## Ethanol Transport Model

We present an optimization model to determine over a repeated time horizon the configuration of leased and purchased cars that will provide the least capital and operating costs given a forecast for a decision period and a set of realized costs due to actual demands in a recourse stage occurring after the procurement decision. It consists of three nested parts: a Scheduling Model, a Provisioning Model, and a Horizon Model. We will explain the timing in terms of our case; Scheduling is stochastic and run for an operating cycle, in our case two weeks; Provisioning is run for each decision cycle, for us a quarter of 3 months, comprising 6 Scheduling cycles; and the Horizon model is run to plan over a horizon of multiple quarters; in our case the horizon is 5 decision cycles, or 15 months, since this is the term of a typical railcar lease.

In the Scheduling Model, we use a linear (integer) program to schedule the available cars into unit trains to meet realized demand during the scheduling cycle, which reflects the time required for the cars to go to the destination and return to the base to be used again. For our example this time is two weeks. The objective function calculates the operating cost of assigning the proper number of cars in unit trains to meet demand. Costs are measured per car per trip, and include maintenance, operating expenses, a risk factor reflecting the probability and attendant costs that a car may not finish the trip, inspection costs per trip, and car movement charges. Other costs in the objective are a charge per car per trip for idle cars (not used in this period because they cannot be put in a unit train), and a cost, higher than operating costs, for outsourcing a carload for one trip, incurred if demand cannot fit into the unit trains. We assume demand cannot be backlogged, so outsourcing transport allows us to fill it exactly.

There are several constraints. Unit trains must contain between a minimum and maximum number of cars. If demand cannot be fit into the unit trains, it must be outsourced. We cannot load more cars than are available. Available cars are those in our leased pool, and those in our purchased pool, less those that have been subleased for this time period (in Figure 1, the blue parts only). There is a cost associated with idle cars; storage space, insurance, maintenance, and security, as well as tracking.

Scheduling is run multiple times over all possible demands from 10 to 150, so we can obtain the expected cost over a distribution of demand. For our example we presumed demand is Poisson distributed with mean calculated from the forecast by weighting each operating period by an appropriate ‘seasonal’ factor; experience shows that realized demand does not occur evenly across the six two-week periods in our decision period of 3 months. We truncated the Poisson distributions at a minimum of 10 carloads and maximum of 150 in a two week cycle. The output from the Scheduling model is therefore the expected value of the recourse cost during the scheduling cycle, calculated as the sum of the cost from each run at a specific potential demand weighted by its appropriate Poisson likelihood and summing. Similarly we calculate the weighted average number and costs of idle and outsourced cars. It would also be possible to produce exact schedules for unit train cars, but this is not necessary for the overall goal of determining the least cost portfolio of leased and purchased cars.

The Provisioning Model uses a simple integer program to determine the number of leased and purchased cars to make available at the start of a three month provisioning period. Leased and purchased cars are acquired in different provisioning periods, and so have different acquisition costs. Thus the pools are broken into sub-pools of identical cost per car. In addition, subleasing may occur only at the start of a provisioning period; however there may be cars subleased at different rates. The objective includes the fixed costs per car of the leased, purchased, and (negative, since it is revenue) subleased cars.

The model constraints include assuring that the cars in the sub-pools add to the total number of cars of each type; the subleased cars in each sub-pool do not exceed the number of leased plus purchased cars in the sub-pool; and the number of cars must be within a lower and an upper bound. The minimum constraint can be initialized at the minimum number of cars in a unit train, since there is no point in having cars unless we can make at least one unit train; the maximum constraint can be initialized at a number large enough to at least cover the largest expected demand in any single two-week period.

The minimum constraint is adjusted as we cycle through the Scheduling and Provisioning models to search for the minimum cost provisioning configuration.

