Optimal Loading of Transportation Conveyances

Yosef Sheffi
Shangyao Yan
Benjamin Teitelbaum

The capacity of transportation conveyances is typically constrained by both their volume and their weight. In many cases the operator does not have extra conveyances and has to manage the booking for each conveyance judiciously in order to maximize profit. This paper addresses the optimal loading of a transportation conveyance in the context of the loading of a vessel. The problem is formulated as a linear integer program and a special purpose branch and bound algorithm is developed and applied to solve it. A microcomputer-based decision support system and a case study are also described.

This paper develops a methodology for optimal loading of a transportation conveyance. The issue has been addressed in the context of an international vessel operation but the methods employed are applicable to the loading of barges, railcars, trailers, trucks, airplanes or any other conveyance. The method developed has been implemented in a user-friendly microcomputer package. This paper describes the problem, the algorithmic solutions approach, the implementation and case study.

The general problem is to load a transportation conveyance with a set of shipments, each characterized by revenue, volume and weight. Some of these shipments have to be taken in their entirety (if serviced at all) while others may be broken and taken partially. The conveyance itself is subject to both weight and volume constraints. The optimal loading problem has been formulated as a mixed integer program in which the decision variables are to take or not to take specific shipments.

The objective parameters in the problem studied here do not include any costs. The reason is that when the optimal loading problem is solved, the conveyance and all the costs of operating it are known and fixed. In fact, the algorithm calculates these costs before any optimization is performed. The objective then is to maximize the conveyance revenue subject to the volume and weight constraints. Since the costs are fixed, this amounts to profit maximization. Naturally, given the solution procedure, the conveyance's cost parameters and other problem characteristics can be changed so one can ask various "what if?" questions regarding the operation of the conveyance.

This paper is organized as follows: The authors describe the particular setting for the problem addressed here, and formulate the linear program (when all shipments can be taken partially) and describe the solution algorithm. Then, the problem is generalized to the integer programming case; the authors outline the software implementation, then present an actual case study. Finally, the authors summarize and conclude the paper.

Problem Setting

As mentioned above, the optimal loading methodology was applied to the case of a vessel traveling between a given set of foreign freight origins (all within the same country) and a set of U.S. destinations. The operation under consideration involves a monthly chartered ship which has to be contracted separately prior to each voyage.

The problem solved directly by the algorithm described here is the following:

Given: vessel capacity (weight and volume)

vessel cost parameters

cost per port visit for each potential vessel and port combination

shipments parameters (weight, volume, revenue, handling costs)

Find: which shipments should be taken to maximize profit

The vessel's booking agent faces the following problems:

1. Determine which vessel to charter,
2. Given the vessel characteristics, determine its routing, and
3. Given the vessel and its routing, determine which shipments should be booked.

Typically, the vessel to be used for a particular voyage is decided upon and contracted about one month ahead of its arrival at the first loading port. The information available before this decision is made includes a set of projections of the possible booked shipments. At that point the ship broker, who works for the shipping line, scans the international charter market electronically. To decide on the best size and ship characteristics, the line has to get an idea of what the expected revenue may be with each ship type. Thus, without even entering the detailed routing, the optimal loading model can be solved many times—once for each available vessel type—to

determine the best one to order. Once the vessel is ordered, the freight booking agent in the origin country, together with the lines, has to determine which ports to call on there. A port call includes a fixed charge (for port entry, wharfage, and other port services) in addition to the vessel time charges (the daily charter on a vessel can range from less than $5,000 to more than $30,000, depending on the ship size, condition, and the conditions in the international market). The fixed charges for port entry are handled exogenously in the model described here. In other words, the model does not determine internally which port should be visited. Instead, the user can easily and quickly input the routing, and the model will then determine the vessel cost (using the vessel cost parameters in conjunction with the ports' cost parameters), optimize the loading and report the vessel revenue and profit. By trying several combinations of loading ports, the line decides on the best ship routing. (Note that the cost of a port stay depends, to some extent, on the amount and type of freight involved. These relationships are not modeled here because they are not strong and the data is not available until the shipment actually shows up.)

Lastly, when the routing is fixed, the line starts to confirm the booking of shipments. At this point, some bookings have already been confirmed (based on prior model runs, experience, or long term contractual commitments) and they represent "must take" loads. Other shipments are being called in by customers, or solicited by the freight booking agent. In any case the line has to decide which ones to take. This is where the data available is most accurate and the optimization results in actual confirmations. In actual operations, some shipments are not confirmed until the last minute so that, if more profitable shipments become available later, those can be taken instead. This means that the model may be run many times in the few days before all cargo is positively confirmed and loaded on the vessel.

