sum_{(t_1, t_2) \in A_j} c_{t_1 t_2} + \text{StartCost} \). The dual variable \( \Pi^1_t \) is summed over all trips on a particular route \( j \). The reduced cost can be evaluated for each \( j \in \mathcal{R} \ l \in \mathcal{L} \).

\[
\left( \hat{T}_{lj} + \sum_{(t_1, t_2) \in A_j} c_{t_1 t_2} + \text{StartCost} \right) - \sum_{t \in R_j} \Pi^1_t + \Pi^2_l \forall j \in \mathcal{R} \ l \in \mathcal{L} \quad (6.4)
\]

### 6.2 Subproblem

The subproblem is used to find a route for a fixed leader \( l \) that is done at minimum cost with respect to the trips already covered. The objective function for the subproblem for a given leader \( l \) is a function of the training cost, transfer cost and the dual variable \( \Pi^1_t \), in fact it is Equation (6.4) – \( \Pi^2_l \). However we need to find a way to model this so as to arrive at a tractable optimization problem for finding minimum cost routes. The start cost is not considered in the subproblem since the cost is only associated with limiting the number of leaders used, not the actual costs of creating a route. The dual variable for a trip \( t \) is subtracted from the objective function if it is done on the created route. Since the objective function is a minimization problem a favorable route will be a trade off between having lots of trips done on a route and the costs associated with actually doing those trips.

#### Variables

\[
X_{ij} = \begin{cases} 
1, & \text{if trip } i \text{ is followed by trip } j \\
0, & \text{Otherwise}
\end{cases} \quad (6.5)
\]

\[
P_k = \begin{cases} 
1, & \text{if a created route possesses trip type } k \\
0, & \text{Otherwise}
\end{cases} \quad (6.6)
\]

#### Parameters

\[
tolerance = -0.001
\]

#### Objective Function

\[
\min \sum_{k \in \mathcal{H} \setminus Q_l} T_k P_k + \sum_{i \in T} \sum_{j \in T} X_{ij} c_{ij} - \sum_{t \in T} \Pi^1_t \left( \sum_{k \in \mathcal{E}} X_{tk} \right) \quad (6.7)
\]
Constraints

The constraints on the problem capture the logic that has been previously used in Chapter 5. The variable $X_{ij}$ is used to find a set of feasible trip pairs for a fixed leader $l$ such that the objective function is minimum. The variable $P_k$ is used to capture the trip types done on the route created by the variable $X_{ij}$.

\[ P_{\theta_i} \geq \sum_{j \in \mathcal{E}} X_{ij} \quad \forall \ i \in T \]  \hspace{1cm} (6.8)
\[ \sum_{i \in T} X_{i \text{ENDNODE}} = 1 \]  \hspace{1cm} (6.9)
\[ \sum_{j \in \mathcal{E}} X_{ij} \leq 1 \quad \forall \ i \in T \]  \hspace{1cm} (6.10)
\[ \sum_{i \in T} X_{ij} \leq \sum_{k \in \mathcal{E}} X_{jk} \quad \forall \ j \in T \]  \hspace{1cm} (6.11)

Once the subproblem for a particular leader was complete the reduced cost for that leader to be assigned that route was calculated. If the reduced cost was $< \text{tolerance}$ then the route properties were appended to the master problem. Refer to Appendix H for the mosel code implemented into Xpress-MP.

6.3 Computational Analysis of Column Generation

6.3.1 Comparison of Column Generation and Route Based Model

The computational results will provide information on just how optimal the route based model was using different heuristic route generating methods. This is achieved since column generation loops back and checks that no feasible routes exist that will price the master objective function out favorably. The heuristic methods to create routes do not consider the training cost, thus it was suspected that the route based model would be far from optimal. Table 6.1 presents the computational results of the objective value found with the column generation and selected heuristic route finding methods. A graphical representation of the convergence of solution and the number of iterations needed for a small time horizon problem is shown in Figure 6.1. The number of columns needed to be generated in the subproblem for $|T| = 100$ using actual data was 5163. The route based model used is 5.1.1. The heuristic models used are 5.2.3 and 5.2.5. Model 5.2.3 is tested since it creates routes with no regard to work load on routes. Model 5.2.5 is tested since it creates
routes that limit the work load on routes.

| $|T|$ | Column Generation | Model 5.2.3 | Model 5.2.5 |
|---|---|---|---|
| 42 | 13018 | 13051 | 14061 |
| 100 | 17607 | 18074 | 21988 |
| Full Data | 47233† | 46126 | 53872 |

Table 6.1: Computational comparison of the Route Based model and Column Generation for actual data. †This simulation did not converge at all.

Figure 6.1: Iterations needed to find an optimal solution for $|T| = 100$ using Column Generation and actual data

As the results show from Table 6.1 the heuristic method for creating routes was actually close to optimal. When column generation showed convergence the route based model was within $\approx 2\%$. This could be due to a number of reasons. The value of the training cost is very small in comparison to the transfer costs and the start costs. Symmetry could also exist in the data. The symmetry is a consequence of the rules for which the qualification matrix was built, particularly that if a leader was qualified to lead a trip in a certain region then they were considered qualified to lead any trip in that region. Thus no routes generated in the subproblem were pricing out favorably because there was too much similarity between the leaders. To combat this and the comparatively small values of training cost, the use of random parameters for the training costs and qualification matrix
were used.

The randomization of the qualification matrix is hoped to break any symmetry, thus enabling column generation to demonstrate theoretically how sub-optimal the route based model was. The start cost was reduced so that it was small in relation to the training costs. Refer to Appendix A for a discussion on the randomized data. Table 6.2 shows the computational results for the randomized data set. Figure 6.2 shows the number of iterations needed to obtain convergence for a selected time horizon. The number of columns needed to be generated in the subproblem for $|T| = 100$ using random data was 1198.

| $|T|$  | Column Generation |  |  | Model 5.2.3 |  |  | Model 5.2.5 |  |
|------|-------------------|---|---|-------------|---|---|-------------|---|
|      | Objective Value   | # Routes | # Columns Added | Objective Value | # Routes | Objective Value | # Routes |
| 42   | 6146.33           | 21       | 240             | 6416.33        | 22       | 7952.565      | 23       |
| 100  | 8350.33           | 40       | 1198            | 9343.66        | 42       | 10380.677     | 43       |
| Full Data | 17563.6‡  | 47       | 9333‡           | 73594.4‡       | 49       | 41825         | 51       |

Table 6.2: Computational comparison of the Route Based model and Column Generation for randomized data. ‡This simulation did not converge to optimal, refer to Appendix B

![Graph](image)

Figure 6.2: Iterations needed to find an optimal solution for $|T| = 100$ using Column Generation and random data.

The column generation with randomized parameters shows that actually at small $|T|$ the route
based model is only $\approx 10\%$ greater than the optimal value found by column generation. However using the full data set the difference increased to $\approx 75\%$. This means that if the column generation procedure finds the optimal solution which would have been found by the complete model (which will only be verified with small time horizons,) then the route based model is dependent in terms of viability to obtain a good enough solution on the actual data used. This means that possibly the actual structure of the problem is not a factor in determining the viability. Fundamentally this is due to the value of the training costs being relatively small compared to other costs.

6.3.2 Comparison of Column Generation and Complete Model

A comparison between column generation and the complete model 4.2.2 was done in Table 6.3. This was to determine how the column generation model was performing compared to the complete model 4.2.2. If the column generation model found the same integer objective value as in the complete model for small time horizons then it was assumed that it would perform the same with larger $|T|$. To make this comparison a start cost was introduced for each leader in the complete model 4.2.2 as in Section 5.5, such that the objective functions are the same. This was done by modifying Equation (4.14) to be (4.14) + $\sum_{l \in L} \sum_{i \in T} StartCostZ_{li}$

| $| T |$ | Column Generation | Complete Model |
|---|---|---|---|
| LP Relaxation | Integer Solution | # Routes | LP Relaxation | Integer Solution | # Routes |
| 30 | 2881 | 2881 | 14 | 2881 | 2881 | 14 |
| 42 | 6146.33 | 6146.33 | 24 | 6146.33 | 6146.33 | 24 |
| 100 | 8210.663 | 8350.33 | 42 | Empty Memory | - | - |

Table 6.3: Computational comparison of the Complete model and Column Generation for randomized data.

