

*The University of Melbourne,
Department of Mathematics and Statistics*

Analysis of the Military Lift Problem

David Bannister

Honours Thesis in Operations Research

Supervisors: Associate Professor Natasha Boland, Dr Heng-Soon Gan
Second Reader: Dr Sanming Zhou

November 2006.

This work was supported by the Commonwealth of Australia through the Defence Science and
Technology Organisation

Acknowledgements

I am indebted to my supervisor, Associate Professor Natasha Boland, for giving me the opportunity to be involved in such a fantastic project. Her enthusiasm for mathematics is a great inspiration, and made a potentially daunting year a very enjoyable one.

I am also grateful to Dr Heng-Soon Gan, whose patient guidance was a great help during the year.

I gratefully acknowledge the support provided by the Commonwealth of Australia through the Defence Science and Technology Organisation. Also, my sincere thanks to Dr Timothy Surendonk from the DSTO, for supporting my role in this project.

Finally, many thanks to my parents, my sister Louise, Greg Fry, Catherine Allan, and Michelle Willcox. Whether it was reading my thesis or giving me love and support, you have made this year possible.

Contents

1	Review of the literature	10
2	Review of the techniques	14
2.1	Integer programming	14
2.2	Column generation	17
2.3	Eulerian paths	20
3	The Aggregated Lift Model	22
3.1	Validity of the ALM	26
4	Improvement of the model	31
4.1	Redundant constraints	31
4.2	Why are some models hard to solve?	39
4.3	Nuances of the simulation	39
5	Symmetry and tails	42
6	Column generation	44
6.1	Aggregate Lift Column Generation	44
7	Conclusion	47
A	Data sets	48
B	Redundant constraint comparisons	51
C	Symmetry proof	57
D	Errata	58

Introduction

Understanding the problem

One of the most crucial elements in formulating any plan or strategy in any environment is the efficiency and speed with which the steps and elements of the plan can be placed into action. A share trader needs to have equity readily available and be able to buy or sell shares quickly in order to realise the potential profit to be made. A company wishing to launch an advertising campaign would prefer to have the campaign running as soon as possible to make sure the style of the advertisement still appeals to the target demographic. In either of these two examples, a slow implementation will yield a penalty of decreased revenue. A much more serious case, however, is where a slow implementation could result in a loss of life. Such a case is the topic of this thesis.

Disaster relief operations, peacekeeping duties, and the movement of military troops into hostile situations are examples whereby the speed of execution becomes literally a matter of life and death. How quickly crucial elements (such as personnel, medical supplies, and vehicles), arrive at the location has a very real impact on the potential to save the lives of disaster victims, or maintain (or retain) control of a trouble-spot. In a generalisation of any problem of this gravity, the goal is to get personnel and cargo from Point A to Point B as quickly as possible.

The general transportation problem is well-known in the combinatorial optimisation community. It may take any of a number of possible forms, such as a parcel delivery network[2][8], the flow of data through a network, or the movement of passengers and cargo as is examined here. Despite the varying facades, the underlying construct is the same and most transport problems are extensions of the *Minimum Cost Network Flow problem*(MCNF) [17].

Consider a directed graph, $G=(V,E)$, and along each arc $e \in E$ a decision x has to be made, such that x units (eg. cargo, information) of a single commodity flow along the arc. Every arc e has a nonnegative capacity u_e , which implies that for two connected vertices, v and w , the limit of the flow between them is u_{vw} . Each arc also has a cost per unit flow, c_{vw} , which may be monetary or temporal, and thus the cost of pushing flow along vw is $c_{vw}x_{vw}$.

Every node $v \in V$ there is a requirement on the *net flow* into v , b_v . If b_v is positive, then v requires b_v units of flow and is a *sink*. If b_v is negative, then v requires to provide flow, and is a *source*. Finally, if b_v is zero, v can neither accept nor provide an excess of flow, and can be

thought of as an interim location through which flow can move but must not stop. A network of this type is shown in Figure 1.

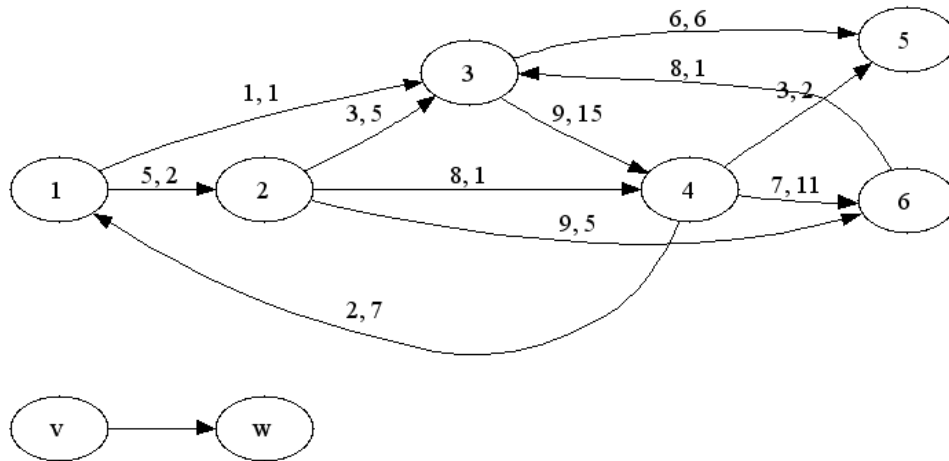


Figure 1: A graphical example of the generalised Minimum Cost Network Flow problem.

Therefore, the MCNF problem can be modelled

$$\begin{aligned} & \min \sum_{vw \in E} c_{vw} x_{vw} \\ & \text{subject to } \sum_{wv \in E} x_{wv} - \sum_{vw \in E} x_{vw} = b_v, \text{ for all } v \in V \\ & 0 \leq x_{vw} \leq u_{vw}, \text{ for all } vw \in E \end{aligned}$$

This generalisation can be extended in order to better model the situations arising in the real world. Instead of Point A to Point B, it is more likely that the objective will be to move personnel and cargo from multiple sources to (perhaps) multiple destinations. This is, in part, due to the fact that for operations on a large scale the required personnel and cargo are unlikely to be housed at the same facility for practical reasons. Furthermore, the reason for the operation (natural disaster or military conflict, for example) is unlikely to be confined to a single location and would possibly affect several towns or regions, perhaps even several countries.

Making the situation even more complicated, there will nearly always be multiple methods by which the personnel or cargo can travel to the required locations, namely different vehicles that can be used to transport one or the other, or both. The different vehicles may have different speeds and capacities, and perhaps a host of other constraints, for example a ship cannot deliver anything to a landlocked location. This means that the mode of transport becomes an important consideration when planning an operation of this type.

In this thesis, the problem being dealt with is focussed on the movement of troops and cargo from various locations (specifically cities) around Australia to several locations overseas (localised to one country) called *the Theatre* via a series of connecting legs, and these locations and legs are referred to as the *network*. There are multiple vessel types in two categories, sea and air, all with different capacities, capabilities, and speeds. There may be several vessels of the same type in the scenario, in which case they will be assumed to be identical. There is a full list of cargo demands for each location in Australia to each location in the Theatre. This provides the ability to see what is required at any location, and from where it is required.

There is a further specification on the consideration of the problem governing the way in which vessels (and anything they carry) can leave Australia and enter the Theatre. The conditions are that all vessels must leave Australia through a particular city, termed the *Point of Embarkation (POE)*, and similarly must enter the Theatre through a particular location, termed the *Point of Debarkation (POD)*. This in turn creates a unique (and important) leg in the network which is the only connection between Australia and the Theatre, called the *Bridge*. This layout is shown in Figure 2, and the special structure of this network has been named the *two-star topology*, as the two “stars” are visible as Australia and the Theatre.

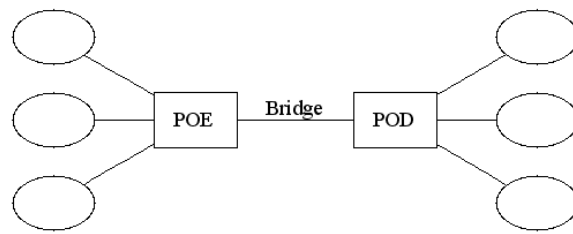


Figure 2: The basic layout of the network, showing the “two-star topology”.

The time at which all demand is satisfied is called the closure time. Naturally, the objective is to minimise this closure time for any possible scenario and circumstances that may arise. This can hopefully be achieved using operations research techniques, elements of graph theory, and software-based solvers. In fact this goal can be achieved - programs written for the Defence Science and Technology Organisation by Tim Surendonk and Natasha Boland solve exactly this problem, and large portions of this thesis stem from the use and modification of these programs. This thesis also provides an examination of the theory and reasoning behind the mathematical background that accompanies, and allows us to find solutions to, this important family of problems.

Chapter 1

Review of the literature

In light of the potential seriousness of this problem outlined in the introduction it is perhaps no wonder that there has been a substantial amount of work done in this area. Indeed, other national governments have previously commissioned research in the optimisation of troop movements and disaster relief planning. Problems of this type are oft-denoted *Multi-commodity, multi-modal network flow* problems (which may also be reduced to single-commodity multi-modal or multi-commodity single-mode network flow problems for simplification). Variations to the basic formula can be extended, without a great deal of imagination, to areas outside that of governmental use. For example, the goal stated in the introduction, “to get personnel and cargo from Point A to Point B as quickly as possible”, could be the company motto for a parcel delivery service, “*Our goal is get your package from Point A to B as quickly as possible*”. Diverse applications such as this further illustrate the interest in problems of this type in the academic and business community. Therefore it is relatively simple, and indeed a very practical idea, to ascertain the state-of-the-art in this field.

One of the most common, and necessary features when solving these types of problems is the inclusion of time into the model. That is, associating a time component when discussing a schedule of a mode and its cargo so as to obtain an evolving viewpoint of the location and movements of the objects in the network.

Some of the most intriguing formulations of the problem arise when dealing with disaster relief management. Not only are the usual features and complexities of the problem apparent, but there is also the possibility of legs and modes (in particular, roads and specific vehicles) becoming unusable over time due to aspects of the disaster. Formulations of this type are worthy of consideration because it is entirely possible that in a military conflict situation the use of legs may be compromised by enemy activities.

The paper by Haghani and Oh [7] examines a disaster relief problem with the aforementioned possibility of legs becoming unusable. Not surprisingly, their model is very complex. It takes into account features such as commodities becoming available for shipment over time, and individual costs for moving various types of commodity per mode at each time period. The goal is thus

to minimise the cost of both commodity flow and vehicular flow, whilst satisfying the various demands and constraints.

The most definitive feature described by Haghani and Oh is the introduction of a *time-space network*. The basic premise is as follows: Time is discretised into periods, and the discretisation is based on the time it takes for vehicles to traverse legs. In this way, it is easy to see the cargo flow per time period by mode. This representation also allows easy visualisation of the times and locations that cargo changes mode. This is a useful idea for aiding to create a feasible schedule, as well as deriving information from the solution. Although the stochastic nature of the problem is not initially considered here, the time-space network is used and code is written that generates a (text-only) time-space network for the problem in this thesis.

Another source of inspiration is the 1999 paper by Kim *et al.* [8] which is concerned with package delivery. As mentioned in the introduction, the formulation of a package delivery network problem is an analogue of the military lift problem. In [cite] the network has packages originating from various locations and destined for their respective recipients, and there are fleets of ground vehicles and aircraft serving as the modes of transport. There are also *service windows*, whereby certain packages must arrive at their destination before a specified time. There is also a time-space network, much the same as the one described in [7].

The key perception in [8] is that the authors address the problem of having a poorly bounded linear program (LP). One of the manners in which they strengthen their bounds is by the introduction of valid inequalities into their model. This is achieved, in part, by exploiting the structure of the network. This is relevant to the network in this thesis, and a full exposition of this idea follows in Chapter 4.

A follow up to [8] was published in 2002 by Barnhart *et al.* [2], and again deals with package delivery. Here, a route-generation subproblem is used to provide routes as needed in order to minimise the main objective, minimising the cost of shipment. This route-generation is potentially useful for developing a column generation model for this thesis, discussed in Chapter 6.

The main difference between the package delivery problem and the military lift problem is that package delivery is ongoing and well-defined, whereas military lift comprises many unknown quantities and is of limited duration. Despite this difference, there are many aspects which are analogous to the military lift problem and there is benefit to examining the literature.

Baker *et al.* [1] published a very technical paper in which they examine a military *airlift* problem based largely on the 1991 Persian Gulf war. Comprising fleets made exclusively of aircraft, the model is none-the-less complex and detailed. The model takes into account such real-world criteria as fuel capacities of airfields, airfield servicing and throughput constraints, and maximum aircraft-hours constraints. The “minute” details of assigning crews to aircraft is also addressed.

One of the features discussed [1] is the fact that modelling problems of this type can enable a decision to be made on the purchase of aircraft. The United States Air Force used a model, ‘*THRUPUT II*’, to decide between two different aircraft as their new airlifter. This is a useful

side application of these models, as they can be modified and used to compare multiple scenarios under the different hypotheses of purchasing different fleets.

The network structure described in the 2002 paper by Morton *et al.* [11], is exclusively concerned with the movement of sea vessels. The network described has only seaports, but all seaports in the home country are seen as POE's, and similarly both of the two ports in the Theatre are seen as POD's. Therefore all vessels are free to move from any home port to any Theatre port. More than this, there is an additional element centred around the possibility of attack on a POD. In their description of the model, this would cause a POD to be rendered inactive for a set period of time, and then gradually increase its throughput capacity until it is again fully operational.