The model algorithm for a single provisioning (three-month) decision period is then as follows:

1. Initialize with the current number of leased, purchased, and subleased cars in each sub-pool.
2. Obtain the Forecast of demand over the three month decision period.
3. Determine demand probability distribution parameters for 6 operating cycles, proportioning the forecasted decision period demand by a seasonal factor for each cycle.
4. Repeat
   1. For each operating cycle:
      1. Solve Scheduling for each possible demand state.
      2. Average costs over possible demand states to obtain expected recourse cost.
      3. Determine average number of idle cars and outsourced cars.
      4. Determine average idle cost and outsource cost.
   2. Sum expected recourse costs, idle costs, outsource costs, idle cars, and outsourced cars over operating cycles in decision cycle
   3. Adjust inputs to Provisioning by changing total number of cars. (1)
   4. Solve Provisioning for leased, subleased and purchased cars and provisioning cost including the expected recourse costs of Scheduling.
   5. Compare average idle cost to average outsource cost.
      1. If idle cost > outsource cost, (1) reduce total number of cars.
      2. If outsource cost >= idle cost, (1) increase total number of cars.
   6. If direction of inequality has changed once or more, update minimum expected recourse costs, idle costs, outsource costs, idle cars, and outsourced cars, and minimum provisioning cost, and minimum number of leased, purchased, and subleased cars. (2)
   7. If direction of inequality has changed twice, stop repeating.
5. Report minimum expected recourse costs, idle costs, outsource costs, idle cars, and outsourced cars, and minimum provisioning cost, and minimum number of leased, purchased, and subleased cars.

Notes:

1. Search on the number of cars to find the optimum number. Increase number of cars if idle cost < outsource cost, and decrease the number of cars if outsource costs>= idle costs.
2. Since the sum of idle costs and outsource costs is a convex function, there is a unique number of cars that minimizes the sum of outsource costs and idle costs, or balances outsource cost against idle cost as well as possible. Therefore we search till the direction switches, then search back, recording minimum value. The number of cars yielding lowest total cost is then optimal for the decision cycle. Once the direction switches twice we know the minimum is trapped between the switch points.

Output is a provisioning policy (number of leased, purchased, and subleased cars) for the provisioning period, and its expected cost. With this in hand the user can easily calculate the adjustments to the prior period provisioning to achieve it. The routine above is run for each provisioning decision period of three months, before the start of the period, and after the decision period forecast is available.

Next we embed the provisioning period problem in a multi-period model for a horizon of five periods. 15 months is chosen in our example because that is the standard length of lease. Other horizons could be studied and would obtain different results. Multi-period problems are sensitive to the ‘edge’ effects created by initial conditions and target status at the end of the horizon.

This Horizon Model allows determination of when to change the number of cars available, and by how much of each type. Figure 1 shows the possible changes to the leased, purchased, and subleased pools.

Figure 1 Dynamics of railcar portfolio adjustment

Purchased

Cars (P)

Subleased

Cars

Leased

Cars (L)

New/ add-on Lease (1)

New Purchase

(2)

Recall from sublease (6L and 6P)

Sublease (3L)

Cancel Lease (4)

Sell (5)

Sublease (3P)

Railcars fall into one of three states: In figure 1 we show how the state may change. Cars are either leased for a period of 15 months, purchased, or the firm has subleased them to another party for a period, assumed to be 6 months (blue and orange inventory boxes). The firm’s portfolio of cars for a given Quarter (90 days) consists of the leased cars and purchased cars that are not subleased to someone else. The subleased cars generate revenue, though they do have some operating costs, calculated in the Recourse model. To add cars, the firm can purchase cars (arrow 2), or newly lease cars or add to an existing lease (arrow 1); these are red (cost money). If there are cars already subleased, they can be recalled from subleases at the appropriate time (arrows 6, also red, since the cash flow will be negative). To reduce the portfolio of cars, the firm can cancel the lease on a car (arrow 4, green, saves money) or sell the car (arrow 5, also green). The firm can also reduce the number of available cars without losing control of them for later by subleasing them (arrows 3, green because net cash flow will probably be positive). In the figure, the Purchased and Leased car inventories are shown with two colored segments, since the subleased cars come from either the Leased inventory or the Purchased inventory. The orange Subleased inventory is identical with the union of the two orange triangles inside the Leased and Purchased inventories. It represents a group of cars which are partially available, a backup inventory of controlled cars which could be pressed into service later, but in the meantime are generating revenue. We have assumed the firm writes only subleases for 6 months, so that cars committed could be available six months later for transport. This safety inventory is more productive than simply idling the cars; but the savings comes with a downside that they are only available 6 months from inception.

Here is my current thinking on this problem.