The results of Table 6.3 show that actually the column generation model is finding the global integer optimal solution for selected small cardinality of $T$. If the assumption is made that column generation will always find this optimal then the route based model not considering work load preferences for actual data with the instances tested is a good approximation, within 10%.
Chapter 7

Concluding Remarks

7.1 Results and Outcomes

The research provided some interesting results in terms of the viability of the route based model and heuristic methods to create routes. With the current actual data of training costs, and not considering a leader work load preference, the route based model proved to be a good approximation. This was because the level of the training cost in comparison to the transfer costs and start costs was small. Also the symmetry of the current qualification data meant that few new routes could possibly be created, so column generation was not needed. Since the data was accumulated as an approximation then the correctness of this was tested by using random data. The column generation model when used with the full random data and with larger transfer costs and smaller start costs did actually prove the route based model 5.1.1 using heuristic models 5.2.3 and 5.2.5 to be unacceptable, with a difference in objective value of $\approx 75\%$. Using the computational information in Chapters 4 and 6 the recommendation is that the route based model and heuristics are acceptable if the average training cost is smaller than the average transfer cost by a factor of less than $\approx 3$.

The route based model 5.1.1 and route generating heuristic 5.2.3 is a good and very quick (4 sec) approximation when the training costs are suitably small. The heuristic model 5.2.5 when used with the route based model 5.1.2 can cover all factors of the problem including leader work load preference and a limit on the days worked in a row.

In theory the code developed in mosel could be used in simple Home Health Care problems where the same costs and penalties apply.
7.2 Future Research

The thesis has created a good model for the TLSP. The model could be further developed to include such features as, a column generation approach with a cost penalty on the desired work load to compare with the route based model and heuristic methods, the incorporation of bidding for routes by leaders as in airline scheduling, modeling the effects that the quality of leaders has on revenue, and modifying the model to include training during the season.
Bibliography


Appendix A

Implementation and Interfaces

Implementation

*Xpress-MP* Notes

In *Xpress-MP* when using an *if* statement on the value of a decision variable it was important to use a tolerance on equalities due to computational rounding. For example in the code when executing an equality it was turned into an inequality by using a value of *ZERO_TOL*. The numerical value of *ZERO_TOL* chosen was 0.001 since the error on a binary variable is predicted to be no greater than $10^{-4}$. An example of the use of this can be seen below where the *if* statement is executed *iff* \( \text{Assign\_TripPair}(i, j) = 0 \). Parameters are not subject to this rounding error.

\[
\text{forall}(j \text{ in TRIPS}) \text{ do}
\begin{align*}
&\text{if getsol(sum}(i \text{ in TRIPS}) \text{ Assign\_TripPair}(i, j)) < \text{ZERO\_TOL} \text{ then} \\
&\quad \text{FirstTripSet} := \text{FirstTripSet} + j \\
&\text{end-if}
\end{align*}
\text{end-do}
\]

The Gap is part of the statistical output from *Xpress-MP* it is defined as

\[
\frac{\text{Integer Solution Found} - \text{Best Integer Bound Found}}{\text{Best Integer Bound Found}}
\]

at the current time the algorithm is exited. If a computation is exited at the same run time for two
different models say, and the Gap is larger for one model, then it was assumed that the run time to optimality would be larger for the method that exited with the largest value of Gap.

Data

The following data was compiled in collaboration with the client. These data sets form the parameters of the model:

- A complete list of the Trip Codes with accompanying finishing date.
  This provided the basis for how to structure the mathematical program. The complete trip set is a list of 675 trips. In some instances this data set is reduced to solve the model in a restricted time horizon, enabling the analysis of the complete model. A time horizon of size $x$ is defined in this thesis to be the cardinality $x = |T|$. This corresponds to the first $x$ trips offered in the season ordered by their start dates $s_i$ from smallest to largest. The cardinality of the set $T$ used in each computational experiment was chosen in this way so as the trips considered were the same.

- For each Trip Type, a start city and an end city with accompanying training cost.
  This made it possible to decipher the trip codes into a start city and an end city which was needed to calculate the travel costs. There are 35 different trip types. It would have been possible for the data set to contain each individual trip with a travel cost associated with it. But since the travel cost is a function of the start and end cities, it was deemed more practical to utilize the data in this format\textsuperscript{1}.

- For each pair of possible start and end cities a transfer cost and transfer time between them.
  The client had only data on transfer costs between city pairs, produced in the preceding year’s schedule. This meant that a lot of data about other potential city pairs was missing. The missing data was estimated using the length of travel, and the size of the cities. The cost matrix was not necessarily symmetric.

- For each leader (51 of them) a list of pre season qualifications, and a desired work load.
  An initial set of qualifications was provided by the client. After collaboration with the client it was decided that if a leader was qualified on a trip in a region then they were considered to be qualified for all trips starting in that region. This is only an estimate since some trips

\textsuperscript{1}The cost is theoretically precisely known, however due to limited time on the clients part good estimates were made instead.
start in one region and end in another; however for the purposes of the model it was deemed to be very satisfactory by the client.

Other data needed in the models comprise penalty terms applied to the objective function and tolerances. These were tested on extensively to give the best results in terms of feasibility and optimality. When implementing column generation, to achieve more helpful results of the convergence characteristics of the program, random values generated from Excel using the RAND function were used for some parameters. The qualification matrix was produced with random binary values. The training cost for the trip types were randomized with value in \([600, 950]\).

**Interfaces**

A schedule was printed out so that an ongoing check of routes could be obtained. The first visualization of a solution was ultimately too 'messy' to be effectively used by the client. It was however a good source for the continuous monitoring and debugging of the code. An example schedule used as an ongoing debugging devise is shown in Table A.1

<table>
<thead>
<tr>
<th>Leader</th>
<th>Trip 1</th>
<th>Start Date</th>
<th>Trip 1</th>
<th>End Date</th>
<th>Trip 1</th>
<th>Trip 2</th>
<th>...</th>
<th>Work Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linda:</td>
<td>ARK060415</td>
<td>04/15/06</td>
<td>04/29/06</td>
<td>WSP060501</td>
<td>...</td>
<td>0.89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mandy:</td>
<td>ASJ060403</td>
<td>04/03/06</td>
<td>04/14/06</td>
<td>ASK060427</td>
<td>...</td>
<td>0.78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dympna:</td>
<td>ASJ060417</td>
<td>04/17/06</td>
<td>04/28/06</td>
<td>ASK060511</td>
<td>...</td>
<td>0.82</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.1: Example of a Schedule Output

The printout procedure works by taking in the first trip done by a leader and then computes the proceeding trips. Another function is called which utilizes the route function to find the route work load done by that leader. The work load function works by accumulating the days worked and dividing by the number of days between the start of a leaders first trip and end of a leader’s last trip.

There were some technicalities in the code related to the use of two optimization statements in the route based model. This was particularly prevalent when printing out the solution. The second optimization statement was re optimizing over all constraints, thus changing the variable \(X_{ij}\) from the first optimization. This had no effect on the solution given in the second optimization since the objective function was not a function of \(X_{ij}\). However, the code written to print the schedule made use of a function that did use the optimal values of \(X_{ij}\) to find the next trip done for a
particular leader. This was overcome by assigning the optimal $X_{ij}$ value to be a constant after the first optimization was complete. Then the function which printed the routes to leaders used the constant values rather than the variable values.

To print the created routes it was necessary to create a set $\mathcal{F}$ that contained all the first trips done for each route. The first trips done in a route are identified by the fact that no arc comes into them from the set $\mathcal{T}$. Set $\mathcal{F}$ then uniquely defines each of the routes by referring to a particular trip since every trip is covered once, no two routes will have the same start trip. The algorithm used to construct this set is shown below. Refer to Appendix B for the mosel code implemented in Xpress-MP.