The Morton *et al.* model takes a stochastic approach to modelling attacks on PODs. They model possible outcomes, such as one POD or the other being attacked but not both. They base the probability of attack on distributions which are estimated representations of, and in practice would be represented by, intelligence reports on the planning of the enemy. Their distributions take into account that attacks may occur earlier or later in the scenario, based on the enemy's ability to mobilise weaponry.

The Morton *et al.* model then seeks to deliver cargo across at a rate that varies based on the attack probability distribution in such a way that the delay of cargo reaching its destination is minimised. Their model finds that "...hedging against a possible attack can provide substantial benefits if an attack occurs, and incurs only minor penalties if not." Although there are aspects of their model that are simplified (all airlift being ignored, for example), the Morton *et al.* model provides a good basis to perhaps later include stochastic scenarios into the model examined in this paper.

The most profound paper reviewed here, in the opinion of this author, is the 2004 paper by Nielsen *et al.* [12] on the so-called *channel route* scheduling problem. The channel route network is component of the US military, and serves to move cargo and personnel to-and-from various locations during peacetime. This structure is considered recurring as the schedule is generated every month.

The model is first constructed in a 'traditional' sense, as a mixed integer programming (MIP) problem. The difficulties in finding integer solutions are discussed, stating that cargo flow variables, vessel route variables, and flow balance constraints all cause fractionality in the optimal LP solution. In addition, the LP-relaxation is weak and thus provides a very poor lower bound. Due to these factors, finding the optimal integer solution requires considerable effort in solving the branch-and-bound tree.

The remainder of the paper is devoted to a '*composite variable formulation*'. The notion of composite variable formulation differs from ordinary reformulations in linear programming. For example, it might be common to combine several trip variables into a single 'path' variable akin to the patterns of column generation. Composite variable formulation combines two or more *types* of decision variables into a single variable. For example, in [12] composite variables are created out of a combination of cargo flow variables and route variables. It is noted that using

composite variables gives a tighter LP-relaxation (closer to the MIP solution), and has shorter solve times.

Composite variables demonstrate the interdependence of certain types of variables in the network formulation. This notion can be used, in part, when considering the formulation of redundant constraints as in Chapter 4.

Chapter 2

Review of the techniques

This chapter provides a review of the mathematical techniques related to the modelling and solving of the problem.

2.1 Integer programming

The development of the simplex method was an important step in solving optimisation problems. It is a powerful method by which a linear programming problem can be solved quickly and to optimality. However, the entire basis of the simplex method is that variables are allowed to (and often will) take non-integer values. This was an immediate problem in the case where integral values were required as it doesn't make sense, for example, to assign 3.41 workers to complete a task. Nor is it a simple case of rounding variables to integer points, as the rounded optimal LP solution may be much worse than the optimal integer solution.

In 1960, Land and Doig developed the method of Branch-and-Bound to extend linear programming to integer programming [9]. In the words of the authors, "The procedure...could be described as 'pushing down the functional line until it meets an integral point.'" More explicitly, they were referring to a maximisation problem, and the manner of pushing down the function line is through the polytopic, n-dimensional subspace generated by the LP relaxation of the integer program (IP).

The Branch-and-Bound algorithm is as follows. Consider a problem of the form

$$\begin{aligned} \max z &= \mathbf{c}^T \mathbf{x} \\ \text{s.t.} & \\ & S \\ & x_i \in \mathbb{Z}^+ \end{aligned} \tag{2.1}$$

where \mathbf{c} is a vector of scalars, x is a vector of decision variables, and S is a set of constraints invoked by the model.

Step 1: Solve the model with the exclusion of the integrality constraint (2.1): the LP-relaxation of the IP. If the optimal solution to the LP-relaxation has each x_i integer, then the algorithm can stop as this is also the optimal solution to the IP. Otherwise, proceed to step two.

Step 2: For some non-integral value x_j , create two subproblems based on the current value of x_j , \bar{x}_j . Let subproblem one be the LP-relaxation, with the additional constraint

$$x_j \leq \lfloor \bar{x}_j \rfloor$$

By analogy, let subproblem two be the LP-relaxation with the additional constraint

$$x_j \geq \lceil \bar{x}_j \rceil$$

In a sense this causes a partition to the feasible region.

At each one of these subproblems we check for feasibility. If there are infeasible constraints or the problem becomes infeasible, we denote that node as infeasible, and no further action is taken there.

Step 3: Consider solving the LP subproblem one, with constraint set

$$S_1 = S \cup x_j \leq \lfloor \bar{x}_j \rfloor$$

If the solution here is not integral, branch into a further two subproblems analogous to Step 2.

If the new subproblem is feasible, the optimal solution is found and the values noted. If the solution is integer, then it is a candidate solution to the IP and the value is noted.

Step 4: Solve each subproblem in this manner until a stopping rule applies.

At any point in time, the best integer solution is used to cut-down the number of subproblems examined. If there is ever a solution, integer or otherwise, that is worse than the best integer solution, we ‘prune’ that node and do nothing more to it. This is because by adding constraints we cut down the feasible region, and therefore the new solution can only be as good as or worse than the solution without the new constraint. In other words, if S is the polytope described by constraint set S , and S_1 the polytope described by set S_1 , it must be that

$$S_1 \subseteq S$$

Step 5: The algorithm continues in this manner until all nodes are infeasible, pruned, or integer solutions. The best integer solution at this stage, if one exists, is the optimal integer programming solution.

Figure 2.1 gives an example of branch-and-bound.

$$\begin{aligned}
& \text{Max } z = x_1 + 2x_2 \\
& \text{s.t. } -2x_1 + 7x_2 \leq 14 \\
& \quad 6x_1 + 2x_2 \leq 27 \\
& \quad x_1, x_2 \in \mathbb{Z}_+
\end{aligned}$$

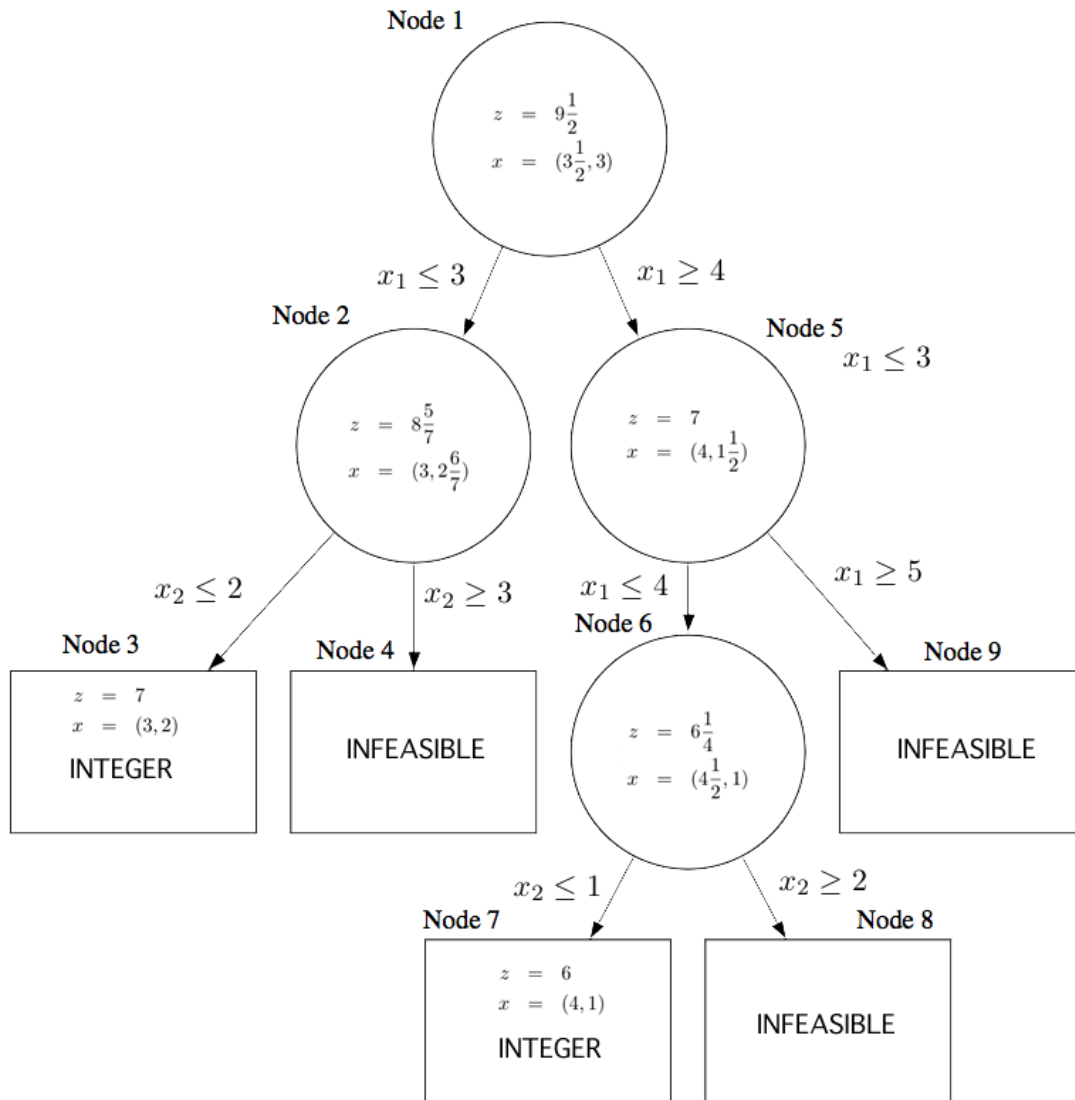


Figure 2.1: An example of branch-and-bound taken from [4], with the optimal solution found at node 3. For illustrative purposes, there has been no pruning. However it should be noted that node 3 creates a lower bound of 7, and thus at node 6 we could prune node 6 by node 3. In this case, nodes 7 and 8 would never be examined.

2.2 Column generation

Often the nature of modelling problems means the creation of hundreds of thousands, or millions, of variables. Solving such problems by hand would be impossible, and often the task is too much even for computers as the method of solving requires the use of memory. The idea behind column generation is to look at only feasible, and very good, solutions in order to find the optimal solution without unnecessary computation of poor solutions.

The method involves starting with an (often very) restricted problem matrix (the set of columns for this reduced matrix is much, much smaller than for the original problem). By solving this set of columns, the algorithm will add one or more columns to the set, such that each added column 'prices out' favourably, and thus improves the current solution. A column prices out favourably if, for a minimisation problem, it has a negative reduced cost based on the dual value of the current solution.

Consider an LP of the standard form

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & A^T \mathbf{x} \geq \mathbf{b} \quad (\pi) \\ & \mathbf{x} \geq 0 \end{aligned}$$

Let \mathbf{x} be a basic feasible point for the basis, \mathcal{B} . Then B is the submatrix of A , determined by \mathcal{B} . By duality theory, we have dual variables

$$\pi = \mathbf{c}_{\mathcal{B}} B^{-1}$$

and that \mathbf{x} is optimal for the LP if and only if

$$\pi A \leq \mathbf{c} \tag{2.2}$$

and from Equation (2.2) we have the formula for the *reduced cost*,

$$\mathbf{c} - \pi A$$

Our original LP is called the *Master*. Often, the Master LP has A with a very large number of columns, such that solving the Master may be impossible. This intractability motivates the use of column generation.

There exists a matrix, \hat{A} , such that $\hat{A} \subset A$, and $\hat{\mathbf{c}}$ the corresponding subvector of \mathbf{c} . The *Restricted LP* is then defined by

$$\begin{aligned}
\min \quad & \hat{\mathbf{c}}^T \mathbf{y} \\
s.t. \quad & \hat{\mathbf{A}}^T \mathbf{y} \geq \mathbf{b} \\
& \mathbf{y} \geq 0
\end{aligned}$$

It is imperative to note that the Restricted LP is still subject to the constraints of the problem, and therefore $\hat{\mathbf{A}}$ must be feasible in terms of the original problem. It is now possible to solve the Master LP via the column generation algorithm.

Step 1: Choose an initial $\hat{\mathbf{A}}$. This initial set of columns may be selected arbitrarily, by some selection procedure, or by using expert knowledge of the problem.

Step 2: Solve the Restricted LP. This generates a basic solution \mathbf{y} , and the dual variables to the problem, π .

Step 3: The relationship by the variables of \mathbf{x} in the Master and \mathbf{y} in the Restricted LP is

$$x_i = y_i, \text{ if column } i \text{ is in } \hat{\mathbf{A}}$$

and zero otherwise. Therefore, the \mathbf{x} generated by \mathbf{y} is a basic feasible point for the initial problem and the Master. Furthermore, it is the optimal solution if $\pi \mathbf{A} \leq \mathbf{c}$.

Step 4: There is now a subproblem associated with the reduced cost, that is of the form

$$\begin{aligned}
\alpha &= \min c_i - \pi a_i \\
s.t. \quad & a_i \text{ is a column of } \mathbf{A}, \text{ and satisfies the constraints of the Master}
\end{aligned}$$

If the solution to this subproblem, α , is nonnegative, then \mathbf{x} is optimal for the original problem and the algorithm terminates. Otherwise, add a_i such that $c_i - \pi a_i = \alpha$ to $\hat{\mathbf{A}}$, and repeat from Step 2, using the new $\hat{\mathbf{A}}$.