In actual operations these stages are not distinct. For example, the vessel booking agent will typically work closely with the commercial agent at the origin country to determine both vessel type and routing simultaneously. Similary, the routing and the booking confirmation process may overlap as the line tries to maximize profit. This is demonstrated in the case study.

At the heart of the computer model is the ship loading optimization and the ability to change the parameters of this optimization quickly and easily. The next two sections discuss the optimization methodology, and the one after describes the software.

The Linear Program

Let us first discuss the continuous ship loading problem, where fractional shipments can be taken. This problem is easier than the one where some shipments are subject to an "all-or-nothing" policy. The importance of discusing this problem is twofold: first, this is the problem which is typically solved in the process of deciding on the ship type. When this decision is made, the shipment information is not accurate enough and the line does not know which shipments are subject to the all-or-nothing policy. Second, and more important, the solution method for the continuous problem is at the heart of the solution to the integer program.

To formulate the optimal loading problem, assume that there are n available shipments numbered arbitrarily 1,...,n. Now define the following variables as follows:

x_i = the fraction of shipment i confirmed for loading, i=1,...,n

p_i = revenue of the i-th shipment

v_i = volume of the i-th shipment

w_i = weight of the i-th shipment

V = total volume available on ship

W = total weight available on ship

The optimal loading problem, LP, can now be described as follows:

$$\text{LP} \quad \max Z(x_1,\ldots,x_n) = \sum_{i=1}^{n} p_i x_i \tag{1}$$

subject to:

$$\sum_{i=1}^{n} v_i x_i \leq V \tag{2}$$

$$\sum_{i=1}^{n} w_i x_i \leq W \tag{3}$$

$$0 \leq x_i \leq 1 \qquad \forall i = 1,\ldots,n \tag{4}$$

This is a simple linear program. Note that if either the volume or the weight constraint were not present, the problem would have reduced to the well-known ***knapsack,*** which can be solved trivially.[1, 2] In fact, our procedure for obtaining a good starting solution draws from this similarity.

Program LP can, of course, be solved by a direct application of the simplex method. In this case, the size of the resulting basis will be $(n + 2) \times (n + 2)$ counting the volume constraint, the weight constraint, and the n upper bound constraints on the decision variables. Since n is typically in the hundreds, using such an approach may take too long on a microcomputer. This is true, in particular, considering that the solution procedure will have to be used iteratively within a method for solving the integer program and since the resulting software has to be used interactively.

A well-known alternative approach for dealing with just such problems is the simplex with bounded variables.[3] In this method, the upper bounds on the decision variables are treated in a way analogous to the non-negativity constraints on these variables. Thus the basis is of size 2×2 regardless of n. A variables x_i is considered out of the basis when it is either at zero or at its upper bound ($x_i = 1$ in our case).

A pivot with the bounded variable simplex method is similar to the standard pivot but with a few modifications. In particular, the condition for optimality (maximum) is for every non-basic variable at zero to have a non-positive objective coefficient (as usual) and for every non-basic variable at its upper bound to have a non-negative objective coefficient. To improve a non-optimal solution, a pivot may decrease the value of a non-basic variable at its upper bound if its objective coefficient is negative. Unlike in standard simplex, here a non-basic variable can move its upper to its lower bound and remain non-basic. Only if a *basic* variable reaches a bound as a result of increasing or decreasing the value of a non-basic variable will an actual change of basis take place.

Note that it is not necessary to check for unboundedness (if the inputs are correct) since the bounds will always prevent it. Note also that the simplex described here does not require a two-phase procedure due to the form of the volume and weight constraints. An initial solution may be found by fractional variables and the rest, say to zero (a better procedure is described below). To start pivoting, the two fractional variables determined by the initialization procedure are simply forced into the basis in the first two pivots (driving out the two slacks).