<table>
<thead>
<tr>
<th>Algorithm (A1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> The optimal binary variables $X_{ij}$</td>
</tr>
<tr>
<td><strong>Output</strong> A set $\mathcal{F}$ corresponding to the start trips of each route created</td>
</tr>
<tr>
<td><strong>begin</strong></td>
</tr>
<tr>
<td>for $j \in \mathcal{T}$ do</td>
</tr>
<tr>
<td>if $\sum_{i \in \mathcal{T}} X_{ij} = 0$ then</td>
</tr>
<tr>
<td>$\mathcal{F} := \mathcal{F} \cup {j}$</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

The solution was printed in such a way that it was easily read by the client. The final stage of the problem in a practical sense was that the created schedule for a particular leader would be emailed out to Europe, and it was paramount for the schedule to be easily understood. Much consultation was done surrounding this issue and the final format was chosen as shown in Table A.2. Refer to Appendix B for the mosel code. The output shows the trips done on each day of the season. If two trips are on the same day then the first trip ends on that day and the next trip starts on that day, for example Daniel on the 06/17/06 finishes ARK060603 and starts WRH060617.
<table>
<thead>
<tr>
<th>Date</th>
<th>Leader</th>
<th>Days Worked</th>
<th>Days Elapsed</th>
<th>WorkLoad</th>
</tr>
</thead>
<tbody>
<tr>
<td>06/11/06</td>
<td>ARK060603</td>
<td>160</td>
<td>181</td>
<td>0.883978</td>
</tr>
<tr>
<td>06/12/06</td>
<td>ARK060603</td>
<td>150</td>
<td>169</td>
<td>0.887574</td>
</tr>
<tr>
<td>06/13/06</td>
<td>ARK060603</td>
<td>187</td>
<td>216</td>
<td>0.865741</td>
</tr>
<tr>
<td>06/14/06</td>
<td>ARK060603</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/15/06</td>
<td>ARK060603</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/16/06</td>
<td>ARK060603</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/17/06</td>
<td>ARK060603, WRH06060617</td>
<td>ZSS06060617</td>
<td>AXD06060610</td>
<td></td>
</tr>
<tr>
<td>06/18/06</td>
<td>WRH06060617</td>
<td>ZSS06060617</td>
<td>AXD06060610</td>
<td></td>
</tr>
<tr>
<td>06/19/06</td>
<td>WRH06060617</td>
<td>ZSS06060617</td>
<td>AXD06060610</td>
<td></td>
</tr>
<tr>
<td>06/20/06</td>
<td>WRH06060617</td>
<td>ZSS06060617</td>
<td>AXD06060610</td>
<td></td>
</tr>
<tr>
<td>06/21/06</td>
<td>WRH06060617</td>
<td>ZSS06060617</td>
<td>AXD06060610</td>
<td></td>
</tr>
<tr>
<td>06/22/06</td>
<td>WRH06060617</td>
<td>ZSS06060617</td>
<td>AXD06060610</td>
<td></td>
</tr>
<tr>
<td>06/23/06</td>
<td>WRH06060617</td>
<td>ZSS06060617</td>
<td>AXD06060610</td>
<td></td>
</tr>
<tr>
<td>06/24/06</td>
<td>WRH06060617</td>
<td>ZSS06060617</td>
<td>AXD06060610</td>
<td></td>
</tr>
<tr>
<td>06/25/06</td>
<td>WRH06060617</td>
<td>ZSS06060617, WST06060625</td>
<td>ASP060625</td>
<td></td>
</tr>
<tr>
<td>06/26/06</td>
<td>WRH06060617</td>
<td>WST06060625</td>
<td>ASP060625</td>
<td></td>
</tr>
<tr>
<td>06/27/06</td>
<td>WRH06060617</td>
<td>WST06060625</td>
<td>ASP060625</td>
<td></td>
</tr>
</tbody>
</table>

Table A.2: Sample final schedule
Appendix B

Miscellaneous Code & Figures

This code creates the network by only creating variables in this environment.

forall(i in TRIPS) do
  forall(j in TRIPS) do
    if (TripStart(j) - TripEnd(i) >=
      Transfer_Time(End_City(TripType(i)), Start_City(TripType(j)))) then
      if (TripStart(j) - TripEnd(i) <= MAX_GAP) then
        create(Assign_TripPair(i,j))
        Assign_TripPair(i,j) is_binary
      end-if
    end-if
  end-do
end-do

This code creates the parameter $\hat{T}_{ij}$

forall(l in LEADERS) do
  forall(j in ROUTES) do
    TrainCost(l,j):=0
    forall(k in TRIPTYPES) do
      if (Qualification(l,k)=0 and Schedule_Prop(j,k)=1) then
        TrainCost(l,j)+=TrainingCost(k)
      end-if
    end-do
  end-do
end-do

This code creates the first trips done, set $\mathcal{F}$
procedure Create_First_Trips.Done
forall(j in TRIPS) do
  if getsol(sum(i in TRIPS | exists(Assign_TripPair(i,j)))
    Assign_TripPair(i,j)) <= ZERO_TOL then
    FirstTripSet:= FirstTripSet + j
  end-if
end-do
end-procedure

This code prints the schedule in the most coherent style

MaxDate:= getsol(max(k in TRIPS) TripEnd(k))
MinDate:= getsol(min(k in TRIPS) TripStart(k))
writeln
writeln
write(strfmt(" ", -30))
forall(l in LEADERS) do
  write(strfmt(";",-2))
  write(strfmt(l,-28))
end-do
writeln
writeln
write(strfmt(" ",-30))
forall(l in LEADERS) do
  count:=0
 forall(j in FirstTripSet) do
    if(getsol(Assign_Schedule(l,j)> ZERO_TOL ) then
      write(strfmt(";",-2))
      write(strfmt("Days Worked: ",-14))
      write(strfmt(DaysOnRoute(j),-14))
      count:=count+1
    end-if
  end-do
  if(count=0) then
    write(strfmt(";",-30))
  end-if
end-do
writeln
write(strfmt(" ",-30))
forall(l in LEADERS) do
  count:=0
 forall(j in FirstTripSet) do
    if(getsol(Assign_Schedule(l,j)> ZERO_TOL ) then
      write(strfmt(";",-2))
    end-if
  end-do
end-do
writeln
write(strfmt(" ",-30))
forall(l in LEADERS) do
  count:=0
 forall(j in FirstTripSet) do
    if(getsol(Assign_Schedule(l,j)> ZERO_TOL ) then
      write(strfmt(";",-2))
    end-if
  end-do
end-do

82
write(strfmt("Days Elapsed: ",-14))
write(strfmt(DaysElapsedOnRoute(j),-14))
count:=count+1
end-if
end-do
if(count=0) then
write(strfmt(";",-30))
end-if
end-do
writeln
write(strfmt(" ",-30))
forall(l in LEADERS) do
count:=0
forall(j in FirstTripSet) do
if(getsol(Assign_Schedule(l,j))> ZERO_TOL ) then
write(strfmt(";",-2))
write(strfmt("WorkLoad: ",-14))
write(strfmt(Schedule_Workload(j),-14))
count:=count+1
end-if
end-do
if(count=0) then
write(strfmt(";",-30))
end-if
end-do
writeln
while(MinDate<=MaxDate) do
write(strfmt(MinDate,-30))
forall(l in LEADERS) do
counter:= 0
forall( i in TRIPS | getsol(Leaders_Trips(l,i)) >= ZERO_TOL ) do
if( (MinDate<= TripEnd(i)) AND (MinDate >= TripStart(i)) ) then
if(counter=0) then
write(strfmt(";",-2))
write(strfmt(i,-13))
elif (counter = 1) then
write(strfmt(";",-2))
write(strfmt(i,-13))
end-if

counter:= counter + 1
end-if
end-do
if(counter=0) then
write(strfmt(";",-30))
end-if
Figure B.1 shows the start of column generation convergence for the full random data set.