The most famous example of column generation was introduced by Gilmore and Gomory in 1961, called the cutting stock problem. This problem tries to minimise the waste generated by cutting boards of different sizes from larger boards, subject to some demand for the smaller boards. There were 40 different lengths demanded, with more than 100 million ways a board could be cut. The final stage of column generation, solved by a single branch-and-bound, indicated that none of the 100 million non-basic patterns would price out favourably, without the need for explicit enumeration.

In the literature, Boland and Surendonk [5] describe a delivery planning problem which they attempt to solve by linear programming and two formulations of column generation. They find that in nine test problems the IP only solves in one case, whilst the remaining eight depleted the computer memory with large optimality gaps. Their initial column generation formulation performs only marginally better than the IP, and still exhausted system memory with optimality gaps remaining. Their second column generation formulation found optimal solutions to each data set, and in comparatively small computing time.

Hence, column generation can be used very effectively to solve large problems provided the underlying formulation is chosen correctly.

Branch and price

In the case of integer programming, it is of course required to find the optimal integer values. For this reason, when column generation is used on an IP it is necessary to complete branch-and-bound at each iteration of the column generation procedure. This hybridisation is known as the “branch-and-price” algorithm.

Dantzig-Wolfe decomposition

Often, an LP can be sectioned into independent sets. These are known as Constraint Sets and involve variables in independent Variable Sets. For example, Constraint Set 1 involves constraints that only impact on variables in Variable Set 1. In this fashion, Constraint Set k affects variables in Variable Set k and does not have an impact on variables in any other set. Any constraints that affect variables in multiple sets are put in the final Constraint Set $k + 1$, where this set is referred to as the *centralised constraints* set. When LPs can be decomposed in this manner, the Dantzig-Wolfe decomposition algorithm can be applied.

The mathematics used in the Dantzig-Wolfe decomposition follow these basic principles. The feasible region of an LP can be entirely expressed in terms of its extreme points also known as basic feasible solutions [16]. This is the same principle one uses to define any point on a line - by expressing it solely in terms of two distinct points on the line. Assuming the extreme points of an LP are $\mathbf{P}_1, \mathbf{P}_2, \dots, \mathbf{P}_k$, each point in the feasible region can be expressed as

$$\mathbf{x} = \mu_1\mathbf{P}_1 + \mu_2\mathbf{P}_2 + \dots + \mu_k\mathbf{P}_k$$

where

$$\mu_1 + \mu_2 + \dots + \mu_k = 1$$

and

$$\mu_i \geq 0 \quad \text{for } i = 1, 2, \dots, k.$$

The Dantzig-Wolfe decomposition uses these equations iteratively to form a *restricted master*. For a case with two variable sets, the algorithm proceeds as follows:

1. Let variables in Variable Set 1 be x_1, x_2, \dots, x_s . Write the variables in terms of the extreme points of the feasible region given by Constraint Set 1, $\mathbf{P}_1, \mathbf{P}_2, \dots, \mathbf{P}_k$. Thus

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_s \end{bmatrix} = \mu_1\mathbf{P}_1 + \mu_2\mathbf{P}_2 + \dots + \mu_k\mathbf{P}_k$$

where $\mu_1 + \mu_2 + \dots + \mu_k = 1$ and $\mu_i \geq 0$ ($i = 1, 2, \dots, k$).

2. Let variables in Variable Set 2 be $x_{s+1}, x_{s+2}, \dots, x_n$. Write the variables in terms of the extreme points of the feasible region given by Constraint Set 2, $\mathbf{Q}_1, \mathbf{Q}_2, \dots, \mathbf{Q}_m$. Thus

$$\begin{bmatrix} x_s \\ x_{s+1} \\ \vdots \\ x_n \end{bmatrix} = \lambda_1 \mathbf{Q}_1 + \lambda_2 \mathbf{Q}_2 + \dots + \lambda_m \mathbf{Q}_m$$

where $\lambda_1 + \lambda_2 + \dots + \lambda_m = 1$ and $\lambda_i \geq 0$ ($i = 1, 2, \dots, m$).

3. Using the equations found in the first two steps, express the LP's objective function and centralised constraints (from Constraint Set $k + 1$) in terms of μ_i 's and λ_i 's. Collating the information, the restricted master is:

$$\begin{array}{ll} \text{max (or min)} & \text{[objective function in terms of } \mu_i \text{'s, } \lambda_i \text{'s]} \\ \text{s.t.} & \text{[centralised constraints in terms of } \mu_i \text{'s, } \lambda_i \text{'s]} \\ & \mu_1 + \mu_2 + \dots + \mu_k = 1 \\ & \lambda_1 + \lambda_2 + \dots + \lambda_m = 1 \\ & \mu_i \geq 0 (i = 1, 2, \dots, k) \\ & \lambda_i \geq 0 (i = 1, 2, \dots, m). \end{array}$$

4. Assuming a basic feasible solution is readily available, use column generation to solve the restricted master.
5. Using the optimal values found for the μ_i 's and λ_i 's, substitute the values into the equations found in steps 1 and 2 to find the optimal solutions for the variables x_1, x_2, \dots, x_n .

One advantage of decomposition is its reduction in memory consumption. The process removes the initial matrix, which would be comprised largely of zeroes anyway, and replaces it with a set of smaller matrices.

The notions of column generation, branch-and-price, and decomposition may be used for a column generation model discussed in Chapter 6.

2.3 Eulerian paths

A path on a graph, $G=(V,E)$, is Eulerian if it visits each edge $e \in E$ exactly once. An Eulerian cycle is a cycle that uses each edge exactly once. A graph is *traversable* if it allows Eulerian paths, and *Eulerian* or *unicursal* if it contains an Eulerian cycle [15]. These special cases are named after Leonhard Euler, who used these notions to solve (and prove impossible) the famous *Bridges of Königsberg* problem, shown in Figure 2.2. In this problem the aim was to walk the seven bridges of Königsberg and return to the starting point, thus walking an Eulerian cycle. Although the story behind this example related to accessing a tavern, the idea of Eulerian paths and cycles is very useful in our problem.

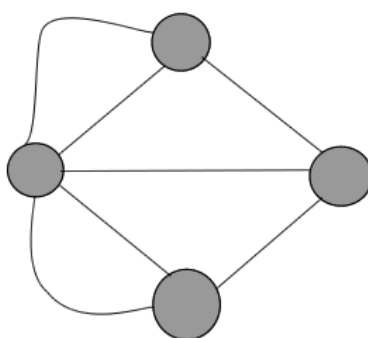


Figure 2.2: Bridges of Königsberg, not a Eulerian Graph. Here, the nodes represent islands and the arcs represent (undirected) bridges.

In graph theory, the ‘degree’ of a node v in an undirected graph is the number of edges incident with v . An undirected graph, $G=(V,E)$, allows for the construction of an Eulerian path iff it is *connected* and there are *at most* two nodes with odd degree. From this it is easily noted that Figure 2.2 cannot contain an Eulerian path as each and every node has odd degree. An undirected, connected graph is unicursal iff every node has even degree. However, the graphs considered for this thesis contain directed arcs, therefore it must be noted that a directed graph is unicursal iff every node has equal ‘in-degree’ and ‘out-degree’. That is, for a directed graph $G=(V,E)$ and a node v , require

$$\text{arcs into } v = \text{arcs out of } v, \quad \text{for all } v \in V$$

in order for G to be unicursal.

As the problem of transport requires the use of a mode of transportation, ie a vessel, there is the added subproblem of maintaining a valid modal flow. Put simply, a vessel can only perform a trip on a leg where the start of that leg is the current location of a vessel. This is similar to the problem which occurs when solving the Travelling Salesman Problem, the creation of subtours means an unrealistic solution. For this reason, we can examine the modal-flow as a version of an Eulerian path. This notion is fully explained in Chapter 3.

Chapter 3

The Aggregated Lift Model

The model primarily examined in this thesis is the Aggregated Lift Model (ALM). This model attempts to minimise the closure time over the network, with a specified fleet and demand schedule. From here on in, the network will be the *two-star network* outlined in Figure 2. The most important feature about the Aggregated Lift Model is to note that its name is not a misnomer. The ALM doesn't take into account a realistic schedule of vessels, rather it calculates how long it would take for a fleet to move as much cargo as there is defined in the model, i.e. it doesn't matter if the cargo is not at a location waiting to be picked up, the vessel will act as though there is as much cargo at the port as it needs. The most accurate way of thinking about this is to realise that there is no time component, in terms of time elapsed since the beginning of the model.

The other important aspects of the ALM are:

- Most of the parameters of the model are user-defined within a basic framework
- There are *tails* that exist as multiple copies of a class of vessel

This second point is a major distinction between the ALM and its predecessor. Previous formulations sought to combine the tails of a vessel class into a 'super-vessel'. These formulations have several drawbacks, one of the most important being a worse solution as a result of the amalgamation. Further discussion on this idea is given later.

The ALM is described by the following MIP:

$$\begin{aligned}
& \min \quad \tau & (3.1) \\
& \text{subject to} \\
& \sum_{l \in L_v} x_{vt}^l c_v^l \leq \tau, \forall v \in V, t \in N_v & (3.2) \\
& \sum_{l \in L^p} x_{vt}^l + b_{vt}^p \geq \sum_{l \in L_p} x_{vt}^l, \forall v \in V, t \in N_v, p \in \mathcal{P} & (3.3) \\
& \sum_{o \in \mathcal{O}} \sum_{d \in \mathcal{D}} \sum_{t=1}^{N_v} q_{vtl,o}^d \leq \sum_{t=1}^{N_v} C_v^l x_{vt}^l, \forall v \in V, l \in L & (3.4) \\
& \sum_{v \in V} \sum_{l \in L^p} \sum_{t=1}^{N_v} q_{vtl,o}^d - \sum_{v \in V} \sum_{t=1}^{N_v} \sum_{l \in L_p} q_{vtl,o}^d = F_{o,p}^d, \forall o \in \mathcal{O}, d \in \mathcal{D}, p \in \mathcal{P} & (3.5) \\
& \left\lceil \frac{\sum_{o \in \mathcal{O}} D_o^d}{C_v^{l'}} \right\rceil \times x_{vt}^{l^*} \geq x_{vt}^{l'}, \quad \forall v \in V, t \in N_v, l' \in L_{POD}^d, d \in \mathcal{D} & (3.6) \\
& x_{vt}^l \in \mathbb{Z}^+, q_{vtl,o}^d \geq 0
\end{aligned}$$

where the deterministic data is given by

- v is a vessel class, from a set of classes V
- N_v is the number of tails of type v
- L is the set of legs in the scenario
- L_x^y indicates the unique set of legs, i.e. path, from x to y , $L_x^y \subseteq L$
- \mathcal{L}_v is the set of legs that vessel class v can travel on
- l_p^q is a leg starting at p and ending at q
- l^* is the bridge leg from the POE to the POD
- b_{vt}^p is a binary parameter true iff tail t of vessel v is based at location $p \in \mathcal{O}$
- \mathcal{O} is the set of Australian locations
- \mathcal{D} is the set of Theatre locations
- $\mathcal{P} = \mathcal{O} \cup \mathcal{D}$ is the set of all locations
- C_v^l is the capacity, in tons, of a vessel of class v on leg l
- c_v^l is the time it takes a vessel of class v to traverse leg l
- D_o^d is the total demand from o to d

$$F_{o,p}^d = \begin{cases} -D_o^d, & p = 0 \\ D_o^d, & p = d \\ 0, & \text{otherwise} \end{cases} \quad (3.7)$$

and the decision variables are

- τ is the time at which all demand has been satisfied, the closure time
- x_{vt}^l is the number of trips that tail t of class v does on leg l
- $q_{vtl,o}^d$ is the amount carried along leg l by tail t in vessel class v , originating from o and destined for d

which must all be nonnegative (and integer in the case of the number of trips performed).

The objective function, equation (3.1), seeks to minimise the total amount of time it takes to complete all trips. As trips are completed by different vessels simultaneously, the objective function is not of a normal linear form. It is easy to see that the time at which all demand is satisfied must be the time at which the final trip is performed. Therefore the objective function is best thought of as a combination of (3.1) and (3.2). The relationship is, effectively,

$$\tau = \max \left\{ \sum_{l \in L_v} x_{vt}^l c_v^l \right\}, \forall v \in V, t \in N_v$$

such that τ is the maximum value in the set of vessel finishing times. Hence, minimising τ also serves to minimise the maximal element of vessel finishing times, subject to the other constraints.

Equation (3.3) maintains a realistic vessel flow. It restricts the number of trips out of a location, p , by a vessel to be less than or equal to the trips into p by that vessel. In other words, a vessel cannot leave a location unless it has entered that location¹. The exception to this rule is the location where the vessel is based, as this is the starting location for the vessel. A more detailed analysis of this constraint follows later.

Constraint (3.4) simply states that the amount of cargo carried over a leg l by a vessel of class v must be less than the maximum cargo capacity of class v over leg l . This varies by leg because different vessel capacities can vary along different legs. An example of this, the further an aeroplane has to fly without refuelling, the less cargo it can carry.

Equation (3.5) is analogous to the value b_v from the MCNF outlined in the introduction. However, whereas the b_v dealt with a homogeneous commodity in the MCNF, here the commodities are defined by their origin and destination. Therefore each location has a set of F values, corresponding to the full set of demand pairings. An F value of zero indicates that the cargo must not stop there, whilst a positive(negative) F value indicates the location is the destination(origin) of F tons of cargo.