The horizon problem bears some resemblance to lot sizing in manufacturing. There are varying needs in each provisioning period. A difference is that the inventory of cars is not consumed, but is available for use the next horizon period. And if we have too many, it is costly, just as having too few is costly.

We wish to determine at what points over the horizon to modify the portfolio of cars by adding or subtracting cars. Adds can be made by new purchases, adding a car to a lease, or recalling a leased or purchased car from a sublease, if there are any eligible. Subtracts can happen through disposing of a purchased car by sale, disposing of a leased car by taking it off the lease, or by subleasing a purchased or leased car to another firm. These transactions are accompanied by several costs. A *major setup cost* is a processing cost for any transaction of any type regardless of size, which includes cost of finding a second party, completing an agreement, and processing the order. A *minor setup cost* is a form of processing cost that is particular to the type of transaction, but otherwise occurs once for each transaction, not per car. Examples are cleaning and preparation costs, the cost of moving or shunting cars to make them available for the new owner, and any maintenance, inspection, or risk associated costs particular to the batch of cars. There is a minor setup cost for each of the 8 different add or subtract actions.

The *carrying cost* of a car consists of two parts, particular to the type of car; the capital cost, and the expected cost per unit of idle cars. The last two figures are outputs available from the Scheduling model. The capital cost is an input to the Provisioning model; for leased and purchased cars it is a cost, for subleased cars it is the net of the sublease revenue and the lease or purchase cost, and may be negative or positive.

We draw from the lot sizing literature on replenishment in the face of irregular or somewhat lumpy demand. In this paper we compare three possible heuristics for determining when to add cars or reduce the number of cars by the means in the figure. We assume the model is implemented in a rolling fashion, and only the horizons up to the first adjustment are actually used. Thereafter the model is run again with the new forecasts and a new plan obtained.

1 Level Strategy (EOQ): adjust car quantities before the first provisioning period, to a level equal to the EOQ based on average demand over the horizon periods. Do not change thereafter.

2 Chase Strategy: adjust car quantities before each provisioning period to the optimum based on the forecasted expected demand for the provisioning period.

3 Silver-Meal Strategy: use a Silver-Meal type heuristic to determine which provisioning period to make an adjustment. Restart the planning process when the adjustment has been made.

A Level strategy has the advantage of requiring only one major and minor setup cost expenditure. However the number of cars chosen will be suboptimal in most periods, leading to higher operating costs, including recourse costs. It takes care of the planning for the entire horizon. The decision maker could choose to rerun if there is a substantial change in forecast demand as time evolves, but this risks ‘nervousness’ which forces more frequent adjustments, losing the value of the strategy.

A Chase strategy insures that in each provisioning period we use the number of cars that best balances the costs of idle cars and outsourced cars, and close to lowest provisioning period costs. However, it requires major and minor setup costs in each provisioning period. Since adjustment will be made each provisioning period, nervousness is not a problem and the model can be rerun each period with new forecasts.

An Average Cost Per Period (ACPP) or Silver-Meal strategy chooses a replenishment schedule to minimize the average cost per period of the sum of the setup costs (major and minor, particular to each item) and the carrying costs of having extra (idle) cars available for multiple periods. Such an objective is locally concave so that there is likely a number of periods which minimizes the cost. The Silver-Meal heuristic shows a very good record of practical applicability in inventory theory. In our horizon, it will require only one major/minor setup because once we reach the period when the adjustment is made we rerun the model with new forecast data into the future. It combines the advantage of the level strategy with one setup with an attempt to identify the best period to make an adjustment, rather than always at the start.

We computed the three suggested models in a spreadsheet once the Scheduling and Provisioning algorithm results for each of the horizon periods is complete. Data needed are available at the end of each provisioning period run.

## An Example

To test the model we ran a small example for 5 provisioning periods as described above.