![Figure B.1: Convergence of Column Generation for full random data.](image-url)
Appendix C

Sorting Algorithm

Donald Shell in 1959 invented the shell sort, it is the most efficient of the \( O(n^2) \) algorithms. This code is from [10]
j:=i  
while (ANum(j-inc)>v) do ! Inner loop of straight insertion  
    ANum(j):=ANum(j-inc)  
    j -= inc  
    if j<=inc then  
        break;  
    end-if  
end-do  
ANum(j):= v  
end-do  

until (inc<=1)  
Flow control constructs 41 Mosel User Guide  
writeln("Ordered list: ")  
forall(i in 1..N) write(ANum(i), " ")  
writeln  
end-model
Appendix D

BigM' Algorithm

forall(i in TRIPS) do
    create_longest_path_trip(i)
end-do

procedure create_longest_path_trip(i)(CurrentNode: string)

declarations
    Max_Length: real
end-declarations
Max_Length:=0
forall(j in TRIPS) do
    if (TripStart(CurrentNode)-TripEnd(j) >= Transfer_Time(End_City(TripType(j)),Start_City(TripType(CurrentNode)))) then
        if (TripStart(CurrentNode) - TripEnd(j) <= MAX_GAP) then
            if(Max_Length< Longest_Path(j)) then
                Max_Length:= Longest_Path(j)
            end-if
        end-if
    end-if
end-do
Longest_Path(CurrentNode):= Max_Length + (TripEnd(CurrentNode)-TripStart(CurrentNode))
end-procedure
Appendix E

Big$M''$ Algorithm

forall(i in TRIPS) do
    BigMM(i)
end-do

procedure BigMM(Node: string)

declarations
    Max_Length: real
end-declarations

Max_Length:=0
forall(j in TRIPS) do
    if (TripStart(Node)-TripEnd(j) >=
        Transfer_Time(End_City(TripType(j)),Start_City(TripType(Node)))) then
        if (TripStart(Node) - TripEnd(j) <= MAX_GAP) then
            if(Max_Length< (Days_Path(j)+ (TripStart(Node)-TripEnd(j))) )then
                Max_Length:= Days_Path(j)+ (TripStart(Node)-TripEnd(j))
            end-if
        end-if
    end-if
end-do

Days_Path(Node):= Max_Length

end-procedure
Appendix F

Complete Model Code

model "Complete Code"
uses "mmxprs", "mmive"

parameters
  DATAFILE= "AllDATANONHOMOSP.mos"
  ZERO_TOL= .001
end-parameters

declarations
!sets
  !sets
    MAX_GAP: integer
    CurrentNode: string
    TRIPS: set of string
    TRIPTYPES: set of string
    LEADERS: set of string
    CITIES: set of string

!parameters
  TripType: array(TRIPS) of string
  TripStart: array(TRIPS) of real
  TripEnd: array(TRIPS) of real
  Start_City: array(TRIPTYPES) of string
  End_City: array(TRIPTYPES) of string
  Transfer_Cost: array(CITIES, CITIES) of real
  Transfer_Time: array(CITIES, CITIES) of real
  Work_Load: array(LEADERS) of real
  Qualification: array(LEADERS, TRIPTYPES) of integer
  TrainingCost: array(TRIPTYPES) of real
end-declarations
!Input Data
initializations from DATAFILE
MAX_GAP
Qualification as 'Leader_Info'
Work_Load as 'Leader_Pref'
[TripType, TripStart, TripEnd] as 'Trip_Info'
[Start_City, End_City, TrainingCost] as 'TripType_Info'
[Transfer_Cost, Transfer_Time] as 'City_Info'
end-initializations

finalize(TRIPS)
finalize(TRIPTYPES)
finalize(LEADERS)
finalize(CITIES)

declarations
!Variables
Assign_TripPair: dynamic array(LEADERS, TRIPS, TRIPS) of mpvar
Last_Trip: dynamic array(LEADERS, TRIPS) of mpvar
First_Trip: dynamic array(LEADERS, TRIPS) of mpvar
Assign_Trip: dynamic array(LEADERS, TRIPS) of mpvar
P: array(LEADERS,TRIPTYPES) of mpvar
Consecutive_Days: dynamic array(LEADERS, TRIPS) of mpvar
end-declarations

!declaration of procedures and functions
forward procedure Create_Assign_TripPair
forward procedure Print_Schedule
forward function NextTrip(l: string, CurrentNode: string): string
forward function LOAD(l: string): real

!pre-processing
Create_Assign_TripPair

!Constraints

!Must leave every trip only once
forall(l in LEADERS, i in TRIPS | exists(Assign_Trip(l,i))) do
  MustLeaveOnce(l,i):=
  Assign_Trip(l,i) = sum(j in TRIPS) Assign_TripPair(l,i,j) + Last_Trip(l,i)
end-do

!Must Enter every trip only once
forall(l in LEADERS, j in TRIPS | exists(Assign_Trip(l,j))) do
MustEnterOnce(l,j):=
Assign_Trip(l,j) = sum(i in TRIPS) Assign_TripPair(l,i,j) + First_Trip(l,j)
end-do

!Must start and finish only once on a particular Trip
forall(l in LEADERS) do
    MustStartOnce(l):=
    sum(i in TRIPS) First_Trip(l,i) <= 1
    MustFinishOnce(l):=
    sum(i in TRIPS) Last_Trip(l,i) <= 1
end-do

!If start then must finish
forall(l in LEADERS) do
    sum(i in TRIPS) First_Trip(l,i) = sum(i in TRIPS) Last_Trip(l,i)
end-do

!Every Trip is done by one leader only
forall(k in TRIPS) do
    AllTripDone(k):=
    sum(l in LEADERS) Assign_Trip(l,k) = 1
end-do

!Logic between Assign_Trip and Assign_TripPair
forall(l in LEADERS, i in TRIPS, j in TRIPS) do
    TripPairLogic1(l,i,j):=
    Assign_TripPair(l,i,j) <= Assign_Trip(l,i)
    TripPairLogic2(l,i,j):=
    Assign_TripPair(l,i,j) <= Assign_Trip(l,j)
end-do

!Train logic constraint
forall(l in LEADERS, i in TRIPS) do
    P(l,TripType(i))>=Assign_Trip(l,i)
end-do

!Work load must not exceed a specified amount of days they work
forall(l in LEADERS) do
    Workload(l):=
    sum(i in TRIPS, j in TRIPS) ((TripStart(j)-TripEnd(i)) * Assign_TripPair(l,i,j)) >=
    (1-Work_Load(l))*
    (sum(i in TRIPS) Last_Trip(l,i)*TripEnd(i) - sum(i in TRIPS)First_Trip(l,i)*TripStart(i))
end-do

!Auxiliary variable constraint to ensure no more than 70 days created in row
forall(l in LEADERS, i in TRIPS, j in TRIPS | exists(Assign_TripPair(l,i,j)) and ((TripStart(j)-TripEnd(i))<=6) ) do
  Consecutive_Days(l,j) >= Consecutive_Days(l,i) + (TripEnd(j)-TripEnd(i))-
  (70+TripStart(j)-TripEnd(i))*(1-Assign_TripPair(l,i,j))
end-do

forall(l in LEADERS, i in TRIPS) do
  Consecutive_Days(l,i)<=70
  Consecutive_Days(l,i)>= TripEnd(i)-TripStart(i)
end-do

!objective
TotalCost := sum(l in LEADERS, i in TRIPS, j in TRIPS) Assign_TripPair(l,i,j)*
Transfer_Cost(End_City(TripType(i)), Start_City(TripType(j))) +
sum(l in LEADERS,k in TRIPTYPES | Qualification(l,k)=0) TrainingCost(k)*(P(l,k))
!minimise cost
minimize(TotalCost)

writeln
Print_Schedule

!*************List of procedures and functions
!Creates the binary variable Assign_TripPair
procedure Create_Assign_TripPair
forall(l in LEADERS) do
  forall(i in TRIPS, j in TRIPS) do
    create(Assign_Trip(l,i))
    Assign_Trip(l,i) is_binary
    create(Assign_Trip(l,j))
    Assign_Trip(l,j) is_binary
    create(First_Trip(l,i))
    First_Trip(l,i) is_binary
    create(Last_Trip(l,j))
    Last_Trip(l,j) is_binary
    create(Consecutive_Days(l,i))
    Consecutive_Days(l,i) is_integer
    if (TripStart(j)-TripEnd(i) >=
    Transfer_Time(End_City(TripType(i)),Start_City(TripType(j))) ) then
      if (TripStart(j) - TripEnd(i) <= MAX_GAP) then
        create(Assign_TripPair(l,i,j))
        Assign_TripPair(l,i,j) is_binary
        writeln("Created: ", l + "," + i + "," + j)
      end-if
    end-if
  end-do
end-do
end-procedure

!post-processing
!Prints out the schedule
procedure Print_Schedule
forall(l in LEADERS) do
  forall(i in TRIPS) do
    if getsol(First_Trip(l,i))>= ZERO_TOL then
      write(l,": ",i)
      CurrentNode:= i
      while( getsol(Last_Trip(l,CurrentNode))<=ZERO_TOL) do
        CurrentNode:= NextTrip(l,CurrentNode)
        write(" ",CurrentNode)
      end-do
      write(" ",LOAD(l))
      writeln
    end-if
  end-do
end-do
writeln
writeln("Total Cost = ", getobjval)
end-procedure