The only possible drawback of this algorithm is that when started with a random basis it may take a large number of pivots to find optimality. To alleviate this concern we determine an initial solution (basis) by using a procedure reminiscent of that utilized to solve *knapsack* problems. The idea is to sort all shipments by an *occupancy index* which is defined, for each shipment, as the bigger of the ratio of the shipment's profit to the portion of volume or weight it occupies. In other words, the index for shipment i, I_i, is given by:

$$I_i = p_i \cdot \max\left(\frac{V}{v_i}, \frac{W}{w_i}\right) \quad (5)$$

Shipments are taken from the top of the list down (i.e., x_i is set to one for these shipments) until the ship capacity (either volume or weight) is exhausted. The last shipment, then, is taken in part to exhaust the applicable constraint. The following is a pseudo-code for this procedure.

Procedure Initialize_Simplex

Begin

Sort all shipments by highest to lowest I_i;

Renumber shipments in sorted list in order, so that highest I_i is for shipment number 1

Let $V_{TEMP} = V$; $W_{TEMP} = W$; Z=0;

For i = 1 to n do:

If $v_i \le V_{TEMP}$ and $w_i \le W_{TEMP}$ then

$Z = Z + p_i$;

$V_{TEMP} = V_{TEMP} - v_i$;

$W_{TEMP} = W_{TEMP} - w_i$;

$x_i = 1$;

Else

$x_i = min\left\{\frac{W_{TEMP}}{w_i}, \frac{V_{TEMP}}{v_i}\right\}$

$Z = Z + x_i p_i$

Go to end of procedure

End_if

End_for

End Initialize_Simplex

Several other obvious variations to this procedure (e.g. sorting by a volume-based ratio or a weight-based ratio alone or using $p_i \cdot \min\{V/v_i, W/w_i\}$) were tried but proved to be less efficient than the one described above.

The Integer Program

Many of the shipments booked have to be taken either in their entirety or not at all. The reason is that shippers are often unwilling to split shipments due to the inconvience associated with extra paperwork and more difficulty in tracking. In such cases, an additional constraint should be added to LP, to wit:

$$x_i = \{0,1\} \text{ for some } i \tag{6}$$

The program (1), (2), (3), (4), (6) is referred to here as IP. It is a mixed integer program which can be solved by a variety of methods including dynamic programming, cutting planes and branch-and-bound.[4] We have focused on branch-and-bound (or implicit enumeration) due to its versatility and simplicity.

The branch-and-bound method works with a given feasible solution *(incumbent)*. At every point in the solution process, the algorithm focuses on a candidate problem. At this stage, some of the integral x_i values are determined and others are "free" (i.e. yet to be determined). The algorithm works here by relaxing the integrality constraint (6) on the free variables and using the simplex method described in the previous section to solve the resulting LP. If the resulting solution satisfies certain conditions mentioned below, then this candidate problem is considered *fathomed.* Otherwise a fractional x_i is to $x_i = 0$ and $x_i = 1$, removing, in effect, the ith shipment from being "free" to being "determined." This *"branching"* results in two problems which can, in turn, be replaced with respect to their free ariables and the process repeats itself. The various candidate problems can be organized in a binary tree whose nodes each represent a problem with $x_i = 1$ for some i, $x_i = 0$ for others and x_i free (undetermined) for the rest.

The fathoming process is central to the branch-and-bound method since it permits an explicit enumeration of the entire set of candidate problems (the "tree"). A given node of the branch-and-bound tree can be fathomed if one of the following three conditions takes place:

1. Either V or W is exceeded by the set of $x_i = 1$ at this node,

2. the objective value of the LP there is not greater than the objective value of the incumbent (this is the ***bounding*** mechanism),

3. the LP at this node solution satisfies the integrality constraint.

If fathoming is due to the third criterion above and the objective function value at that point is higher than that of the incumbent, the new solution becomes the incumbent. In any event, if fathoming occurs at all, the current node of the branch-and-bound "tree" is not branched upon any more and the algorithm branches on a different node. The algorithm continues in this fashion until all nodes are fathomed. Since the number of nodes in the branch-and-bound tree is finite, the algorithm is guaranteed to generate a solution in a finite number of iterations. The incumbent at that point is the globally optimal solution. Three elements influence greatly the performance of this branch-and-bound algorithm: the choice of initial incumbent, the branching procedure and the data structures used. These are described below.