### Parameters

For the Scheduling Model:

**param** Mmax := 50 ; #max length of a unit train

**param** Mmin := 20 ; #min length of a unit train

**param** Nmax := 5 ; #set >= largest E\_demand/Mmin

**param** Nmin := 0 ; #set >= E\_demand/Mmax

#costs of car types

**param** CL := 10 ; #lease cost per trip

**param** CP := 9 ; #purchase cost per trip

**param** CO := 30 ; #outsource cost per trip

**param** CQ := 50 ; #costs of unused car

**param** CY := 1 1 2 2 3 3 4 4 5 5 ; #ordering cost for trains, ascending order and small enough to not affect optimum much

For the Provisioning model:

**param** NLpool := 2; #number of lease pools (diffrerent costs)

**param** NPpool := 2; #number of purchase pools (different costs)

**param** NSpool := 2; #number of sublease pools (different costs)

**set** LPOOL := 1 2 ; #index for lease car pools

**set** PPOOL := 1 2 ; #index for purchased car pools

**set** SPOOL := 1 2 ; #index for subleased car pools

**param** NQ := 3 ; #not currently used

**set** PDS := 1 2 3 4 5 6; #number of periods of input from Scheduling

**param** KL := 1 300 2 240 ; # monthly ownership costs of lease pool

**param** KP := 1 600 2 650 ; # monthly ownership costs of purchase pool

**param** RS := 1 125 2 200 ; #this one will be net benefit from sublease

AMPL Models appear below in Appendices A and B.

We computed the three strategy outcomes. To do this required three runs of the Logic; each selected a different start bound and switch point for the number of cars. The logic is documented in Appendix C.

The ACPP heuristic selected period 3 to adjust in, to 42 cars. The EOQ level for Chase was 48 cars. ACPP gave the lowest cost. Chase performed worst in cost, though holding costs were of course zero, showing the issue with and Level was a bit above the ACPP. Table 1 shows the results.

Another important decision factor is fill rate and idle rate: Fill rate (technically a fill failure rate, 1 – the fill rate) is the expected number of carloads outsourced in a scheduling period (per trip) divided by the number of cars available for use. Idle rate is the expected number of cars idle for a trip divided by the number of cars available. These are reported as percentages in Table 1 for each strategy. Note that fill and idle rate themselves, while costing money, are less important than the variation, reflected in the standard deviation, and the maximum number. Wide variation increases the risk; the maximum percent of idle cars is reflected in the need for storage space for idle cars which might not be used all the time but must be allocated anyway, a cost not captured in the model; maximum outsourced rate reflects the variation in the number of carloads that must be negotiated each period. A large percentage of outsourcing places a burden on purchasing which must negotiate a substantial number of special deals on an ad hoc basis, a burden again hard to capture in the model since it is spread over a stochastic requirement.

In this dimension Chase will perform best, since it tracks demand specifically to keep the outsourced and idle costs balanced, while costing more in setup costs, and therefore serves as a best achievable benchmark. The example shows that we expect average idle and fill rates of 12% and 25% respectively, and maximum idle and fill rates of about 17% and 33%. Variation in idle and fill rates, measured by standard deviation of the percentages, are only 2% and 4% for Chase.

Level performs worst relative to Chase in each dimension. In each measure ACPP performs better than Level, except for average idle rate, where the difference is less than one percent, but in average outsource rate it actually beats Level by almost 7%. Thus in terms of fill and idle rates, ACPP seems marginally superior to Level, though both are somewhat worse than Chase. The largest advantage to ACPP seems to be in the Maximum line. Figure 2 helps us interpret the idle and fill rate analysis.

Table 1: Three decision strategies for provisioning ethanol cars. Source: author calculations.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **STRATEGY** | **LEVEL** | | **CHASE** | | **ACPP (SM3)** | |
| Worksheet | Level Strategy (EOQ) | | Chase Strategy | | SilverMeal 3 Strategy | |
| **Order** | **48** | | **65,49,41,33,53** | | **42** | |
| **In Period** | **1** | | **1,2,3,4,5** | | **3** | |
| S&O Costs | $ 87,630.28 | | $ 86,175.64 | | $ 94,126.37 | |
| Hold Costs | $ 29,500.00 | | $ - | | $ 18,000.00 | |
| Setup Costs | $ 11,700.00 | | $ 59,150.00 | | $ 11,800.00 | |
| **Subtotal** | **$ 128,830.28** | | **$ 145,325.64** | | **$ 123,926.37** | |
|  |  |  |  |  |  |  |
| **F&I Rates** | **Idle** | **Outsource** | **Idle** | **Outsource** | **Idle** | **Outsource** |
| Average | 14.24% | 27.08% | 12.93% | 25.38% | 15.12% | 20.48% |
| Stdev | 10.20% | 14.20% | 2.15% | 4.24% | 7.70% | 13.36% |
| Max | 31.94% | 52.08% | 17.17% | 33.33% | 26.81% | 44.84% |

Figure 2: Idle and outsource rate comparison. Source: authors.