!Function to get next trip done
function NextTrip(l: string, CurrentNode: string): string
declarations
  Result: string
end-declarations
Result:= " 
forall(k in TRIPS) do
  if getsol(Assign_TripPair(l,CurrentNode,k)) > ZERO_TOL then
    Result:= k
    break
  end-if
end-do
returned:= Result
end-function

!Function To determine work load
function LOAD(l: string):real
result:=((sum(i in TRIPS) ( (TripEnd(i)-TripStart(i))*getsol(Assign_Trip(l,i)) ) ) ) /
  (getsol( sum(i in TRIPS) ( (Last_Trip(l,i)*TripEnd(i))- (First_Trip(l,i)*TripStart(i))))))
returned:=  result
end-function

end-model
Appendix G

Route Based Model Code

model "Decomposition Code"
uses "mmxprs", "mmive"

parameters
    DATAFILE= "ALLDATANONHOMOSP.mos"
    ENDNODE= "ENDTRIP"
    dayslimit=70
    daysoff=6
    ZERO_TOL= 0.001
end-parameters

declarations
    !sets
        MAX_GAP: integer
        MIN_GAP: integer
        Number_of_leaders: integer
        Work_Load: real
        Start_Cost: real
        MyNextTrip: string
        Pref_TOL: real
        Penalty_Level: real
        TRIPS: set of string
        ALLNODES: set of string
        TRIPTYPES: set of string
        CITIES: set of string
        FirstTripSet: set of string
        LEADERS: set of string
    !parameters
        TripType: array(TRIPS) of string
TripStart: array(TRIPS) of real
TripEnd: array(TRIPS) of real
TrainingCost: array(TRIPTYPES) of real
Start_City: array(TRIPTYPES) of string
End_City: array(TRIPTYPES) of string
Transfer_Cost: array(CITIES, CITIES) of real
Transfer_Time: array(CITIES, CITIES) of real
Qualification: array(LEADERS, TRIPTYPES) of integer
WorkPref: array(LEADERS) of real
Longest_Path: array(TRIPS) of real
Days_Path: array(TRIPS) of real
Assign_TripPairVAL: dynamic array(TRIPS, ALLNODES) of real
Assign_Days_WorkedVAL: dynamic array(TRIPS, ALLNODES) of real
Assign_Days_OffVAL: dynamic array(TRIPS, ALLNODES) of real
end-declarations

!Input Data
initializations from DATAFILE
MAX_GAP
Work_Load
Start_Cost
Pref_TOL
Penalty_Level
Qualification as 'Leader_Info'
WorkPref as 'Leader_Pref'
[TripType, TripStart, TripEnd] as 'Trip_Info'
[Start_City, End_City, TrainingCost] as 'TripType_Info'
[Transfer_Cost, Transfer_Time] as 'City_Info'
end-initializations

finalize(TRIPS)
finalize(TRIPTYPES)
finalize(CITIES)
finalize(LEADERS)

CountLEADERS := 0
ALLNODES:= TRIPS + ENNODE

declarations
!Variables
Assign_TripPair: dynamic array(TRIPS, ALLNODES) of mpvar
Assign_Days_Worked: dynamic array(TRIPS, ALLNODES) of mpvar
Assign_Days_Off: dynamic array(TRIPS, ALLNODES) of mpvar
Consecutive_Days: dynamic array(TRIPS) of mpvar
end-declarations
!declaration of procedures and functions
forward procedure Create_Assign_TripPair
forward procedure Print_TripPairs_Done
forward procedure Print_Schedule
forward procedure Print_Optimisation1_Info
forward procedure Create_First_Trips_Done
forward procedure BigM(CurrentNode: string)
forward procedure BigMM(Node: string)
forward procedure NextTrip(MyNextTrip: string): string
!pre-processing
Create_Assign_TripPair

!Creates the Big M constraint for Days worked
forall(i in TRIPS) do
  BigM(i)
end-do

!Creates the Big M constraint for Days Off
forall(i in TRIPS) do
  BigMM(i)
end-do

!Counts the number of leaders available
CountLEADERS:= getSize(LEADERS)

!Constraints
!Must leave every trip only once
forall(i in TRIPS ) do
  !MustLeaveOnce(i):=
  sum(j in ALLNODES) Assign_TripPair(i,j) = 1
end-do

!Must Enter every trip only once or not at all if first trip, except ENDTRIP
forall(j in TRIPS ) do
  !MustEnterOnce(j):=
  sum(i in TRIPS) Assign_TripPair(i,j) <= 1
end-do

!The amount of routes is limited by the number of leaders available
!leadersaval:=
  sum(i in TRIPS) Assign_TripPair(i,ENDNODE)<= CountLEADERS

!Constraint to calculate Days worked on a route
forall(j in TRIPS) do

!workload(j):= 
sum(k in ALLNODES) Assign_Days_Worked(j,k) = sum(i in TRIPS) Assign_Days_Worked(i,j) + 
(TripEnd(j)-TripStart(j))
end-do

!Big M Constraint for Days worked on a route
forall(i in TRIPS, j in ALLNODES | exists(Assign_Days_Worked(i,j))) do
!workload1(i,j):=
Assign_Days_Worked(i,j) <=Longest_Path(i) * Assign_TripPair(i,j)
end-do

!Constraint to calculate Days off on a route
forall(j in TRIPS) do
!workload(j):=
sum(k in ALLNODES) Assign_Days_Off(j,k) = sum(i in TRIPS) (Assign_Days_Off(i,j) + 
( (TripStart(j)-TripEnd(i))*Assign_TripPair(i,j)))
end-do

!Big M Constraint for Days Off on a route
forall(i in TRIPS, j in ALLNODES | exists(Assign_Days_Off(i,j))) do
!workload1(i,j):=
Assign_Days_Off(i,j) <= Days_Path(i)* Assign_TripPair(i,j)
end-do

!Work load must not exceed a specified amount of days worked on a route
forall(i in TRIPS | exists (Assign_Days_Worked(i,ENDNODE))) do
!specificdays(i):=
Assign_Days_Worked(i,ENDNODE) <= (Assign_Days_Off(i,ENDNODE) + 
Assign_Days_Worked(i,ENDNODE)) *Work_Load
end-do

!Auxiliary variable constraint to ensure no more than 70 days created in row
forall(i in TRIPS, j in TRIPS | exists(Assign_TripPair(i,j)) and
((TripStart(j)-TripEnd(i))<=daysoff) ) do
Consecutive_Days(j) >= Consecutive_Days(i) + (TripEnd(j)-TripEnd(i))- 
(70+TripStart(j)-TripEnd(i))*(1-Assign_TripPair(i,j))
end-do
forall(i in TRIPS) do
Consecutive_Days(i)<= dayslimit
Consecutive_Days(i)>= TripEnd(i)-TripStart(i)
end-do

!objective function 1st optimisation
TotalCost1 := sum(i in TRIPS, j in TRIPS) Assign_TripPair(i,j)*
Transfer_Cost(End_City(TripType(i)), Start_City(TripType(j))) +
sum( i in TRIPS) Assign_TripPair(i,ENDNODE)*Start_Cost
!minimise cost
minimize(TotalCost1)

forall(i in TRIPS, j in ALLNODES) do
  Assign_TripPairVAL(i,j):= getsol(Assign_TripPair(i,j))
end-do

forall(i in TRIPS, j in ALLNODES) do
  Assign_Days_WorkedVAL(i,j):= getsol(Assign_Days_Worked(i,j))
end-do

forall(i in TRIPS, j in ALLNODES) do
  Assign_Days_OffVAL(i,j):= getsol(Assign_Days_Off(i,j))
end-do

Print_Optimisation1_Info

!******Create a Schedule*****
Create_First_Trips_Done
Print_TripPairs_Done
Print_Schedule