Finally, (3.6) is a constraint that enforces subtour elimination. Recall that (3.3) requires a valid vessel flow in terms of each individual location, but does not stop a vessel performing trips on

¹It should be noted that in a solution of this nature, this flow only has to be maintained in the final solution. The trips do not have to be made in order, as the whole point of the ALM is provide a solution that *can* be transformed into a proper schedule.

both sides of the bridge without actually crossing into the Theatre. Constraint (3.6) prevents this by stating that a vessel can only do trips in the Theatre if it crosses the bridge. It is, in effect, a variation on a theme of a “Big-M” constraint, often seen in linear programming. When used in conjunction with (3.3), a valid flow should be maintained, and this is examined in greater detail later in this chapter. NB: this assumes that all vessels are located in Australia, and by the formulation of the model, all vessels are located in Australia.

There is a second optimisation, in the DSTO program, on the total number of trips. It is intuitive to think that it would be possible to have several different constructions of trips that produce the same closure time, especially on large-scale scenarios. For this reason the status of the solution is noted, in terms of the closure time and the total number of trips. The next optimisation is to then find a total number of trips that is less than or equal to the number found in the first part, and that the closure time remains the same as in the first optimisation (the minimum).

It is in this way that the second procedure is not an optimisation in the true sense of the word. Rather it ‘improves’² the solution by reducing (or, in some cases, returning the same) number of trips to satisfy all demand in the minimum closure time. There is nothing to say that it finds the minimum number of trips, and by the same token there is nothing yet to say (other than common sense) that the optimal number of trips is the minimum number of trips.

In this sense, the ALM at this stage is a lexicographic minimisation problem, with objective function

$$\text{lex min} \left\{ \tau, \sum_{v \in V} \sum_{t=1}^{N_v} \sum_{l \in L} x_{vt}^l \right\}$$

From this point on, an *optimal* solution is the solution to this lexicographic problem. However, the main focus is on the closure time. As the closure time remains invariant, by the lexicographic ordering, the optimal closure time remains the same after both the first and second optimisation.

By the end of the second optimisation, the ALM provides the following data:

- The total closure time for an aggregate solution for meeting all demand
- The trips undertaken by every tail
- The finishing times of every tail
- The cargo movement, by leg and vessel, and
- The total number of trips, both the first solution and the reduced solution.

As stated from the beginning, this list of data does not define a valid *schedule*. However there are two important reasons for using this model:

²The notion of doing less trips being an improvement is subject to the conditions of actually performing trips, eg a large cost associated with each trip

- The ALM provides a closure time which serves as a lower bound to the closure time for an advanced scheduling model, and
- The ALM serves as a heuristic for a scheduling model, in such a way that a proper schedule can be created from the above data.

Indeed, the next step in the program is to simulate a proper schedule from the data generated in the ALM. The DSTO program does include a simulation, and although the workings of this simulation are not the focus of this thesis, a discussion on its use is given below. In terms of providing a lower bound for an advanced scheduling model, consider the objective function of the ALM. As stated before, it minimises the total time it would take for a given fleet to move a specified amount of cargo around the network and does not take into account the movement of cargo over time. In this sense, each vessel acts as though it has as much cargo as it needs waiting at every location, so that the time it takes for a vessel to load/unload at any location is neglected. In a real network of this type, it would not be uncommon for a vessel to have to wait at a location for cargo to arrive there. Although the ALM ignores these waiting times, a real schedule cannot make this assumption. Therefore, the ALM provides a lower bound to a scheduling model because it is the special case where all waiting times are zero. Naturally, waiting times cannot be negative and thus the ALM provides the best closure time a scheduling model could ever achieve.

3.1 Validity of the ALM

As part of this thesis it was necessary to check that the ALM outputs feasible solutions. The first aspect of this was to check the output files ‘First solution’ and ‘2nd solution’, which contain the movements of every vessel after the first and second optimisations, respectively. Checking these solutions involved manual construction of the vessel movements to ascertain whether the trip patterns were feasible.

The second way in which the results were checked saw the creation of a new output file, ‘MSW solution’, which contains the information about cargo movement. It outputs the demand for each location in Australia to each location in the Theatre, how much is carried over each of the legs between the locations (so as to be sure that all demand is satisfied), and the amount carried over each leg by vessel class. This not only allows the validation of satisfying demand, but also provides insight as to how the demand is being satisfied and the role of each vessel class. There was no evidence of the ALM providing unrealistic flows or doing anything unexpected in the empirical examinations.

However, the justification of the ALM is that it can undergo postprocessing to be converted to a proper schedule. Therefore it is necessary to verify mathematically that the vessel movements defined by the constraints of the ALM allow the construction of a feasible schedule.

Definition 3.1.1. *Denote a ‘forward leg’ any leg which:*

- a) goes from any point in Australia to the POE;
- b) goes from the POE to the POD (ie the Bridge);
- c) goes from the POD to any point in the Theatre.

A ‘backward leg’ is the reverse of a forward leg.

Proposition 3.1.2. *From 3.1.1, the last leg in a vessel’s schedule in an optimal solution will be a forward leg.*

Proof. By the definition of the model and explanation in the introduction, all demand in the model is *forward*, and thus the demand over backward legs is zero. Suppose a vessel which finishes on a backward leg. As the demand over the backward leg is zero, the last leg traversed did not contribute to satisfying demand. As such, the vessel could exclude the last leg without affecting the cargo flow, thus decreasing the trip constraint objective (it may or may not affect the closure time objective) †

Given Proposition (3.1.2), it is now possible to ascertain whether the ALM generates a valid vessel flow that can be converted to a proper schedule in postprocessing.

Consider the vessel flow constraint (3.3) for a certain vessel, and the case where the number of trips on each side of the inequality are equal for all $p \in \mathcal{P}$. Therefore at every location *except* the vessel’s base, b' , the constraint holds at equality, and for b' the LHS is one greater than the RHS. Thus the vessel enters and leaves each location the same number of times. By constraint (3.6) and reason above, the vessel does not complete subtours.

Imagine that, in this case, for each trip the vessel takes, a directed arc is drawn on the network representing the leg the vessel traversed. By the equality in vessel flow, each node is entered and exited the same number of times and, by the subtour elimination, all the nodes visited by the vessel are connected. Therefore, the arcs drawn by the vessel movements according to the ALM form a directed graph, which is connected, and each node has even degree. By Section 2.3, this graph of the vessel’s movement is unicursal, and therefore the vessel must finish at its base location. Thus the vessel maps out a *traversable* (closed) walk, and therefore a proper schedule can be generated from it. A representation of this is shown in Figure 3.1.

It is clear that, in the case where the number of trips on each side of (3.3) are equal for all $p \in \mathcal{P}$, the arcs drawn by the vessel’s movement form a traversable (closed) walk. In this case, the inequality did not hold at b' with the LHS greater than the RHS by one.

Consider a single vessel based at the POE, and let x be optimal (for this discussion, the indices v and t are discarded, as only one vessel is being discussed). Then have

$$\sum_{l \in L^{POE}} x^l + 1 > \sum_{l \in L_{POE}} x^l$$

for any $q \in \mathcal{O} \setminus \{POE\}$, such that $b^q = 0$. Then by (3.3) have

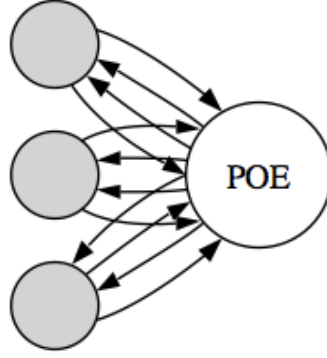


Figure 3.1: Graph of a scenario where each location has an equal number of inbound trips and outbound trips. By (3.6) this generates a connected graph, and is hence unicursal.

$$\sum_{l \in L^q} x^l \geq \sum_{l \in L_q} x^l$$

and, by the two-star structure of the network,

$$\begin{aligned} \sum_{l \in L^{POE}} x^l &= \sum_{q \in O \setminus \{POE\}} \sum_{l \in L_q^{POE}} x^l + x^{\bar{l}^*} \\ &= \sum_{q \in O \setminus \{POE\}} \sum_{l \in L_q} x^l + x^{\bar{l}^*} \\ &\leq \sum_{q \in O \setminus \{POE\}} \sum_{l \in L^q} x^l + x^{\bar{l}^*} \end{aligned} \quad (3.8)$$

where $x^{\bar{l}^*}$ is the leg from the POD to the POE. Note $L_q = L_q^{POE}$ by the network structure. Now,

$$\begin{aligned} \sum_{l \in L_{POE}} x^l &= \sum_{q \in O \setminus \{POE\}} \sum_{l \in L_{POE}^q} x^l + x^{\bar{l}^*} \\ &= \sum_{q \in O \setminus \{POE\}} \sum_{l \in L^q} x^l + x^{\bar{l}^*} \end{aligned} \quad (3.9)$$

Using this, we have

$$\begin{aligned}
\sum_{l \in L_{POE}} x^l &= \sum_{q \in O \setminus \{POE\}} \sum_{l \in L_{POE}^q} x^l + x^{\bar{l}^*} + 1 \\
&\geq \sum_{l \in L_{POE}} x^l + 1 && \text{by (3.8)} \\
&> \sum_{l \in L_{POE}} x^l && \text{(by our case definition)} \\
&= \sum_{q \in O \setminus \{POE\}} \sum_{l \in L^q} x^l + x^{l^*} && \text{by (3.9)}
\end{aligned}$$

which implies

$$x^{\bar{l}^*} + 1 > x^{l^*}$$

and due to integrality

$$x^{\bar{l}^*} \geq x^{l^*}$$

Due to the symmetry of the network, an exactly analogous case holds for $q \in \mathcal{D} \setminus \{POD\}$, giving

$$x^{\bar{l}^*} \leq x^{l^*}$$

which means

$$x^{\bar{l}^*} = x^{l^*}$$

Suppose $x^{\bar{l}^*} > 0$, and define a new solution as

$$\hat{x}^l = \begin{cases} x^l, & \text{if } l \neq \bar{l}^* \\ x^l - 1, & \text{otherwise} \end{cases}$$

Clearly, this only has an impact for (3.3) for POE and POD. Equation (3.3) is *still* valid in this case, and therefore all constraints still hold. By this fact, \hat{x} is a feasible solution, and as the number of trips of \hat{x} is less than the number of trips for x , it is a strictly better solution for the lexicographic problem, and therefore x is not optimal by contradiction. As this holds for all x , must be that

$$x^{\bar{l}^*} = x^{l^*} = 0$$

and thus the trips shown in Figure 3.1 is proved correct.

For the situation where the inequality *does* hold for the base location must be considered. For this to occur, the vessel must not return to b' : the number of trips on the RHS of (3.3) is one greater than the number of trips on the LHS. Thus, the vessel must finish at a location other than b' , and so for some location $p' \in \mathcal{P}$, the inequality must *not* hold: the number of trips into p' is one greater than the number of trips out of p' . Using arguments similar to the first case, every other location, i.e., with the exception of b' and p' , must have the same number of trips in as trips out. As such, all but two of the locations have even degree whilst b' and p' have odd degree. By Section 2.3, this graph is *not* unicursal.

Consider now the inclusion of an ‘imaginary’ leg connecting, and in the direction of, p' to b' . The vessel does not actually travel on this leg, but it plays an important role. As stated above, the number of arcs out of b' is one greater than the number of arcs into b' , and for p' the number of inbound arcs is greater than the outbound arcs by one. Therefore, the inclusion of the imaginary leg, $p'b'$, means that for both locations the number of arcs in equals the number of arcs out. This idea is shown in Figure 3.2.

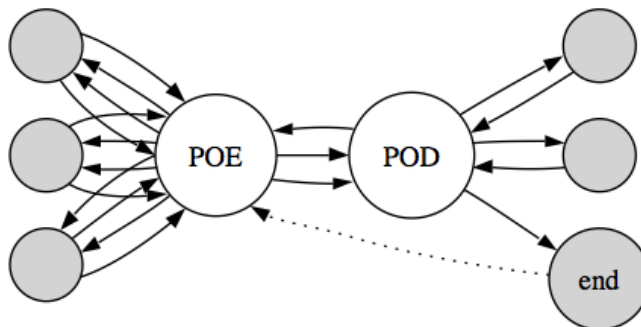


Figure 3.2: Graph of trips, vessel starts at POE and finishes at ‘end’, with the imaginary arc show as a dashed line.

By the inclusion of this arc, as is seen in Figure 3.2, each location has even degree and, therefore, the graph is once again unicursal. Hence, a closed walk can be made from b' that covers every arc and returns to b' . For any closed walk, $v w_1, w_1 w_2, \dots, w_{n-1} w_n, w_n v$, it is true that there exists an ‘open’ walk $v w_1, w_1 w_2, \dots, w_{n-1} w_n$, i.e., an open walk can be formed by removing the last arc in the closed walk. For the vessel trip graph, as mentioned before, we can denote the last arc in the closed walk to be the imaginary arc $p'b'$ ³. Therefore, removing the imaginary arc removes the closed walk and forms a directed open walk from b' to p' that visits every edge in the graph. This implies that the graph of vessel trips is *traversable* for a vessel starting at b' and ending at p' . Therefore, by the notion that a traversable graph of trips can be made into a schedule, the ALM provides a valid vessel flow.

³The definition for a graph to be unicursal means that any leg entering the starting location can trivially be called the last arc of a closed walk

Chapter 4

Improvement of the model

4.1 Redundant constraints

It is now apparent that the scope of this problem is almost limitless, and from a modelling aspect there may be millions of variables and constraints. In some cases, the solution space may be unworkably large and an optimal (or even feasible) solution may not be found in a reasonable amount of time, or within the capacity of the computer solving the problem.