### Analysis

These results are not unexpected. One would expect a Silver-Meal type heuristic to perform better than an EOQ strategy, from the lot sizing considerations. Following the lot sizing analogy, the lumpier the demand, the better ACPP should perform relative to Level; conversely the more stable demand is, the better Level should perform.

The method does not deal yet with some issues assisting in flexibility of response to varying demand, which might also result in more cost or helping to defray the costs of making cars available.

* Is it prudent to keep a certain number of purchased cars as a buffer against the possibility that cars are difficult or very expensive to sublease when we need them? My suggestion here as a first cut is to purchase enough cars to make one unit train. For the example this is a number between 20 and 50 cars. This requirement can easily be implemented in our model as a lower bound on the number of purchased cars on hand in the Provisioning Model. At present we do not have a good choice. If costs of purchasing relative to leasing are as at present, the minimum would always be selected. One could likewise stipulate an upper bound on purchased cars, which would constrain the amount of capital tied up, and would act like a budget constraint.
* Is there a right time to sublease purchased or leased cars, or recall from sublease. My suggestion here is to start with the ACPP solution and determine the gaps between required and available cars in each period the acquisition is made to cover. One could define a decision rule as follows:
  + Sublease cars that are not needed in expectation for hS or more horizon periods.
  + If there are cars on sublease, recall the cars needed in expectation within hR or fewer periods.

Choice of optimal hS , hR could be established by multiple runs; we do not currently have an algorithm for making this choice. In our example when the adjustment is only 3 horizon periods, one can only choose a value of 1 or 2, so the search is limited, as it always will be. We suggest that depending on the volatility of demand, it might be prudent to require that all subleases allow recall after one 3-month period. There are substantial costs associated with subleasing or recalling cars, since they must be cleaned and maintained for the change to a new use. Ethanol tank cars can be used for other materials.

## North Dakota Case Study

We now proceed to investigate a real case involving a North Dakota ethanol producer who ships each two weeks by rail to refineries in the Far West and elsewhere.

## Notes on the Silver-Meal Model

## The Silver-Meal model assumes we choose to replenish up to a particular period h in the overall horizon from 1 to H. In this period the replenishment cost is the sum of the setup costs (occurring in period 1) and the carrying costs (occurring in periods t in 1 through h). Transaction types are denoted by b. B\_b is a two valued variable +1 for an add and -1 for a subtract; Y\_bt is a binary variable indicating whether a particular transaction is used or not in a period.

RC(h) = A + sum\_{b} a\_b + sum\_ {t in 1 .. h} sum\_{b} Delta\_bh B\_b Y\_bh (ECQ/EQ + K\_b)

The objective we wish to minimize is then

RCUT(h) = RC(h) / h

The Silver-Meal heuristic proceeds by

Let h := 1.

Let Last\_RCUT := Infinity

For {b in B} Let Last\_Delta\_b := 0

Repeat {

Calculate RCUT(h)

If RCUT(h) > Last\_RCUT then break; #RCUT should be descending with increasing h

If h := H then break;

Let Last\_RCUT := RCUT(h);

For { b in B } let Last\_Delta\_b := Delta\_bh

Let h := h + 1;

}

If RCUT(h) <= Last\_RCUT then

Let h\* := h

Let RCUT(h\*) := RCUT(h)

For {b in B} Let Delta\_b\* := Delta\_bh

;

If RCUT(h) > Last\_RCUT then

Let h\* := h-1

Let RCUT(h\*) := Last\_RCUT(h-1)

For {b in B} Let Delta\_b\* := Last\_Delta\_b

;

Since RCUT is concave, and setup costs are being spread over more periods, RCUT will decline till the minimum is past, and the optimal number of periods is on the low side of the upturn.

The result should tell us that we should make the changes in portfolio allocation in period h\*, and the Delta\_b\* will be the changes to make; the cost will be RCUT(h\*). Once h\* is determined, the decision is frozen for h\* periods, the changes Delta\_b\* are made, and the decision is reevaluated for H future periods when the change ‘expires’.