!************List of procedures and functions 1st Optimisation

!Creates the binary variable Assign_TripPair and Workload
procedure Create_Assign_TripPair
forall(i in TRIPS) do
  create(Assign_TripPair(i,ENDNODE))
  Assign_TripPair(i,ENDNODE) is_binary
  create(Assign_Days_Worked(i,ENDNODE))
  create(Assign_Days_Off(i,ENDNODE))
  writeln("Created: ", i + "," + ENDNODE)
forall(j in TRIPS) do
  if (TripStart(j)-TripEnd(i) >= Transfer_Time(End_City(TripType(i)),Start_City(TripType(j)))) then
    if (TripStart(j) - TripEnd(i) <= MAX_GAP) then
      create(Assign_TripPair(i,j))
      Assign_TripPair(i,j) is_binary
      create(Assign_Days_Worked(i,j))
      create(Assign_Days_Off(i,j))
      writeln("Created: ", i + "," + j)
!Creates the longest path in the network for the big M constraint Days Worked
procedure BigM(CurrentNode: string)

declarations
Max_Length: real
end-declarations

Max_Length:=0
forall(j in TRIPS) do
  if (TripStart(CurrentNode) - TripEnd(j) >= Transfer_Time(End_City(TripType(j)),Start_City(TripType(CurrentNode)))) then
    if (TripStart(CurrentNode) - TripEnd(j) <= MAX_GAP) then
      if(Max_Length< Longest_Path(j) )then
        Max_Length:= Longest_Path(j)
      end-if
    end-if
  end-if
end-do
Longest_Path(CurrentNode):= Max_Length + (TripEnd(CurrentNode)-TripStart(CurrentNode))
end-procedure

!Creates the longest path in the network for the big M constraint Days Off
procedure BigMM(Node: string)

declarations
Max_Length: real
end-declarations

Max_Length:=0
forall(j in TRIPS) do
  if (TripStart(Node) - TripEnd(j) >= Transfer_Time(End_City(TripType(j)),Start_City(TripType(Node)))) then
    if (TripStart(Node) - TripEnd(j) <= MAX_GAP) then
      if(Max_Length< (Days_Path(j)+ (TripStart(Node)-TripEnd(j)))) )then
        Max_Length:= Days_Path(j)+ (TripStart(Node)-TripEnd(j))
      end-if
    end-if
  end-if
end-do
end-do
Days_Path(Node):= Max_Length
end-procedure

!Prints out the created schedule for each route
procedure Print_Schedule
writeln
forall(j in FirstTripSet) do
MyNextTrip:= NextTrip(j)
write(j, "(" , TripStart(j), ",", TripEnd(j), ") ")
if MyNextTrip = ENDNODE then
write(ENDNODE)
write(" ",getsol(Assign_Days_Worked(j,ENDNODE))/
(getsol((Assign_Days_Off(j,ENDNODE) + Assign_Days_Worked(j,ENDNODE))))
else
write(MyNextTrip, "(" ,TripStart(MyNextTrip), ",", TripEnd(MyNextTrip), ") ")
end-if
while(MyNextTrip <> ENDNODE) do
Previous_Trip:= MyNextTrip
MyNextTrip:= NextTrip(MyNextTrip)
if MyNextTrip = ENDNODE then
write(ENDNODE)
write(" ",getsol(Assign_Days_Worked(Previous_Trip,ENDNODE))/
(getsol((Assign_Days_Off(Previous_Trip,ENDNODE) +
Assign_Days_Worked(Previous_Trip,ENDNODE))))
else
write(MyNextTrip)
write(" ",TripStart(MyNextTrip), ",")
write(TripEnd(MyNextTrip), ") ")
end-if
end-do
writeln
end-do
end-procedure

!Creates the set of first trips done
procedure Create_First_Trips_Done
forall(j in TRIPS) do
if getsol(sum(i in TRIPS | exists(Assign_TripPair(i,j))) Assign_TripPair(i,j) <
ZERO_TOL then
FirstTripSet:= FirstTripSet + j
end-if
end-do
end-procedure
!Prints out the Trip Pairs Done
procedure Print_TripPairs.Done
writeln
forall(i in TRIPS, j in ALLNODES)do
  if getsol(Assign_TripPair(i,j)) > ZERO_TOL then
    writeln("Trip Pairs done: ", i + "," + j)
  end-if
end-do
end-procedure

!Prints Out the 1st Optimisation information
procedure Print_Optimisation1_Info
writeln
writeln("Total Cost to create schedule = ", getobjval)
writeln("Number of leaders used = ", getsol(sum(i in TRIPS) Assign_TripPair(i,ENDNODE)))
writeln("seasonlength ", getsol(max(k in TRIPS) TripEnd(k) - min(k in TRIPS) TripStart(k)))
end-procedure

!Function to determine a route: Returns next Trip in the route
function NextTrip(MyNextTrip: string): string
declarations
  Result: string
end-declarations

Result:= " 
forall(k in ALLNODES) do
  if Assign_TripPairVAL(MyNextTrip,k) > ZERO_TOL then
    Result:= k
    break
  end-if
end-do
returned:= Result
end-function

!******Second optimisation to assign leaders to routes*******

declarations
!Variables
  Assign_Schedule: array(LEADERS, FirstTripSet) of mpvar
  Schedule_Workload: array(FirstTripSet) of real
!Parameters
  Schedule_Prop: dynamic array(FirstTripSet, TRIPTYPES) of integer
  Penalty: array(LEADERS,FirstTripSet) of real
end-declarations
forward procedure Create_Schedule.Prop
forward procedure Print_Optimisation2_Info
forward procedure Print_Leaders_to_Routes
forward procedure Create_Trip_Work_Load

Create_Schedule.Prop
Create_Trip_Work_Load

!Creates a binary variable between leaders and each schedule produced
forall(i in LEADERS, j in FirstTripSet) do
  Assign_Schedule(i,j) is_binary
end-do

!Every Schedule is done by exactly one leader only
forall(j in FirstTripSet) do
  sum(i in LEADERS) Assign_Schedule(i,j) = 1
end-do

!A leader can only do one schedule or none at all
forall(i in LEADERS) do
  sum(j in FirstTripSet) Assign_Schedule(i,j) <= 1
end-do

!Constraint to allow pref on work load
forall(l in LEADERS, j in FirstTripSet) do
  if ((WorkPref(l) + Pref_TOL) >= Schedule_Workload(j)) and
    (WorkPref(l) - Pref_TOL <= Schedule_Workload(j)) then
    Penalty(l,j) := 0
  else
    if ((WorkPref(l) + Pref_TOL) <= Schedule_Workload(j)) then
      Penalty(l,j) := Penalty_Level * (Schedule_Workload(j) - (WorkPref(l) + Pref_TOL))
    else
      Penalty(l,j) := Penalty_Level * ((WorkPref(l) - Pref_TOL) - Schedule_Workload(j))
    end-if
  end-if
end-do

!Total Cost to assign leaders to routes
TotalCost2 := sum(i in LEADERS, j in FirstTripSet, t in TRIPTYPES) ((TrainingCost(t) *
  (maxlist(0,Schedule_PROP(j,t)-Qualification(i,t)))*(Assign_Schedule(i,j)))) +
  sum(i in LEADERS, j in FirstTripSet) Penalty(i,j)*Assign_Schedule(i,j)
!minimise cost
minimize(TotalCost2)

Print_Optimisation2_Info
Print_Leaders_to_Routes

!******List of procedures and functions 2st Optimisation

!Created Route Properties by comparing the triptypes within the route.
procedure Create_Schedule_Prop
for all (j in FirstTripSet) do
  MyNextTrip := NextTrip(j)
  Schedule_Prop(j,TripType(j)) := 1
  if MyNextTrip <> ENDNODE then
    Schedule_Prop(j,TripType(MyNextTrip)) := 1
  end-if
  while (MyNextTrip <> ENDNODE) do
    Previous_Trip := MyNextTrip
    MyNextTrip := NextTrip(MyNextTrip)
    Schedule_Prop(j,TripType(Previous_Trip)) := 1
  end-do
end-do
end-procedure

!Procedure to create Trip Work load
procedure Create_Trip_Work_Load
for all (j in FirstTripSet) do
  MyNextTrip := NextTrip(j)
  if MyNextTrip = ENDNODE then
    Schedule_Workload(j) := Assign_Days_WorkedVAL(j,ENDNODE)/((Assign_Days_OffVAL(j,ENDNODE) + Assign_Days_WorkedVAL(j,ENDNODE))
  end-if
  while (MyNextTrip <> ENDNODE) do
    Previous_Trip := MyNextTrip
    MyNextTrip := NextTrip(MyNextTrip)
    if MyNextTrip = ENDNODE then
      Schedule_Workload(j) := Assign_Days_WorkedVAL(Previous_Trip,ENDNODE)/((Assign_Days_OffVAL(Previous_Trip,ENDNODE) + Assign_Days_WorkedVAL(Previous_Trip,ENDNODE))
    end-if
  end-do
end-do
end-procedure