This relates to the manner in which the solver finds solutions. The branch-and-bound method iteratively updates two values, the best solution found and the best bound. The best solution is the best integer solution found at some point, and the best bound is a value based on the LP solution and the b-b tree, a value which it has been determined the solution must be bounded by (ie a value that, if minimising, the objective function can never be below). The optimal solution is thus determined when the gap between the best bound and the best solution is zero. If the initial bound and/or solution are far from optimal then the solver may spend a lot of time (and b-b tree calculations) in attempting to close the gap between these values¹. Taking this into account, the ideal situation would have the upper and lower bounds relatively close, and the gap narrow quickly in terms of the branch-and-bound iterations completed. This can be achieved in some cases by performing a cut on the polytope in which the solution is found.

One manner in which this can be achieved is the introduction of so-called *redundant constraints*. In general, these are simple, obvious (in fact may be the verbal objective of the problem stated mathematically), and are redundant because they are already satisfied by the more complicated constraints in the program. Although the idea may be counter-intuitive, and seemingly add unnecessary lines of code to the program and potentially slow the solver down, in practice it can actually have a dramatic positive effect on the solving time.

This particular model lends itself readily the creation of redundant constraints by its unique structure. Due the presence of the bridge in the network and the requirement that every piece of

¹a more detailed analysis of this phenomenon follows later

cargo must cross over the bridge, a number of redundant constraints regarding the flow over the bridge can be easily generated. These constraints are thus of an aggregate nature.

In most, but not all, cases in this chapter, the model is set to stop after 1000 seconds. Therefore, unless otherwise stated, any discussion of comparison relates to the solution after 1000 seconds of solving time. The full tables of results are found in Appendix B.

First redundant constraint

The first redundant constraint, for reference named ‘DB1’, developed in this thesis in an attempt to increase the solving speed is of aggregate type. The constraint is as follows:

$$\sum_{v \in V} \sum_{l=1}^{M_v} \sum_{1 \in I_{POE}^{POD}} x_{vl}^l \geq \left\lceil \frac{\sum_{o \in O} \sum_{d \in D} D_o^d}{\max_{C_v \in V} C_v} \right\rceil \quad (4.1)$$

This is a perfectly natural and obvious assumption to make, that is, there must be at least as many trips across the bridge as the total demand divided by the largest capacity. Although very simple, in some cases this constraint has a drastic effect on the solving time. Take for example the dataset SpeedTest10, a dataset created by this author for the purpose of, as the name implies, testing the solving speed. Without this redundant constraint the solution was found after 1.5 hours. With the redundant constraint activated, the solution was found in 13.5 seconds.

To see why the solution was found so much quicker we must examine the solving status provided by Xpress-MP. To begin with we look at the following graphs for the first hundred seconds of the solver running in the absence of equation 4.1. It must be taken into account that Figures 4.1 and 4.2 show only the first 100 of 5400 seconds of running time.

From these figures we can see that the gap narrows very quickly at the beginning, and then takes around 80 seconds to reduce to $\approx 0.1\%$ by the 8th integer solution found (in this case the best solution is 971, and the best bound is 970, found after 60 seconds). The gap does not close any further until 90 minutes when the ninth integer solution, which is 970, is found. Compare this to the solution with the inclusion of equation 4.1, shown in Figure 4.3.

As can be seen in Figure 4.3, the gap narrows much quicker than without redundant constraint DB1. More integer solutions are found, and they are found very quickly, as well as the best bound reaching optimality in approximately 5 seconds. The drop from a solution of 971 to an optimal solution of 970, although taking a very long time in the absence of the constraint, happens within one second in the presence of the constraint. The large number of integer solutions is a positive result in its own right, as it allows for termination of the solver prior to optimality with the assurance of having found a range of integer solutions.

Another key aspect of comparing the effectiveness of redundant constraints is observing the state of the solution in the model without the constraint at the time it takes to solve to optimality in

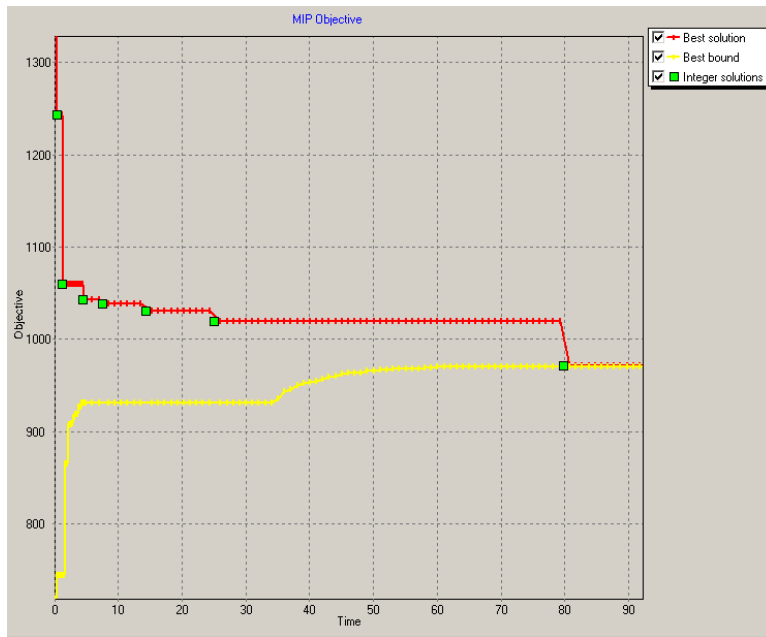


Figure 4.1: Graph of best bound(lower line) and best solution (top line, with square showing the point where integer solutions are found).

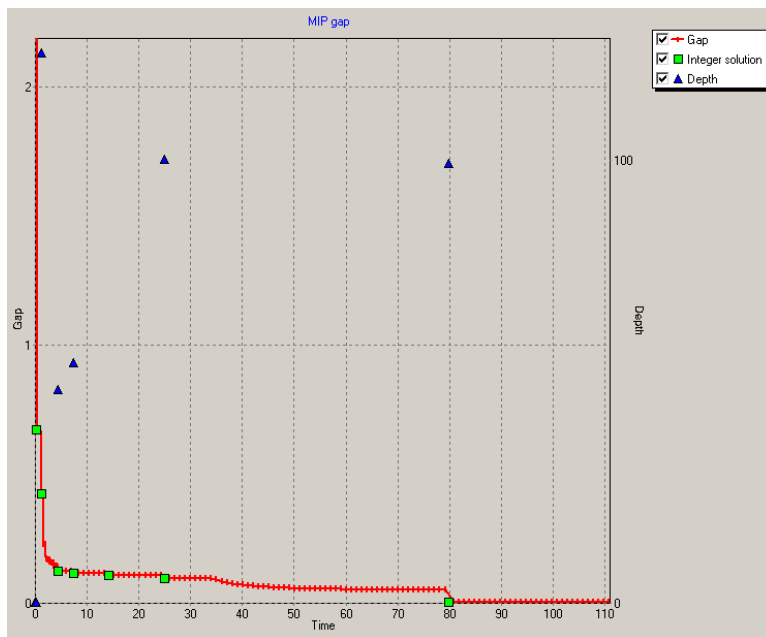


Figure 4.2: Graph showing the MIP gap, the solutions found and the depth (in the b-b tree) at which they were found.

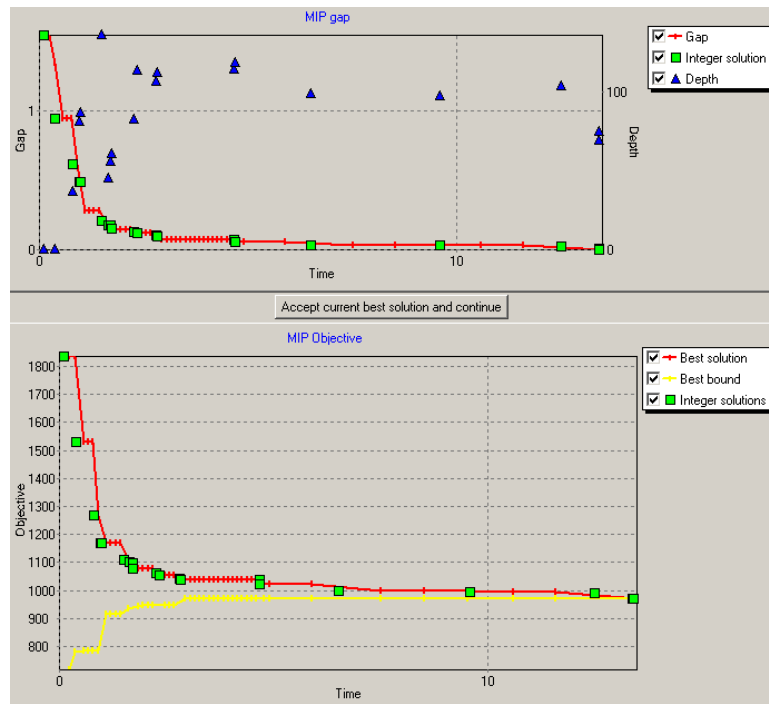


Figure 4.3: Graph of solution found using the redundant constraint.

the presence of the redundant constraint. In this case, as the solution in the presence of DB1 is found in 13.5 seconds we must observe the original model’s status at 13.5 seconds.

The solution found at 13.5 seconds (compared with the optimal solution, 970):

- A best bound of 931, found at 4.6 seconds
- A best solution of 1031, found at 14.9² seconds
- A gap of 10.74%, and a solution 6.29% off optimal

As can be seen by these figures, the constraint makes a significant impact in the solving time. The solution status is shown in Figure 4.4.

Although tremendously successful in this case, achieving more than a 99% reduction in solving time, this result is not replicated for every dataset that is simulated. Although in some data sets are solved to optimality within a fraction of the time of the unconstrained model, in a large number of cases there is barely any difference by the inclusion of DB1. In cases where the solver is allowed to run continuously, DB1 does seem to find better solutions by the time the memory is depleted. The table of comparisons is Table B.1.

²Although this is longer than the 13.5 seconds stated, the solver would stop only after this time. However, it is very close in terms of seconds elapsed, and in this case is acceptable

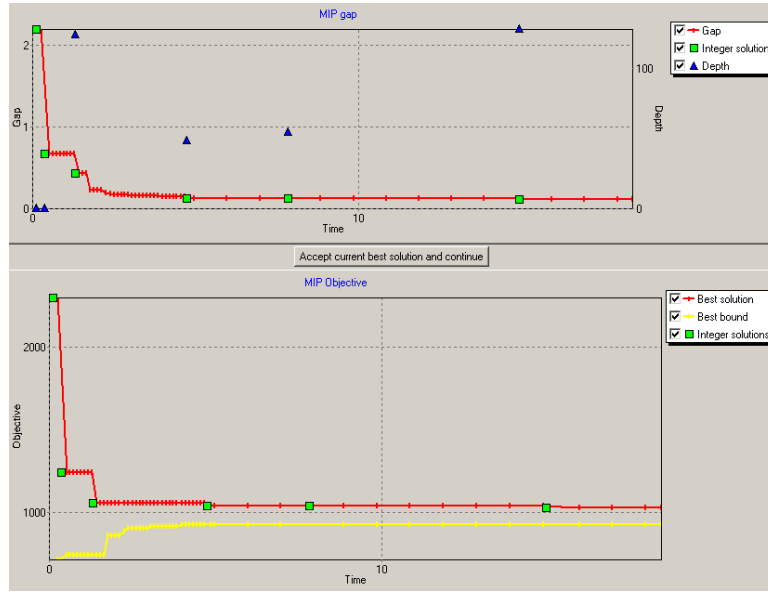


Figure 4.4: Status of the original model at the solution time of the constrained model.

Second redundant constraint

The second attempt at formulating a successful redundant constraint, DB2, takes advantage of the two-star topology of the network, and an understanding of the structure of an optimal solution.

$$\sum_{v \in V} \sum_{t=1}^{N_v} \sum_{l \in L_v} x_{vt}^l \leq 6 \times \left[\frac{\sum_{o \in O} \sum_{d \in \mathcal{D}} D_o^d}{\min C_v, \forall v \in V} \right] \quad (4.2)$$

In words, this constraint requires that the total number of trips in the solution be less than or equal to the total demand divided by the minimum capacity, multiplied by six. This is because, in an optimal solution, the total number of trips across any leg should be less than or equal to the amount of trips the smallest vessel would have to do to move all the cargo across the leg. Therefore, the total number of trips should be less than or equal to the value on the right of equation 4.2, the ‘mincap ratio’, multiplied by the length of the longest possible route (in legs). As can be seen in Figure 4.5, the longest path is six legs (NB this includes the return trip, so that a vessel can start another trip immediately).

In practice the value on the right hand side of 4.2 is far greater than the value on the left hand side. There are two reasons for this:

- 1) not every instance of demand will produce a longest-route
- 2) there will often be vessels with larger capacities than the vessel that defines the mincap

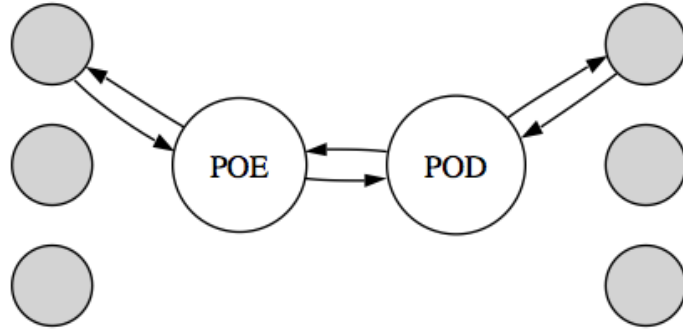


Figure 4.5: Longest possible route in the network, including return path.