Initial Solution

A good initial incumbent will have a high objective function value, causing early fathoming of the branch-and-bound tree by tight bounding (see the second condition above), thus expediting the algorithm. Our initialization method is a modification of the aforementioned LP initialization method. We start by relaxing the problem and solving the resulting LP. All the shipments for which $x_i = 1$ in the solution are labeled as *taken,* while any non-integer shipment and shipments not taken constitute the set of *potential shipments.* Essentially, we now use the same ranking procedure described in connection with the LP with two exceptions. It works only on the *potential shipments* and, instead of terminating with a fractional shipment, we continue searching down the occupancy index list for a shipment which can best fit in its entirety within the weight and volume constraints. The value of the objective function, once the ranked list is exhausted, is the initial incumbent value. The pseudo-code for this program is the following:

Two other similar initialization heuristics were investigated. The first uses the procedure outlined above but skips the simplex solution. In other words, all shipments are included in the set of potential shipments. The second heuristic was also similar to that described above but used various other sorting criteria to rank the potential shipments. None of these procedures performed as well as the one shown.

Branching Criterion

At each stage of the branch-and-bound algorithm, one can make a choice regarding which node of the three to branch on. A good rule for choosing a branching node can expedite the algorithm significantly by leading to good incumbents, thereby causing early fathoming.

Procedure Initial_Incumbent

Begin

Solve LP by the simplex method;

Set any fractional x_i to x_i=0;

Set $W_{TEMP} = W - \sum_{i=1}^{n} w_i x_i$; $V_{TEMP} = V - \sum_{i=1}^{n} v_i x_i$; $Z = \sum_{i=1}^{n} p_i x_i$;

Sort all shipments by I_i and renumber from highest to lowest;

For i=1 to n while x_i=0 do

If $v_i \leq V_{TEMP}$ and $w_i \leq W_{TEMP}$ then

$Z = Z + p_i$;

$V_{TEMP} = V_{TEMP} - v_i$;

$W_{TEMP} = W_{TEMP} - w_i$;

$x_i = 1$;

Else if (x_i can be fractional) then

$x_i = \min\left(\frac{V_{TEMP}}{v_i}, \frac{W_{TEMP}}{w_i}\right)$;

$Z = Z + p_i x_i$;

Go to end of procedure;

End_if;

End_for;

End Initial_Incumbent.

In general, one can think of many strategies to choose the next node to branch on. If these nodes are stored in a *candidate list* then the list can be managed, say, by a *FIFO* (first-in-first-out) rule, *LIFO* (last-in-first-out) rule, or some other list discipline. Alternatively, the priority can be based on the tree's geometry, as in *depth-first* or *breadth-first* search. A third alternative is to create a special search criterion which measures the fathoming potential of a node and then branches on nodes in order of their highest potential.

A FIFO discipline is managed as a queue. In general, if offers no particular advantages over a random choice of nodes. A LIFO discipline offers some advantage in terms of being able to reconstruct the next

problem branched from the last one. A similar advantage occurs with a depth-first search where the algorithm backtracks up each branch after fathoming a lower node. A breadth-first procedure can be advantageous in cases where at each level of the tree one is branching on the same variable. This procedure will save core, since one can know which variables are set and which are free just by tracking the location of a node. Thus prior solutions can be easily reconstructed. The advantage of all these branching criteria is irrelevant in our case since our initialization procedure was found to be more effective than an initial solution based on any downstream nodes. In other words, our procedure did not profit from the ability to reconstruct any prior solution.

Thus, in an environment in which none of the structure-driven branching disciplines appears to be particularly effective, it seems intuitive that a good search criterion based on the potential of each node may help trim the tree efficiently. The criterion used here is based simply on the value of the relaxed objective function. Thus the next node to branch on is always the one with the highest relaxed objective value. The rationale here is that there may be more "potential" for good solutions in branches emanating from such a node than from any other node. And indeed, tests on random data sets indicate that branching based on this *largest-upper-bound-next* procedure needed to enumerate on average only half the number of nodes in the branch-and-bound trees that were evaluated by all other procedures tested (the performance of all of which was very similar to each other).

Another decision which has to take place in the proces of branching is the choice of which *variable* to branch on once it is determined that a given node is not fathomed. Naturally, there are two candidates—the two fractional variables in the LP solution. Numerical experiments here have demonstrated a slight advantage towards choosing the variables with the higher occupancy index (see equation 5).