!Prints the 2nd optimisation information
procedure Print_Optimisation2_Info
writeln
writeln("Total Cost to Assign Leaders = ", getobjval)
end-procedure
!Prints the leaders assigned to each route
procedure Print_Leaders_to_Routes

writeln
forall(j in FirstTripSet) do
  forall(l in LEADERS | getsol(Assign_Schedule(l,j)) > ZERO_TOL) do
    write(l,"":"")
  end-do
  MyNextTrip:= NextTrip(j)
  write(j," ",TripStart(j)," ",TripEnd(j)," ")
  if MyNextTrip = ENDNODE then
    write(ENDNODE)
    write(" ",Assign_Days_WorkedVAL(j,ENDNODE)/((Assign_Days_OffVAL(j,ENDNODE) + Assign_Days_WorkedVAL(j,ENDNODE))))
  else
    write(MyNextTrip," ",TripStart(MyNextTrip)," ",TripEnd(MyNextTrip)," ")
  end-if
  while(MyNextTrip <> ENDNODE) do
    Previous_Trip:= MyNextTrip
    MyNextTrip:= NextTrip(MyNextTrip)
    if MyNextTrip = ENDNODE then
      write(ENDNODE)
      write(" ",Assign_Days_WorkedVAL(Previous_Trip,ENDNODE)/((Assign_Days_OffVAL(Previous_Trip,ENDNODE) + Assign_Days_WorkedVAL(Previous_Trip,ENDNODE))))
    else
      write(MyNextTrip)
      write(" ",TripStart(MyNextTrip)," ")
    end-if
  end-do
writeln
end-do
writeln
forall(i in LEADERS, j in FirstTripSet | getsol(Assign_Schedule(i,j)) >ZERO_TOL ) do
  writeln(i," ",j," ",WorkPref(i))
end-do
end-procedure
end-model
Appendix H

Column Generation Code

model "Tour Scheduling column generation"
uses "mmxprs", "mmive"
!******** Initialise with two-stage solver - use solution to create U_l,j
parameters
   DATAFILE= "ALLDATANONHOMOSP.mos"
   ENDNODE= "ENDTRIP"
   ZERO_TOL= 0.001
end-parameters

declarations
!sets
   MAX_GAP: integer
   Start_Cost: real
   MyNextTrip: string
   TRIPS: set of string
   ALLNODES: set of string
   TRIPTYPES: set of string
   CITIES: set of string
   FirstTripSet: set of string
   LEADERS: set of string
   ROUTES: set of integer !do not finalize this
!parameters
   TripType: array(TRIPS) of string
   TripStart: array(TRIPS) of real
   TripEnd: array(TRIPS) of real
   TrainingCost: array(TRIPTYPES) of real
   Start_City: array(TRIPTYPES) of string
   End_City: array(TRIPTYPES) of string
   Transfer_Cost: array(CITIES, CITIES) of real
   Transfer_Time: array(CITIES, CITIES) of real
Qualification: array(LEADERS, TRIPTYPES) of integer
WorkPref: array(LEADERS) of real

!variables
schedule: dynamic array(ROUTES, TRIPS) of integer
Schedule_Prop: dynamic array(ROUTES, TRIPTYPES) of integer
RouteTrips: dynamic array(ROUTES, TRIPS) of integer
Pairsdone: dynamic array(ROUTES, TRIPS, TRIPS) of integer
TrainCost: dynamic array(LEADERS, ROUTES) of real
RouteTransferCost: dynamic array(ROUTES) of real
dualeverytrip: dynamic array(TRIPS) of real
dualeveryroute: dynamic array(LEADERS) of real

end-declarations

!Input Data
initializations from DATAFILE
MAX_GAP
Start_Cost
Qualification as 'Leader_Info'
WorkPref as 'Leader_Pref'
[TripType, TripStart, TripEnd] as 'Trip_Info'
[Start_City, End_City, TrainingCost] as 'TripType_Info'
[Transfer_Cost, Transfer_Time] as 'City_Info'
end-initializations

finalize(TRIPS)
finalize(TRIPTYPES)
finalize(CITIES)
finalize(LEADERS)

ALLNODES := TRIPS + ENDNODE

declarations
!Variables
Assign_TripPair: dynamic array(TRIPS, ALLNODES) of mpvar
Assign_Leaders: dynamic array(LEADERS, ROUTES) of mpvar

!For colgen subprob
Traintrips: array(TRIPTYPES) of mpvar
end-declarations

!declaration of procedures and functions
forward procedure Create_Assign_TripPair
forward procedure Create_First_Trips_Done
forward procedure Create_Schedule_Prop
forward function NextTrip(MyNextTrip: string): string
!!!Pre-processing
Create_Assign_TripPair

!Counts the number of leaders available
CountLEADERS:= getsize(LEADERS)

!Constraints
!Must leave every trip only once
forall(i in TRIPS ) do
    MustLeaveOnce(i):=
    sum(j in ALLNODES) Assign_TripPair(i,j) = 1
end-do

!Must enter every trip only once or not at all if first trip, except ENDTRIP
forall(j in TRIPS ) do
    MustEnterOnce(j):=
    sum(i in TRIPS) Assign_TripPair(i,j) <= 1
end-do

!The amount of routes is limited by the number of leaders available
leadersaval:=
    sum(i in TRIPS) Assign_TripPair(i,ENDNODE)<= CountLEADERS

!Objective function 1st optimisation
TotalCost1 := sum(i in TRIPS, j in TRIPS) Assign_TripPair(i,j)*
    Transfer_Cost(End_City(TripType(i)), Start_City(TripType(j))) +
    sum( i in TRIPS) Assign_TripPair(i,ENDNODE)*Start_Cost

!minimise cost
minimize(TotalCost1)

!Turn off the constraints in first optimisation
forall(i in TRIPS) sethidden(MustLeaveOnce(i), true)
forall(j in TRIPS) sethidden(MustEnterOnce(j), true)
sethidden(leadersaval, true)

Create_First_Trips_Done
Create_Schedule_Prop

!Creates the variable Assign_TripPair
procedure Create_Assign_TripPair
forall(i in TRIPS) do
    create(Assign_TripPair(i,ENDNODE))
    Assign_TripPair(i,ENDNODE) is_binary
forall(j in TRIPS) do
  if (TripStart(j) - TripEnd(i) >= Transfer_Time(End_City(TripType(i)), Start_City(TripType(j)))) then
    if (TripStart(j) - TripEnd(i) <= MAX_GAP) then
      create(Assign_TripPair(i,j))
      Assign_TripPair(i,j) is_binary
    end-if
  end-if
end-do
end-do
end-procedure

! Creates the set of first trips done
procedure Create_First_Trips_Done
  counter1:=0
foreach(j in TRIPS) do
  if getsol(sum(i in TRIPS | exists(Assign_TripPair(i,j))) Assign_TripPair(i,j)) <= ZERO_TOL then
    counter1:= counter1+1
    ! FirstTripSet:= FirstTripSet + j
    schedule(counter1,j):=1
  end-if
end-do
end-procedure

! Created Route Properties by comparing the triptypes within the route.
procedure Create_Schedule_Prop
forall(i in ROUTES) do
  forall(j in TRIPS) do
    if(schedule(i,j)=1) then
      MyNextTrip:= NextTrip(j)
      Schedule_Prop(i,TripType(j)):= 1
      RouteTrips(i,j):=1
      previous:=j
      while(MyNextTrip <> ENDNODE) do
        Schedule_Prop(i,TripType(MyNextTrip)):= 1
        RouteTrips(i,MyNextTrip):=1
        RouteTransferCost(i)+=
        Transfer_Cost(End_City(TripType(previous)), Start_City(TripType(MyNextTrip)))
        Pairsdone(i,previous,MyNextTrip):=1
        previous:=MyNextTrip
        MyNextTrip:= NextTrip(MyNextTrip)
      end-do
    end-if
  end-do
end-do
forall(l in LEADERS) do
  forall(j in ROUTES) do
    TrainCost(l,j):=0
    forall(k in TRIPTYPES) do
      if(Qualification(l,k)=0 and Schedule_Prop(j,k)=1 ) then
        TrainCost(l,j)+=TrainingCost(k)
      end-if
    end-do
  end-do
end-do
end-procedure