(unless of course the entire fleet is homogeneous).

DB2 was included in the model with a modest effect. Of the first 21 data sets, four were solved to optimality compared to three in the unconstrained model. The solutions were, however, found in much less time. For all other data sets, the solutions of both DB2 and the unconstrained model were within 1% of each other with the *unconstrained* model usually slightly better. The hopes that DB2 would provide much faster solutions did not come to fruition, and the results are shown in Table B.2.

Third redundant constraint

DB3 is more-or-less a merger of DB1 and DB2. It simply states that the sum of trips across the bridge to the Theatre must be less than the mincap ratio, *in the aggregate (non-scheduling) model*.

$$\sum_{v \in V} \sum_{t=1}^{N_v} x_{vt}^l \leq \left\lceil \frac{\sum_{o \in O} \sum_{d \in \mathcal{D}} D_o^d}{\min C_v \in V} \right\rceil, \text{ for all } l \in L_{POE}^{POD} \quad (4.3)$$

Consider the case where 1000 tons of cargo were required to cross the bridge and, of all the vessel types available, the smallest vessel capacity across the bridge is 10 tons. In a situation where this, the smallest vessel, was required to move all cargo across the bridge by itself, it is easy to see that the number of trips across the bridge (from Australia) is 100. Therefore, it can be stated that the maximum number of trips across the bridge must be 100 for cases where each vessel carries a full load. Of course, it is unlikely that the maximum number of trips across the bridge would ever reach this number and empirical data reinforces this assumption³.

³Except in the case where the entire fleet is homogeneous, in which case equality must hold in 4.3

REVISION: Although highly unlikely given the structure of the problem, there exist scenarios in which Equation 4.3 does not provide a valid upper bound. Consider the case where the fleet is composed of only two ‘super-vessels’, so large that each one can transport the entire demand in one trip. Therefore, according to (4.3, the total number of trips across the bridge must be less than or equal to one, which forces one of the vessels to remain in Australia. However, once the cargo arrives in the Theatre it must be distributed to the locations in the Theatre, by means of splitting the cargo and transporting it along the set of legs from the POD to every location in the Theatre. It should be apparent that having both vessels in the Theatre at this point will provide a better solution. If there are, say, two locations in the Theatre, the sole vessel there must complete two forward trips and one backward trip to satisfy the demand. By allowing both vessels to enter the Theatre, even if one carries no cargo across the bridge, the distribution of cargo in the Theatre is completed faster, as each vessel needs only complete one forward trip.

Hence, the true form of DB3 should be

$$\sum_{v \in V} \sum_{l=1}^{N_v} x_{vt}^l \leq \max \left[\left[\frac{\sum_{o \in O} \sum_{d \in D} D_o^d}{\min_{C_v \in V} C_v} \right], \sum_{v \in V} N_v \right], \forall l \in L_{POE}^{POD} \quad (4.4)$$

Although Equation (4.4 is required formulation of DB3, in practice (that is, in this thesis) the two forms are equivalent as the number of vessels is usually much smaller than the mincap ratio.

□

The inclusion of DB3 into the model had a significant effect on the solving times of a large proportion of the data sets. Of the first 21 data sets, the standard program could only solve two to optimality within 1000 seconds, and another in over 5000 seconds. The inclusion of DB3 sees 12 data sets solve to optimality, and all 12 solved within 150 seconds. Of the remaining nine sets, the solutions found are within $\pm 1\%$ of the unconstrained solutions in the same elapsed time. See Table B.3 for the full list of results.

Fourth redundant constraint

DB4 is the most complicated of all the redundant constraints modeled so far. It is constructed by examining the distances between demand pairs.

For example, there is only one case where the distance, in legs, for a $D_o^d \neq 0$, is one leg, namely when o is the POE and d is the POD. Thus, the total demand over which the distance is one leg is given by equation 4.5:

$$D_{POE}^{POD} = A \quad (4.5)$$

For the case where the distance is two legs, we have either

- 1) o is the POE and d is in the Theatre but is *not* the POD, or

2) o is in Australia and *not* the POE, and d is the POD.

This is shown in equation 4.6:

$$\left(\sum_{o \in O \setminus \{POE\}} D_o^{POD} + \sum_{d \in D \setminus \{POD\}} D_{POE}^d \right) = B \quad (4.6)$$

For cases where the distance is three legs, we must have o is in Australian but not the POE, and d is in the Theatre but is not the POD.

$$\left(\sum_{o \in O \setminus \{POE\}} \sum_{d \in D \setminus \{POD\}} D_o^d \right) = C \quad (4.7)$$

Now that we have these definitions, we can model the constraint thusly:

$$\sum_{l \in \mathcal{L}_{forward}} x_{vt}^l \leq \left\lceil \frac{(A) + 2 \times (B) + 3 \times (C)}{C_v} \right\rceil, \forall v \in V, t \in N_v \quad (4.8)$$

As can be seen, this is a vessel specific version of DB2, and thus a tighter constraint. Simply put, it states that a vessel will only do as many forward legs as it would be required if that vessel was required to move all the cargo in the scenario.

DB4 also had a significant impact, solving 12 of the first 21 data sets to optimality. The solution times are reasonable, and in some cases extremely fast - it took only 0.64 seconds to solve SpeedTest10. In cases where the optimal solution was not found within the given time limit, the solutions are about the same as the unconstrained model. All results are in Table B.4.

Combinations of redundant constraints

Although the expected response of activating several redundant constraints would be an even greater decrease in solving time, this is not always the outcome. This may be to do with changing the structure of the problem so that the solver attempts to solve it in a different manner, but there is usually no clear reason.

Having said this, the effect of running the model under all four DB constraints proved to be successful. 12 of the first 21 data sets were solved to optimality, with all 12 solving in under 200 seconds. For the sets that didn't solve to optimality, the solutions found were plausible, with a maximum optimality gap of 3.4%. Table B.5 shows the full list of results.

The results suggest that this model does benefit from inclusion of redundant constraints, and particularly multiple constraints in unison.

4.2 Why are some models hard to solve?

Models that contain a very large number of variables are often hard to solve simply because the solution requires so much memory and processing power that computers cannot provide. However there are other instances where the size and structure of the problem would appear, at first glance, to suggest the solving time would be rather small, and yet there is such difficulty finding the optimal solution that it might not be found at all.

In these cases the reason that the model doesn't solve to optimality, by definition, is the gap between the best bound and best solution doesn't narrow to zero, even in cases where it expected to do so relatively quickly. This is usually because one or both of bound and solution get 'stuck', that is even after numerous iterations the solver can not find better values.

This phenomenon can sometimes be averted by the use of redundant constraints, however it must be noted that, in a large number of cases, specific constraints are required to deal with certain instances of this problem. Making the analysis of these cases even harder is the fact that the troublesome model must be solved to optimality in order to see where the problem occurs, and see which of the bound or solution (or both) are causing the problem.

A case in point is the example examined in the section on the first redundant constraint, solving SpeedTest10. Recall that the optimal solution was 970, and that a bound of 970 was found after 60 seconds, but the best solution remained at 971 for 90 minutes. In this case it was the solution that got stuck as the solver couldn't (quickly) find an integer solution of less than 971 (that is, it couldn't find an optimal solution).

In this model, empirical data suggests that certain combinations of vessel types cause great difficulty when solving the problem. For example InitialSpeedTest is a data set that is solved by any redundant constraint inclusion (but not by the unconstrained model). SpeedTest2 and SpeedTest3 are identical to InitialSpeedTest except that they add a 'Humvee' and an 'Explorer', respectively. Neither of these sets have been solved to optimality within a reasonable amount of time. This is peculiar, as in each case the total number of vessels increases by only one. Compare this to SpeedTest9, which includes 15 more vessels and is still solved to optimality fairly quickly.

It may be possible to add vessel specific constraints for these difficult cases to make solving easier. However, at this time there is no method for counteracting this phenomenon.

4.3 Nuances of the simulation

As discussed in chapter 3, the ALM creates a set of data that satisfies an aggregate solution, and then this data is used by a simulator to produce an initial feasible schedule. The simulator, if given the same set of data twice, will produce the same schedule both times. In this sense is not a simulation, but rather a conversion, as there is no random element in this step⁴.

⁴c.f. statistical simulation

The first key point to note in this section revolves around the second optimisation of the ALM. It is imperative to remember that the second optimisation attempts to (and, most of the time, does) find a solution with minimum closure time but a reduced number of trips than is found in the first optimisation. This means, for all intents and purposes, there are many ways of achieving the minimum closure time in terms of the number of trips and the scheduling of vessels and cargo.

The second key point refers to the way the solver works, particularly with the addition of redundant constraints. The redundant constraints are added to cut down the size of the solution space in order to find the solutions faster, but this also changes the structure of the problem. The solver in this model is set to solve problems in the same way, so if you solve a given problem twice, it will return the same answer twice. This is because in cases where there are multiple optimal solutions, running the solver several times will return several solutions (unless the solver is told otherwise as is the case here). However changing the structure of the problem, at least as far as the solver is concerned, changes the problem itself. Therefore the addition of redundant constraint(s) has the potential to return a different optimal solution, in terms of trips and schedule, than the solution in the absence of the redundant constraints.

When these two points are considered together, a strange occurrence begins to be observed. As the constraint for the second optimisation is based on the number of trips found in the first solution, and adding a redundant constraint changes the number of trips found in the first solution, then adding a redundant constraint changes the constraint for the second optimisation. By changing this constraint, the second optimisation will find a different number of trips, and hence a different schedule.

Although this seems innocuous at first, given the fact that the aggregate closure times are the same, the effect of this change on the simulation can be large. The different aggregate schedules give rise to often very different proper schedules.

A pertinent example of this was found by experimenting on the dataset SpeedTest19, under various versions of DB2. The dataset was solved in the absence of any redundant constraints, and the details of the solution noted. From this solution it was seen that the multiplier on the right hand side of DB2 (equation 4.2), taken as '6' as standard, could be reduced anywhere as low as '2'. This meant it was possible to observe the effect of modifying a redundant constraint, in this case increasing the multiplier by one each time and noting the solution. Table 4.2 displays the results.

Another example shows that a smaller ALM closure time does not necessarily imply a smaller simulation closure time. This was found when comparing the solution at the solving time of DB1, c.f. Figure 4.4. To illustrate this point, observe the following table of data:

As shown in table 4.1, despite a 6% increase in the aggregate closure time, the simulation closure time actually decreased.

Constraint status	Agg. Cl. Time	% Change	Initial Trips	Reduced Trips	Sim. Time
None	970	N/A	429	365	1236
DB1	970	N/A	440	420	1476
Time constrained	1031	6.29 %	429	365	1221

Table 4.1: Larger ALM closure time led to reduced simulation time (in the same model).

Multiplier	Agg. Cl. Time	Initial Trips	Reduced Trips	Simulation Time	% Change
Absent	884	316	261	1351	N/A
2 [#]	884	276	171	1351	0.00%
3	884	280	209	1229	-9.03%
4 [*]	884	404	345	1333	-1.33%
5	884	277	257	1121	-17.02%
6	884	369	251	1363	0.89%
7	884	277	169	1355	0.30%
8	884	335	241	1237	-8.44%
9	884	390	381	1319	-2.37%
10	884	357	318	1231	-8.88%
11	884	245	197	1351	0.00%
12	884	314	230	1286	-4.81%
13	884	373	261	1283	-5.03%
14	884	272	225	1280	-5.26%
15	884	290	253	1098	-18.73%
16 [*]	884	304	251	1098	-18.73%
17	884	319	259	1235	-8.59%
18	884	339	294	1190	-11.92%
19 [*]	884	269	238	1323	-2.07%
20 ^{*#}	884	393	386	1374	1.70%
21	884	350	329	1282	-5.11%
22 [#]	884	283	264	1191	-11.84%
23	884	386	216	1113	-17.62%
24	884	323	274	1037	-23.24%
25	884	306	169	1351	0.00%
26 [#]	884	298	247	1339	-0.89%
27	884	258	173	1351	0.00%
28	884	342	245	1367	1.18%
29	884	272	187	1351	0.00%
30 [#]	884	269	229	1351	0.00%

a b

^aMultipliers marked with an '*' increased the solving time slightly, whilst all others lowered it

^bMultipliers marked with a '#' had anomalies, such as vessels completing unnecessary trips

Table 4.2: Effect of changing the redundant constraint and comparison with the first line of the table, the absence of the redundant constraint.

Chapter 5

Symmetry and tails

The inclusion of tails into the model (having multiple vehicles from one vessel class) increases the difficulty of finding a solution (and thus the solving time), to the point where the problem may become so large it is unable to be solved to optimality. This is due to a phenomenon that sometimes occurs in IP branch-and-bound called *symmetry*. To understand how symmetry arises and why it causes such problems, we first must be comfortable with a few notions related to the problem.

Proposition 5.0.1. *If two lots of cargo, A and B, at the same location and required at the same destination, are to be transported by two tails of the same vessel type, tailOne and tailTwo, that depart at the same time, then the allocation of cargo to tails is inconsequential and both A and B will arrive at the destination at the same time.*

The proof of this is given in the appendix.