Data Structures and Algorithm

To execute the algorithm efficiently and save on core usage, one can use several data structures. The information kept for each partial solution at a given node of the branch-and-bound tree includes the variables already branched (and their values) and the next variable to branch on. To save memory, the information regarding nodes already examined is discarded by rewriting the same memory locations with information regarding new nodes. This is done by simply setting the index of the first descendent of a node which needs work to the index of its predecessor. If the second descendent is also a candidate, its node number is set to one more than the current total number of candidates (this number needs to be continuously updated).

nodes which require branching in a candidate list. This list is sorted by the branching criterion—in our case the value of the LP solution. The list is updated whenever a node is branched on (the next node becomes the top of the list and whenever it is determined that a node is not fathomed. Such a node will require branching and thus has to be properly inserted in the sorted list. Numerical experiments with several structures here, such as binary search trees[5] and various threads[6] have concluded that binary search trees have performed best.

As mentioned in the previous section, experimentation has shown that it is not necessary to keep and transfer from a predecessor node to its descendents any information regarding the relaxed solution. As it turns out, the procedure *Initialize_Simplex* typically gives initial solutions (to the relaxed problem) that are not worse than those given by the predecessor's solution. In addition, keeping the simplex solutions requires additional memory.

A complete pseudo-code for the branch-and-bound algorithm can be outlined as follows:

The Software Package

A software package for solving the optimal vessel loading problem was developed to run on microcomputers in the IBM/DOS environment. The package was programmed using the Clarion Professional Developer[7] and Borland's Turbo C compiler.[8] Clarion was used to create and manage the data base operations and to build a sophisticated user interface, while the C code was used to implement the computation-intensive optimization routines. The integration of the C code into the Clarion environment proved to be smooth and simple.

While the entire implementation could be achieved in a conventional programming language such as C, the use of a development package cut development time dramatically. Before settling on Clarion, three alternative development environments were tested: Paradox 2.0, R:Base/DOS, and Advanced Revelation 1.0. Paradox and R:Base/DOS were easy to use but were limited in the flexibility of their generated user-interfaces and were slow. Clarion and Advanced Revelation were comparable in their efficiency and sophistication.

The coding of I/O and data-base management usually accounts for a large share of the development time of any software package, but, by using a development package such as Clarion, this overhead can be greatly

```
Procedure Branch_ and_Bound;
Begin
        Solve the Linear Program by using the simplex method;
        If (all integrality constraints are satisfied) then
                Return; (* The solution is optimal!*)
        Else;
                Put the solution in the candidate list and point to it as the top node;
        End_if;
        Set incumbent by using Initial_Incumbent;
        While (candidate list is not empty) do
                Let the top node be current_node;
                        Delete current_node from candidate list and update list;
                        Branch on current_node by setting one of the fractional variables to 0 and 1,
                creating two descendents;
                        For each descendent i do
                                Modify Z, V, W given the variables already determined;
                                Relax integrality on the free variables and solve by the simplex method;
                                If not fathomed, then
                                        Tag the next variable to branch on;
                                        If (i is the first descendent not fathomed) then
                                                Reuse its ancestor location;
                                        Else
                                                Create a new node location for i
                                        End_if;
                                        Add i to the candidate list and update list;
                                Else
                                        If (solution is integer) and (current objective > incumbent value)
                                then
                                                incumbent value = current objective;
                                                incumbent solution = current solution;
                                                fathom all nodes with objective ≤ incumbent value in candidate
                                                list;
                                        End_if;
                                End_if;
                        End_for;
                End_while;
Optimal solution = incumbent solution;
End Branch_and_bound.
```

reduced. In fact, the ease and speed of programming in any of the special purpose environments blurs the line between software design, prototyping and final code production. Development becomes a single, continuous effort and takes a very short time. The generation of user screens, complete with windows, menus, and context-sensitive help is a matter of days. Similarly, report generation, processing and editing of data, and printer interface become almost trivial to code.

When using a code generator such as Clarion, the software architecture is predetermined to a large degree. In our model, data is stored in "tables," each of which is a disk file. The model was designed with two types of tables, one for scenarios and the other for underlying data. Both types of tables can be edited by form entries, or viewed through generated screen and hard copy report. The underlying tables hold information which rarely changes, such as distances, ship class specifications, port cost and operating parameters, etc.

A scenario is a set of three scenario tables. The first one contains information on bookings, where each record holds information on a booking's availability, weight, volume (cube), profit (FIO*), origin port, destination port, divisibility, shipper, and consignee, in addition to describing its exact contents and identification number. A second table contains information on the vessel to be considered in the scenario, its type, fuel efficiency and the price of fuel. The third scenario table includes the routing of the vessel under consideration. It lists the ports to be visited and the times of arrival and departure for each port. Each scenario is kept on disk in its own directory. To use the software, a scenario must be loaded into the workspace, where it can be modified, viewed, or optimized. Scenarios can be read, saved, or deleted from within the software package.