!Function to determine a route: Returns next Trip in the route
function NextTrip(MyNextTrip1: string): string
declarations
Result: string
end-declarations

Result:= " "
forall(k in ALLNODES) do
  if getsol(Assign_TripPair(MyNextTrip1,k)) > ZERO_TOL then
    Result:= k
    break
  end-if
end-do
returned:= Result
end-function

!initial set of variables to be considered in the column generation
forall(l in LEADERS) do
  forall(i in ROUTES) do
    create(Assign_Leaders(l,i))
    Assign_Leaders(l,i) is_binary
  end-do
end-do
forall(j in ROUTES) do
  forall(i in TRIPS |exists(RouteTrips(j,i)) )do
    write(i," ")
  end-do
writeln
end-do
declarations
EPS = 0.001
NewRoute: integer
end-declarations

declarations
MasterObj: linctr
SubProbObj: linctr
Training: linctr
Transfer: linctr
Start: linctr
end-declarations

forward procedure ColGen
forward function SolveSubProblem(l:string): integer

!Constraints Master Problem
forall(t in TRIPS) do
everytrip(t):=
  sum(l in LEADERS, j in ROUTES | RouteTrips(j,t)=1 ) Assign_Leaders(l,j)>=1
end-do

forall(l in LEADERS) do
everyroute(l):=
  -sum(j in ROUTES)Assign_Leaders(l,j)>=-1
end-do

Training:= sum(l in LEADERS, j in ROUTES)TrainCost(l,j)*Assign_Leaders(l,j)
Transfer:=sum(l in LEADERS, j in ROUTES)RouteTransferCost(j)*Assign_Leaders(l,j)
Start:= sum(l in LEADERS, j in ROUTES)( Start_Cost)*Assign_Leaders(l,j)

MasterObj:= Training+Transfer+Start
!Perform Column Generation
ColGen

!Perform global search
minimise(MasterObj)

**************
procedure ColGen
declarations
NumVarCreated: integer
CGiter: integer
end-declarations

setparam("zerotol", EPS) ! Set Mosel comparison tolerance
setparam("XPRS_CUTSTRATEGY", 0) ! Disable automatic cuts
setparam("XPRS_PRESOLVE", 0) ! Switch presolve off

CGiter:= 0
while(true) do
    CGiter+= 1
    writeln
    write("ITNUM_MASTEROBJ_NUMVAR: ", CGiter)
    ! writeln("Solving master prob.")
    minimise(XPRS_LIN+XPRS_PRI,MasterObj)
    write(" ", getobjval)
    savebasis(1)

    ! store dual for all constraints
    forall(t in TRIPS) dualeverytrip(t):=getdual(everytrip(t))
    forall(l in LEADERS) dualeveryroute(l):=getdual(everyroute(l))

    ! Build subproblem - remember to turn off all master prob constraints
    forall(t in TRIPS) sethidden(everytrip(t), true)
    forall(i in LEADERS) sethidden(everyroute(i), true)

    ! Solve sub-problem
    forall(l in LEADERS) do
        writeln("Solving sub-prob for ", l)
        NumVarCreated+= SolveSubProblem(l)
    end-do

    forall(t in TRIPS) sethidden(everytrip(t), false)
    forall(i in LEADERS) sethidden(everyroute(i), false)
    write(" ", NumVarCreated)
    if NumVarCreated = 0 then
        break
    end-if

    NumVarCreated:= 0
    loadprob(MasterObj)
    loadbasis(1)
end-do
end-procedure

function SolveSubProblem(l:string):integer
declarations
    NewRoute: integer
end-declarations
!

Constraints
forall(j in TRIPS) do
    TripTypeConstraint1(j):=
        Traintrips(TripType(j))>=sum(i in TRIPS)Assign_TripPair(i,j)
end-do

forall(i in TRIPS) do
    TripTypeConstraint2(i):=
        Traintrips(TripType(i))>=sum(j in ALLNODES)Assign_TripPair(i,j)
end-do

onetrip:= sum(i in TRIPS)Assign_TripPair(i,ENDNODE)=1

forall(i in TRIPS) do
    nomore(i):= sum(j in ALLNODES)Assign_TripPair(i,j)<=1
end-do

forall(j in TRIPS) do
    conser(j):= sum(i in TRIPS) Assign_TripPair(i,j)<=sum(k in ALLNODES)Assign_TripPair(j,k)
end-do

SubProbObj:= sum(t in TRIP_TYPES | Qualification(l,t)=0) TrainingCost(t)*Traintrips(t) +
sum(i in TRIPS, j in TRIPS) Assign_TripPair(i,j)*
Transfer_Cost(End_City(TripType(i)), Start_City(TripType(j))) -
sum(t in TRIPS) (dualeverytrip(t)*(sum(k in ALLNODES)Assign_TripPair(t,k)))

!minimise SubProbObj
minimise(SubProbObj)

!retrieve route and update ROUTES and RouteTrips and route pairs
!writeln("sub prob obj: ", getobjval)
reducedcost:= getobjval+dualeveryroute(1) + Start_Cost
!writeln("reducedcost: ", reducedcost)

if (reducedcost<-EPS) then
    ROUTES+= getsize(ROUTES)+1
    NewRoute:= getsize(ROUTES)
!getsizen(ROUTES) is now the current route index
!updates the schedule and first trip
forall(j in TRIPS) do
    if getsol(sum(i in TRIPS | exists(Assign_TripPair(i,j))) Assign_TripPair(i,j))<=
ZERO_TOL and getsol(sum(k in ALLNODES | exists(Assign_TripPair(j,k))) Assign_TripPair(j,k)) > ZERO_TOL then

!First Trip Set for this route
FirstTripSet:= FirstTripSet + j
schedule(getsize(ROUTES),j):=1

!If the first trip is done then break can be no more first trips
break
end-if
end-do

!Updates the schedule prop, and route trips
forall(j in TRIPS) do
  if(schedule(getsize(ROUTES),j)=1) then
    MyNextTrip:= NextTrip(j)
    Schedule_Prop(getsize(ROUTES),TripType(j)):= 1
    RouteTrips(getsize(ROUTES),j):=1
    prev:=j
    count11:=0
    while(MyNextTrip <> ENDNODE) do
      Schedule_Prop(getsize(ROUTES),TripType(MyNextTrip)):= 1
      RouteTrips(getsize(ROUTES),MyNextTrip):=1
      Pairsdone(getsize(ROUTES),prev,MyNextTrip):=1
      RouteTransferCost(getsize(ROUTES))+= Transfer_Cost(End_City(TripType(prev)), Start_City(TripType(MyNextTrip)))
      prev:=MyNextTrip
      MyNextTrip:= NextTrip(MyNextTrip)
      count11+=1
    end-do
    if(count11=0) then
      RouteTransferCost(getsize(ROUTES)):=0
    end-if
    TrainCost(l,getsize(ROUTES)):=0
    forall(k in TRIP_TYPES) do
      if(Qualification(l,k)=0 and Schedule_Prop(getsize(ROUTES),k)=1 ) then
        TrainCost(l,getsize(ROUTES))+=TrainingCost(k)
      end-if
    end-do
    break
  end-if
end-do
create(Assign_Leaders(l,NewRoute))
Assign_Leaders(l,NewRoute) is_binary
MasterObj+=
(TrainCost(l,NewRoute) + RouteTransferCost(NewRoute) + Start_Cost)*
Assign_Leaders(l,NewRoute)
!add the created var to constraints
foreach(t in TRIPS | exists(RouteTrips(NewRoute,t))) do
    everytrip(t) += Assign_Leaders(l, NewRoute)
eend-do

everyroute(l) -= Assign_Leaders(l, NewRoute)
returned := 1 !if var created
else
    returned := 0
eend-if

!foreach(i in TRIPS | exists(RouteTrips(getsize(ROUTES), i))) do
!write(i," ")
eend-do
!writeln

!reset subproblem and subproblem constraints
SubProbObj := 0
foreach(j in TRIPS) do
    TripTypeConstraint1(j) := 0
eend-do
oreach(i in TRIPS) do
    TripTypeConstraint2(i) := 0
eend-do
onetrip := 0
foreach(i in TRIPS) do
    nomore(i) := 0
    conser(i) := 0
eend-do

end-function
end-model