Corollary 5.0.2. *The above proposition can be extended to cases with more than two identical tails performing the same trip.*

Consider the case whereby the size of the cargo loads A and B are such that both A and B could be loaded into one tail (or equivalently that A is as usual, and B is empty). In this situation it is possible for the cargo to reach the destination with only one tail making the trip. This leads to the following statement.

Corollary 5.0.3. *If a unit of cargo can be taken by a particular vessel, and there is more than one tail of this vessel that can take the cargo, it does not matter which tail takes the cargo.*

In light of 5.0.3, it is now apparent that in all instances where there are n tails capable of doing exactly the same job, then there are n “parallel¹” outcomes for the job being completed in exactly the same amount of time. These parallel solutions are the reason that adding tails into the

¹Much the same as the notion of parallel universes in string theory, except in this case different actions result in exactly the same outcome

problem makes the solution much harder to calculate.

For those familiar with branch-and-bound solving techniques the nature of the problem may be fairly obvious, and many may have seen this phenomenon first-hand. The difficulties arise because if there are many ways of doing the same thing, the solution space in the branch-and-bound tree will have several distinct sub-spaces which are almost identical (and most likely will cause symmetrical sub-spaces of the original sub-spaces, and so on).

Consider the case where there are two identical tails at the same location at the beginning of the simulation. One of these tails is needed to transport a load of cargo immediately. Thus one of the first “decisions” is to choose one of these tails to transport the cargo. This is the creation of the symmetrical sub-spaces - one where the first tail is chosen and another where the second tail is chosen. This is the outcome for only two tails of one vessel type. In larger problems the compounding effect of the symmetry may be so great that solving the problem becomes too time consuming or even impossible given the amount of computing power required. Symmetry is indeed an important topic, and how to deal with it in specific cases can be found in the literature[14].

Unfortunately, there is no panacea for dealing with symmetry, and often the way to deal with symmetry is problem-specific. In [14], for example, this is done by fixing certain variables from the beginning to prevent (some) symmetry occurring.

One way the DSTO program has been written is to include a constraint designed to break symmetry in tails. The constraint says that in a set of 1,2,...,n tails, the number of trips undertaken by those tails is governed by a sequence of inequalities, such that

$$Numtrips(1) \geq Numtrips(2) \geq \dots \geq Numtrips(n)$$

and thus reduces some of the possible ways in which symmetry could occur by only looking at the first tail being allocated to do a trip when multiple tails could do the same trip.

Of course, the addition of redundant constraints can also help deal with symmetry. The more tightly a program is constrained, the smaller the solution space. Therefore, even if symmetry causes multiple copies of the solution subspace to be examined the effect will be lessened if it doesn't take long to solve each subspace. Thus, the more constrained a model is, the less the effect of symmetry.

Due to the nature of the method of solving, perhaps the best way to address symmetry is column generation, discussed in the next chapter.

Chapter 6

Column generation

As outlined in the previous chapters of this thesis, solving the problem to optimality is often hindered by symmetry and/or the size of the problem itself. For these reasons, the logical way to try to solve the problem is by using column generation.

Column generation could be used to not only find optimal aggregate solutions, but ideally to find optimal solutions to the scheduling of the problem also. For now, a framework for the column generation of an aggregate model will be proposed.

6.1 Aggregate Lift Column Generation

Beginning with an aggregate version of the problem for column generation is a good place to start as the construction is easily visualised. As is the usual first step for column generation, we start here by deriving a initial restricted matrix, denoted ' M_0 ', of feasible solutions.

Let the rows of M_0 be every leg in the network, both forwards and backwards and sea and air. We now consider modelling M_0 in the form of 'trip multi-sets'. A trip multi-set maps out a potential vessel movement for the scenario by describing the number of trips along each leg in the scenario. In this manner, the columns of M_0 are possible trip multi-sets for the scenario. An example is illustrated in Table 6.1.

In Table 6.1, the first column represents a possible aircraft movement comprising two trips from Brisbane to Darwin, two trips from Darwin to Brisbane, and so on. Column two represents a possible sea vessel movement. Notice that for each column there must be a feasible flow, subject to constraints 3.3 and 3.6. By only creating multi-sets that satisfy these constraints, means the exclusion of these constraints in solving as the information is already contained within the construction of M_0 .

The model can now be explained. Call M_0^j , the j^{th} column of M_0 , trip multi-set j . Consider the set of vessels, V . For each vessel $v_i \in V$, either v_i uses an M_0^j or it performs no trips.

Solution number	1	2	3	...
Brisbane-Darwin-Air	2	0	0	...
Darwin-Brisbane-Air	2	0	0	...
Brisbane-Perth-Air	3	0	8	...
Perth-Brisbane-Air	3	0	8	...
⋮				
Brisbane-Darwin-Sea	0	2	0	...
Darwin-Brisbane-Sea	0	2	0	...
⋮				

Table 6.1: Example of potential M_0 based on trip multi-sets.

Define a binary variable $y_{ij} = 1$, if v_i uses M_0^j , and $y_{ij} = 0$ otherwise. Therefore, we have

$$\sum_{j \in M_0} y_{ij} \leq 1, \forall i \in V$$

in other words, each vessel in the fleet is limited to using at most one trip multi-set.

The manner in which vessels are assigned to multi-sets must comply with the need to meet demand. Define

$$\begin{aligned} R_l^j & \text{ the number of trips on leg } l \text{ in multi-set } j \\ T_l & \text{ the total demand in tons across leg } l \end{aligned}$$

Then, recalling C_v^l denotes the capacity over leg l by vessel class v , we have

$$\sum_{j \in M_0} \sum_{i \in V} y_{ij} C_v^l R_l^j \geq T_l, \forall l \in L$$

in such a way that there is at enough capacity to cover the demand across each leg in the scenario. It should be apparent that M_0 must contain trip multi-sets that allow this construction. Of course, it is easy to come up with initial feasible solutions and M_0 can be made large very easily by many different permutations of multi-sets.

Although column generation is much more effective than pure integer programming on large problems, if the initial feasible solutions are far from optimal then the column generation procedure will run for a large number of iterations. As the purpose of column generation is to obtain results quickly it would be preferential to have the procedure terminate in few iterations. One way in which column generation can be made more efficient is by the use of ‘‘Column generation stabilisation’’[10]. This is unfortunately a difficult concept to put into practice.

Another way to increase the efficiency of the column generation is to begin with “better” feasible solutions in $M - 0$. Although the manner in which to accomplish this improvement may not be immediately apparent, it can be achieved by applying many features of the preceding chapters in this thesis.

By the nature of the problem, column generation is needed here only when the data sets become large and cannot be solved by linear programming. However, by solving smaller instances of a large problem using LP, it is possible to generate trip multi-sets that are part of the optimal solution to a smaller problem. By this technique, the initial columns of M_0 should be better than if they were created arbitrarily. Indeed, the improvement of the model in Chapter 4 allow reasonably large data sets to be solved quite quickly, either to optimality or near-optimality. This means that some of the initial columns of M_0 may be optimal at the beginning, which saves time when adding new columns. Of course, it must be evaluated whether the time it takes to solve an LP offsets the saving in time to solve the column generation problem.

Regrettably, these ideas for column generation have not been put into practice as of this time, and remain an aspect of future research.

Chapter 7

Conclusion

The aim of this thesis was to investigate and, ultimately, to improve the model for solving military lift developed for DSTO. Using careful consideration of the network structure and constraints, the linear programming formulation was substantially improved. The program was examined in detail and a great deal of empirical analysis was carried out to ensure the model's validity. A potential column generation method has been proposed, as well as a novel way of achieving better columns for the initial matrix.

Areas for further research include

- A full formulation of the column generation model described here
- Further investigation into the use of redundant constraints for the LP
- The inclusion of stochastic events into the model, representative of the uncertainty faced in real-world scenarios

Appendix A

Data sets

The following tables are the data sets generated for this Thesis and used for testing the model. The tables contain the name of the data set, and the fleet that is representative of that data set. All data sets have everything else in common, ie demand, etc. The scenario used for each data set is 'Madrid', except in the case where there is an additional 3 letter code at the end of the name which indicates the scenario used. For example, SpeedTest10ROM is exactly the same as SpeedTest10 except the scenario examined is Rome instead of Madrid.

It should be noted that not all data sets were tested, or not compared amongst the use of redundant constraints. A large number of these data sets were generated for another project for automated column generation, and were never intended to be examined in this thesis. Therefore, there are not results for each and every data set listed here.

Data Set	Prius	C'dore	S'man	I'gra	H'vee	Corolla	Edsel	XPT	Mill'm	L.Pacific	TGV	Thalis	E'star	Exp.	C'cornia	O'lander	Ghan
InitialSpeedTest	4		3					1			1						
SpeedTest1	4		3			1		1			1						
SpeedTest2	4		3	1				1			1						
SpeedTest3	4		3					1			1			1			
SpeedTest4	4		3					1			1				1		
SpeedTest5	4		3					1			1				1		
SpeedTest6	4		3					1			1						1
SpeedTest7	4		3					1			1			1			
SpeedTest8	4		3					1			1				1		1
SpeedTest9	4		3					1			1				4		6
SpeedTest10	4		3			1	2	1		1	1	1					
SpeedTest10ROM																	
SpeedTest10BRU																	
SpeedTest10ANK																	
SpeedTest11	4		3				1	1			1						
SpeedTest12	4		3			1	1	1			1						
SpeedTest13	4		3					1		1	1						
SpeedTest14	6		3					1			1						
SpeedTest15	5		3					1			1						
SpeedTest16	4		4					1			1						

Table A.1: Data sets and their constituent fleets.

Data Set	Prius	C'dore	S'man	I'gra	H'vee	Corolla	Edsel	XPT	Mill'm	I.Pacific	TGV	Thalis	E'star	Exp.	C'cornia	O'lander	Ghan
SpeedTest17	4		3					2			1						
SpeedTest18	4		3					1			2						
SpeedTest19	4		3					1			3						
SpeedTest20	4		3					1			4						
SpeedTest21		4	3					1			1						
SpeedTest22		3	3					1			1						
SpeedTest23		2	3					1			1						
SpeedTest24	1	2	3					1			1						
SpeedTest24ROM																	
SpeedTest25	1	2	2					1			1						
SpeedTest26	1	1	1					1			1						
SpeedTest27	4		3					1					1				
SpeedTest28	4		3					1			1		1	1	1	1	
SpeedTest29	4		3					1			1		1				
SpeedTest30	1	1	1	1				1			1						
SpeedTest31	1				1	1		1			1			1	1	1	
SpeedTest32		1		1				1						1	1	1	
SpeedTest33	4		3						1		1						
SpeedTest34	1																
SpeedTest35	10																

Table A.2: Data sets and their constituent fleets II.

Appendix B

Redundant constraint comparisons

This chapter contains the comparisons between the original model and the various constraints. In each table, the first three columns of time, gap, and closure time are for the original model, whilst the next three are for the constrained model. The final two columns show the change in closure time and solving time, where applicable.

	Time	Gap	Closure	Time*	Gap*	Closure*	CT change	Time change
InitialSpeedTest	1025	0.49%	1230	1030	0.49%	1230	0.00%	0.49%
SpeedTest1	1026	0.49%	1230	402	0%	1224	-0.49%	-60.82%
SpeedTest2	1135	3.10%	1074	1097	3.10%	1074	0.00%	-3.35%
SpeedTest3	1035	1.73%	1135	1031	1.70%	1135	0.00%	-0.39%
SpeedTest4	1025	0.49%	1230	1035	0.49%	1230	0.00%	0.98%
SpeedTest5	1023	0.49%	1230	1047	0.49%	1230	0.00%	2.35%
SpeedTest6	1023	0.49%	1230	1035	0.49%	1230	0.00%	1.17%
SpeedTest7	1027	1.70%	1135	1030	1.70%	1135	0.00%	0.29%
SpeedTest8	1028	0.24%	1227	1027	0.49%	1230	0.24%	-0.10%
SpeedTest9	1028	0.24%	1227	1029	0.49%	1230	0.24%	0.10%
SpeedTest10	5400	0.00%	970	13.52	0.00%	970	0.00%	-99.75%
SpeedTest11	1027	0.24%	1227	1027	0.49%	1230	0.24%	0.00%
SpeedTest12	1029	0.24%	1227	1027	0.49%	1230	0.24%	-0.19%
SpeedTest13	1026	1.90%	1095	1028	2.25%	1099	0.37%	0.19%
SpeedTest14	1070	2.31%	1131	1029	3.20%	1144	1.15%	-3.83%
SpeedTest15	1090	1.47%	1209	1101	1.60%	1210	0.08%	1.01%
SpeedTest16	1027	1.50%	1078	1082	2.82%	1085	0.65%	5.36%
SpeedTest17	1037	2.56%	1025	1025	5.80%	1027	0.20%	-1.16%
SpeedTest18	13.5	0.00%	971	3	0.00%	971	0.00%	-77.78%
SpeedTest19	5	0.00%	884	11	0.00%	884	0.00%	120.00%
SpeedTest20	1026	3.00%	675	1031	3.10%	675	0.00%	0.49%

Table B.1: Comparison of solutions between the original problem and DB1 constrained problem.