Case Study

To test the model in a realistic setting, data from a chartered vessel operation between Brazil and U.S. gulf ports was used. To mask the characteristics of the actual shipment and to generate enough data for a variety of cases, the original data were used only to estimate distribution functions parameters. These distributions of shipment size and shipment density were then used to generate simulated shipment data. Similarly, the vessel characteristics shown here are similar but not identical to the vessel characteristics used in the actual operation.

*FIO stands for "Free-In-Out" meaning the profit on the ocean line-haul portion of the voyage only, excluding any port or loading/unloading charge at either origin or destination.

Now consider a case where 42 possible shipments are available in three Brazilian ports (Rio de Janeiro, Santos, and Sao Paulo). These shipments are destined to two U.S. ports (New Orleans, Louisiana and Mobile, Alabama). The origin-destination distribution of the number of shipments is the following:

Table 1. Shipment Origin-Destination Matrix

To:	New Orleans	Mobile
From:		
Rio	15	0
Santos	17	0
Sao Paulo	0	10

The weights for each shipment were drawn from a normal distribution with a mean of 500 metric tons (MT) and a standard deviation of 170 MT. Shipment densities were drawn from a normal distribution with a mean of 2.0 metric tons per cubic meter (MT/M3) and a standard deviation of 0.9 MT/M3. The revenue for each shipment was drawn from a uniform distribution between $20/MT and $60/MT. (This is "Free-In-Out" or FIO revenue, including only the price of the ocean voyage portion of each shipment.) Using 0.50 probability, half the shipments were assumed to be indivisible while the other half was assumed to be divisible if the need arose. Port entry fees and other relevant costs were accounted for at a realistic level.

Given these shipments, the booking agent had to choose between two possible routings: either visit Rio de Janeiro and Santos only, taking whatever cargo is most profitable to New Orleans (scenario A), or visiting all ports, including Sao Paulo and Mobile (senario B). The available vessel had a net weight capacity of 14,250 MT and a net cubic capacity of 13,300 M3. The ship consumes bunker fuel at a rate of 20 MT/day at 12 knots and 36 MT/day at the maximum speed of 14 knots (this fuel costs $100/MT). The ship also consumes two tons of diesel fuel for every day in operation (this fuel costs $160/MT.

Tables 2 and 3 below depict the model's summary report screen for the two routings involved. These tables show the revenues, costs and profits for visiting only three ports and for visiting all five ports, respectively.

Note that the revenue for going to five ports ($670,195) is significantly higher than the revenue associated with the three port trip ($596,816) even

Table 2. Output Screen for the Three Port Scenario

Scenario	June			Port stops	3
Vessel	Freedom			Port costs	$55,000
				Route length	6,020 NM
	Daily charter	$7,000			
	Ballast bonus	$40,000			
	Insurance	$10,000			
			Pre-booked FIO		0
Pre-booked 0			Opt. FIO		$596,816
Available to optimizer		32			
Booked by optimization		28	Total FIO		$596,816
Speed: 12 knots				Fuel cost	$42,000
	Duration-travel	21 days		Diesel cost	$11,200
	Duration - in port	14 days		Vessel charter	$245,000
	Fuel consumed	4200 MT		Total costs	$403,200
	Diesel consumed	70 MT		Net profit	$193,616

Press Esc to return from viewing this screen report.

Table 3. Output Screen for the Five Port Scenario

Scenario	June			Port stops	5
Vessel	Freedom			Port costs	$95,000
				Route length	6,486 NM
	Daily charter	$7,000			
	Ballast bonus	$40,000			
	Insurance	$10,000			
			Pre-booked FIO		0
Pre-booked 0			Opt. FIO		$670,195
Available to optimizer		42			
Booked by optimization		28	Total FIO		$670,195
Speed: 12 knots				Fuel cost	$46,000
	Duration-travel	23 days		Diesel cost	$13,760
	Duration - in port	23 days		Vessel charter	$322,000
	Fuel consumed	4600 MT		Total costs	$526,760
	Diesel consumed	86 MT		Net profit	$143,435

Press Esc to return from viewing this screen report.

though the number of shipments booked by the optimization is 28 in both cases. Naturally, the reason is that the optimization can choose a better set of shipments when presented with more options. Despite the higher FIO revenue, however, it is more profitable to visit only the three ports rather than the five, due to the higher cost involved in the latter trip.

In both cases shown here, the algorithm chose the lower speed (12 knots) over the higher speed due to the excessive fuel consumption at the higher speed (otherwise the cost with the higher speed would have shown in the report.)

Summary

This paper presents an algorithm for optimal loading of a transportation conveyance. The objective is to maximize the voyage's profit subject to both weight and volume constraints. The problem is formulated as a mixed integer program and the solution algorithm is based on a branch-and-bound methodology with a relaxation which is solved by a special-purpose LP algorithm. While using the standard branch-and-bound method, the algorithm developed here is unique in its efficient initialization of the relaxed problem. The benefits of this initialization are compounded in the search strategy.

The algorithm has been implemented in a software package. The package has been used to solve for the optimal loading of vessels operating between Brazil and the U.S. Gulf Coast. While the model solved the optimal loading problem given the vessel and the routing, it is useful in determining the vessel type as well as its routing. This can be accomplished through repeated applications of the same model in an interactive environment. Interestingly, the use of microcomputers in an interactive environment places a renewed premium on algorithmic efficiency. This arises from the desire to provide immediate user feedback which is needed for continuous interactive sessions.

The basic optimization approach utilized here is applicable to many similar problems encountered in the transportation arena. These problems are characterized by indivisible shipments and both volume and weight constraints.

Another approach to the problem at hand could have been to incorporate the routing decision (and the vessel type decision) within the optimization process. This would have created a more difficult integer program involving fixed charges and tour constraints. While this may be the subject of further research, the utility of such a tool is limited since it is much less likely to be accepted by the potential user. The reason is that it does not leave any decision control with the user, thus requiring a significant leap of faith on his part. In other words, the user may want more control over the "big" decisions such as vessel type and routing. This dichotomy of decision domain allocation between the computer and the user can also be generalized to many other situations.

Acknowledgements

The help of William Reck, Director of International Operations of LogiCorp, Inc., was instrumental in obtaining data and implementing the model described here.

References

1. Taha, H. A., *Operations Research, An Introduction,* MacMillan, 1982.

2. Wagner, H. and T. Whitin, "Dynamic Version of the Economic Lot Size Model," *Management Science 5,* 1958a.

3. Bradley, S. P., A. C. Hax and T. L. Magnanti, *Applied Mathematical Programming,* Addison-Wesley, Reading, Massachusetts, 1977.

4. Paradimitriou, C. H. and K. Steiglitz, *Combinatorial Optimization, Algorithms and Complexity,* Prentice-Hall, 1982.

5. Horowitz, E. and S. Sahni, *Fundamentals of Computer Algorithms,* Computer Science Press, 1978.

6. Aho, A. V., Hopcroft and J. D. Ullman, *Data Structures and Algorithms,* Addison-Wesley, Reading, Massachusetts, 1983.

7. Clarion Corporation, *Clarion Professional Development Reference Manual,* Pompano Beach, Florida, 1988.

8. Borland International Corp., Turbo C, Version 1.0, 1987.

About the Authors

Dr. Yosef Sheffi is a professor of Civil Engineering at M.I.T., Cambridge, Massachusetts. He is heading both the transportation systems division and the program in engineering systems and computation there. His areas of research include logistics management, carrier operations, pricing of movements, and other aspects of freight transportation. He has developed many computerized decision-support systems which are used by major shippers, LTL and TL truck-lines, railroads and steamship lines. Dr. Sheffi has served as a consultant to many leading national and international shippers and carriers on these issues. He is the author of a book on transportation network optimization and of over 50 refereed articles in scientific journals. Dr. Sheffi obtained his undergraduate engineering degree from the Technion in Israel (1975) and both science major (1977) and doctorate (1978) from MIT. He has been teaching at MIT since 1977.

Benjamin Teitelbaum is a senior in the department of mathematics at the Massachusetts Institute of Technology. He is interested in network algorithms as well as questions of transportation policy. He is planning on continuing with graduate studies at MIT.

Shangyao Yan is currently a graduate student in the transportation division of the department of civil engineering at MIT. He received a master's degree in civil engineering from Taiwan University; there, he focused on traffic engineering. He is interested in applying mathematical programming, especially network optimization, to transportation logistics analysis.