	Time	Gap	Closure	Time	Gap	Closure	CT change	Time change
InitialSpeedTest	1025	0.49%	1230	371	0%	1224	-0.004878049	-0.63804878
SpeedTest1	1026	0.49%	1230	1027	0.57%	1231	0.08%	0.10%
SpeedTest2	1135	3.10%	1074	1066	2.63%	1059	-1.40%	-6.08%
SpeedTest3	1035	1.73%	1135	1122	1.70%	1138	0.26%	8.41%
SpeedTest4	1025	0.49%	1230	1025	0.57%	1231	0.08%	0.00%
SpeedTest5	1023	0.49%	1230	1031	0.57%	1231	0.08%	0.78%
SpeedTest6	1023	0.49%	1230	1083	0.57%	1231	0.08%	5.87%
SpeedTest7	1027	1.70%	1135	1037	1.69%	1137	0.18%	0.97%
SpeedTest8	1028	0.24%	1227	1028	0.57%	1231	0.33%	0.00%
SpeedTest9	1028	0.24%	1227	1025	0.57%	1231	0.33%	-0.29%
SpeedTest10	5400	0.00%	970	6.7	0.00%	970	0.00%	-99.88%
SpeedTest11	1027	0.24%	1227	1071	0.57%	1231	0.33%	4.28%
SpeedTest12	1029	0.24%	1227	1073	0.57%	1231	0.33%	4.28%
SpeedTest13	1026	1.90%	1095	1142	1.65%	1095	0.00%	11.31%
SpeedTest14	1070	2.31%	1131	1040	1.58%	1129	-0.18%	-2.80%
SpeedTest15	1090	1.47%	1209	1044	1.50%	1212	0.25%	-4.22%
SpeedTest16	1027	1.50%	1078	1065	1.26%	1077	-0.09%	3.70%
SpeedTest17	1037	2.56%	1025	1046	2.70%	1027	0.20%	0.87%
SpeedTest18	13.5	0.00%	971	2	0.00%	971	0.00%	-85.19%
SpeedTest19	5	0.00%	884	2.9	0.00%	884	0.00%	-42.00%
SpeedTest20	1026	3.00%	675	1028	3.53%	679	0.59%	0.19%

Table B.2: Comparison of solutions between the original problem and DB2 constrained problem.

	Time	Gap	Closure	Time	Gap	Closure	CT change	Time change
InitialSpeedTest	1025	0.49%	1230	132		1224	-0.49%	-87.12%
SpeedTest1	1026	0.49%	1230	107		1224	-0.49%	-89.57%
SpeedTest2	1135	3.10%	1074	1123	2.27%	1067	-0.65%	-1.06%
SpeedTest3	1035	1.73%	1135	1030	2.19%	1141	0.53%	-0.48%
SpeedTest4	1025	0.49%	1230	112		1224	-0.49%	-89.07%
SpeedTest5	1023	0.49%	1230	108		1224	-0.49%	-89.44%
SpeedTest6	1023	0.49%	1230	110		1224	-0.49%	-89.25%
SpeedTest7	1027	1.70%	1135	1036	2.19%	1141	0.53%	0.88%
SpeedTest8	1028	0.24%	1227	107		1224	-0.24%	-89.59%
SpeedTest9	1028	0.24%	1227	108		1224	-0.24%	-89.49%
SpeedTest10	5400	0.00%	970	11		970	0.00%	-99.80%
SpeedTest11	1027	0.24%	1227	107		1224	-0.24%	-89.58%
SpeedTest12	1029	0.24%	1227	110		1224	-0.24%	-89.31%
SpeedTest13	1026	1.90%	1095	1106	2.50%	1103	0.73%	7.80%
SpeedTest14	1070	2.31%	1131	1029	2.50%	1136	0.44%	-3.83%
SpeedTest15	1090	1.47%	1209	1084	1.06%	1206	-0.25%	-0.55%
SpeedTest16	1027	1.50%	1078	1060	2.50%	1088	0.93%	3.21%
SpeedTest17	1037	2.56%	1025	1050	6.17%	1031	0.59%	1.25%
SpeedTest18	13.5	0.00%	971	4	0.00%	971	0.00%	-70.37%
SpeedTest19	5	0.00%	884	14.6	0.00%	884	0.00%	192.00%
SpeedTest20	1026	3.00%	675	1029	4.50%	683	1.19%	0.29%

Table B.3: Comparison of solutions between the original problem and DB3 constrained problem.

	Time	Gap	Closure	Time	Gap	Closure	CT change	Time change
InitialSpeedTest	1025	0.49%	1230	499		1224	-0.49%	-51.32%
SpeedTest1	1026	0.49%	1230	772		1224	-0.49%	-24.76%
SpeedTest2	1135	3.10%	1074	1055	1.66%	1063	-1.02%	-7.05%
SpeedTest3	1035	1.73%	1135	1113	1.86%	1138	0.26%	7.54%
SpeedTest4	1025	0.49%	1230	775		1224	-0.49%	-24.39%
SpeedTest5	1023	0.49%	1230	775		1224	-0.49%	-24.24%
SpeedTest6	1023	0.49%	1230	775		1224	-0.49%	-24.24%
SpeedTest7	1027	1.70%	1135	1046	1.86%	1138	0.26%	1.85%
SpeedTest8	1028	0.24%	1227	765		1224	-0.24%	-25.58%
SpeedTest9	1028	0.24%	1227	774		1224	-0.24%	-24.71%
SpeedTest10	5400	0.00%	970	0.64		970	0.00%	-99.99%
SpeedTest11	1027	0.24%	1227	780		1224	-0.24%	-24.05%
SpeedTest12	1029	0.24%	1227	762		1224	-0.24%	-25.95%
SpeedTest13	1026	1.90%	1095	1091	1.54%	1092	-0.27%	6.34%
SpeedTest14	1070	2.31%	1131	1026	2.16%	1133	0.18%	-4.11%
SpeedTest15	1090	1.47%	1209	1074	1.56%	1211	0.17%	-1.47%
SpeedTest16	1027	1.50%	1078	1067	1.52%	1078	0.00%	3.89%
SpeedTest17	1037	2.56%	1025	1030	2.71%	1029	0.39%	-0.68%
SpeedTest18	13.5	0.00%	971	1.5		971	0.00%	-88.89%
SpeedTest19	5	0.00%	884	2.9		884	0.00%	-42.00%
SpeedTest20	1026	3.00%	675	1035	3.50%	679	0.59%	0.88%

Table B.4: Comparison of solutions between the original problem and DB4 constrained problem.

	Time	Gap	Closure	Time	Gap	Closure	CT change	Time change
InitialSpeedTest	1025	0.49%	1230	173		1224	-0.49%	-83.12%
SpeedTest1	1026	0.49%	1230	91		1224	-0.49%	-91.13%
SpeedTest2	1135	3.10%	1074	1035	1.96%	1063	-1.02%	-8.81%
SpeedTest3	1035	1.73%	1135	1152	1.69%	1136	0.09%	11.30%
SpeedTest4	1025	0.49%	1230	90		1224	-0.49%	-91.22%
SpeedTest5	1023	0.49%	1230	90		1224	-0.49%	-91.20%
SpeedTest6	1023	0.49%	1230	93		1224	-0.49%	-90.91%
SpeedTest7	1027	1.70%	1135	1052	1.69%	1136	0.09%	2.43%
SpeedTest8	1028	0.24%	1227	93		1224	-0.24%	-90.95%
SpeedTest9	1028	0.24%	1227	93		1224	-0.24%	-90.95%
SpeedTest10	5400	0.00%	970	10		970	0.00%	-99.81%
SpeedTest11	1027	0.24%	1227	91		1224	-0.24%	-91.14%
SpeedTest12	1029	0.24%	1227	92		1224	-0.24%	-91.06%
SpeedTest13	1026	1.90%	1095	1092	2.04%	1097	0.18%	6.43%
SpeedTest14	1070	2.31%	1131	1005	2.12%	1133	0.18%	-6.07%
SpeedTest15	1090	1.47%	1209	1033	2.47%	1220	0.91%	-5.23%
SpeedTest16	1027	1.50%	1078	1081	2.08%	1084	0.56%	5.26%
SpeedTest17	1037	2.56%	1025	1026	2.55%	1026	0.10%	-1.06%
SpeedTest18	13.5	0.00%	971	1.7		971	0.00%	-87.41%
SpeedTest19	5	0.00%	884	3		884	0.00%	-40.00%
SpeedTest20	1026	3.00%	675	1055	3.42%	678	0.44%	2.83%

Table B.5: Comparison of solutions between the original problem and all constraints.

Appendix C

Symmetry proof

Proposition C.0.1. *If two lots of cargo, A and B, at the same location and required at the same destination, are to be transported by two tails of the same vessel type, tailOne and tailTwo, that depart at the same time, then the allocation of cargo to tails is inconsequential and both A and B will arrive at the destination at the same time.*

Proof. Consider otherwise:

Let A loaded onto tailOne arrive at the destination before B on tailTwo. In order for this to occur, we have

$$\text{Transit time of A} < \text{Transit time of B}$$

which implies

$$\frac{\text{length of leg}}{\text{Speed of tailOne}} < \frac{\text{length of leg}}{\text{Speed of tailTwo}}$$

However, the vessels are traversing the same leg, and as such the length of leg is equal on both sides, which leads to

$$\frac{1}{\text{Speed of tailOne}} < \frac{1}{\text{Speed of tailTwo}}$$

which means

$$\text{Speed of tailOne} > \text{Speed of tailTwo}$$

But tailOne and tailTwo are the same vessel class, and are identical in terms of capabilities including speed, and thus we have a contradiction. †

Appendix D

Errata

Redundant constraint DB1

Recall that the equation for the redundant constraint DB1 is:

$$\sum_{\substack{v \in \text{vessels}, \\ 1..n \in \text{num.vessels}(v)}} \text{Trips from poe to pod} \geq \left\lceil \frac{\text{Total demand from Aus to Theatre}}{\text{Maximum vessel capacity in the set}} \right\rceil \quad (\text{D.1})$$

The actual code in place for this constraint read:

$$\sum_{v \in V} \sum_{t=1}^{N_v} x_{vt}^I \geq \left\lceil \frac{\sum_{o \in O} \sum_{d \in \mathcal{D}} D_o^d}{\max_{C_v \in V} C_v} \right\rceil, \forall l \in L_{POE}^{POD} \quad \square \quad (\text{D.2})$$

In this case the forall statement is applying the constraint only to trips that start at the POE and end at the POD, ie the Bridge in the direction of Australia to the Theatre. By the nature of the network structure there is only one of these legs, as that is the whole definition and point of the bridge.

However, the way the problem is set up (and the very aspect that allows the model to deal with both sea and air vessels) is that there are effectively two networks, one of sea legs and one of air legs. Hence the original version was forcing a minimum number of trips for each of sea and air vessels when the intent was to force a minimum number of combined trips of sea and air vessels.

This was solved by including the legs in the summation, namely a minimum number of trips over the sum of all (read: both) bridge legs. The final version of the constraint looked like:

$$\sum_{v \in V} \sum_{t=1}^{N_v} \sum_{l \in L_{POE}^{POD}} x_{vt}^l \geq \left\lceil \frac{\sum_{o \in O} \sum_{d \in \mathcal{D}} D_o^d}{\max_{C_v \in V} C_v} \right\rceil \quad \blacksquare \quad (\text{D.3})$$

Bibliography

- [1] Baker, Steven F.; Morton, David P.; Rosenthal, Richard E.; Williams, Laura Melody; Optimizing military airlift, *Operations Research*, Vol. 50, No. 4, (2002)
- [2] Cynthia Barnhart; Niranjana Krishnan; Daeki Kim; Keith Ware; Network design for express shipment delivery, *Computational optimization and applications*, 21, (2002)
- [3] Personal Communication, Associate Professor Natashia Boland, 2006
- [4] Boland, Natashia and Sotirov, Renata; 620-362 course notes, The University of Melbourne, 2005
- [5] Boland, N. and Surendonk, T.; A column generation approach to delivery planning over time with inhomogeneous service providers and service interval constraints, *Annals of Operations Research*, 108 (2001)
- [6] Personal Communication, Dr Heng-Soon Gan, 2006
- [7] Haghani, Ali and Oh, Sei-Chang; Formulation and solution of a multi-commodity, multi-modal network flow model for disaster relief operations, *Transpn. Res.-A.*, vol. 30, no. 3, (1996)
- [8] Daeki Kim; Cynthia Barnhart; Keith Ware; Multimodal express package delivery: a service network design application, *Transportation science*, vol 33, no. 4, (1999)
- [9] Land, A. H. ; Doig, A. G.; An automatic method of solving discrete programming problems, *Econometrica* 28 1960 497520
- [10] Lübbecke, Marco E. and Desrosiers, Jacques; Selected topics in column generation, *Oper. Res.* 53 (2005), no. 6, 10071023.
- [11] Morton, David P.; Salmerón, J; and Wood, R. Kevin; A stochastic program for optimizing military sealift subject to attack, 2002.
- [12] Nielsen, C.A.; Armacost, A.P.; Barnhart, C.; Kolitz, S.E.; Network design formulations for scheduling U.S. air force channel route missions, *Mathematical and computer modelling*, Vol. 39, (2004)
- [13] Personal Communication, Tim Robinson, 2006

- [14] Valério de Carvalho, J.M.; Exact solution of bin-packing problems using column generation and branch-and-bound, *Ann. Oper. Res.*, Vol. 86, (1999)
- [15] Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Eulerian_cycle
- [16] Wayne L. Winston, Operations research: applications and algorithms, 3rd edition, Duxbury Press, 1994.
- [17] Zhou, Sanming; 620-463 course notes, The University of Melbourne, 2006