## Appendices

## A Scheduling Model and Data

### Model

Model file for Scheduling Subproblem

/\* MODEL FOR SUBPROBLEM (RECOURSE)

Ethanol Transport Model Simple version

A bin packing problem with constraints. \*/

#function gsl\_cdf\_poisson\_P;

set TRAINS;

param F\_demand >= 0 integer; #Forecasted demand in carloads

param E\_demand >= 0 integer; #Realized demand in carloads

param Mmax >=0 integer; #max length of a unit train

param Mmin >= 0 <=Mmax, integer; #min length of a unit train

param Nmax integer >=0 ; #max no of unit trains ;

param Nmin integer >=0 ; #minimum no of unit trains

#given number of cars

param L ;

param P ;

param S ; #starting no of L, P, S cars

#costs of car types

param CL >= 0 ;

param CP >=0 ;

param CO >= 0 ;

param CS >=0; #costs of car types

param CQ >=0; #costs of unused car

param CY {TRAINS} >=0; #ordering cost for trains

var XL {i in TRAINS} >=0 integer; #no of L cars in ith train

var XP {i in TRAINS} >=0 integer; #no of P cars in ith train

var O >= 0 integer ;

var Q >=0 integer; #no outsourced, no not used

var N >=0 integer; # number of trains actually used

var Y {i in TRAINS} >=0 <=1 binary; # binary, 1 if used and 0 if not

minimize RCost: CL\*sum {i in TRAINS} XL[i] + CP\*sum {i in TRAINS} XP[i] + CO\*O + CQ\*Q + sum {i in TRAINS} CY[i]\*Y[i]

;

#costs to run L, P cars

#cost to outsource, idle

#fill order cost parameters

subject to notrains: N = sum {i in TRAINS} Y[i] ; #no of trains used

subject to mintrains: sum {i in TRAINS} Y[i] >= Nmin ; #at least Nmin trains

subject to maxtrains: sum {i in TRAINS} Y[i] <= Nmax ; #no more than Nmax trains

subject to trainmin {i in TRAINS}: (XL[i] + XP[i]) >= Mmin \* Y[i] ; #must have Mmin in a train

subject to trainmax {i in TRAINS}: (XL[i] + XP[i]) <= Mmax \* Y[i] ; #no more than Mmax in train

subject to useallcars: sum{i in TRAINS} ( XL[i] + XP[i] ) + Q = L + P - S ; #define Q the idle cars

subject to outsource: sum{i in TRAINS} ( XL[i] + XP[i] ) + O = E\_demand ; #define O the outsourced cars

subject to Lcars: sum{i in TRAINS} XL[i] <= L ; #cant exceed the no of L cars available

subject to Pcars: sum{i in TRAINS} XP[i] <= P ; #cant exceed the no of P cars

### Data

Data file for Scheduling Problem

/\* Ethanol Transport Data Simple version

\*/

#param F\_demand := 200 ; #Forecasted demand in carloads, not needed in recourse model

param E\_demand := 100 ; #Realized demand in carloads

param Mmax := 50 ; #max length of a unit train

param Mmin := 20 ; #min length of a unit train

param Nmax := 5 ; #E\_demand/Mmin ;

param Nmin := 0 ; #E\_demand/Mmax ;

set TRAINS := 1 2 3 4 5 ; #no of hypothetical trains

#given number of cars

param L ;

param P ;

param S ; #starting no of L, P, S cars

#costs of car types

param CL := 10 ;

param CP := 9 ;

param CO := 30 ;

#costs of car types

param CQ := 50 ; #costs of unused car (and setup cost of unit train not here)

param: CY :=

1 1

2 2

3 3

4 4

5 5

; #ordering cost for trains

## B Provisioning Model and Data

### Model

/\* MASTER MODEL FOR QUARTER

Ethanol Transport Period Model (Quarters) \*/

**param** NLpool >= 0 **integer** **default** 3; #size of price pools

**param** NPpool >= 0 **integer** **default** 3;

**param** NSpool >= 0 **integer** **default** 3;

**set** LPOOL ;

**set** PPOOL ;

**set** SPOOL ;

**set** PDS ;

**param** F ; # this is the forecasted demand for the 3 month period.

**param** NQ ; #number of stages in each period (6 semimonths in a quarter)

#param seas {PDS} >=0 <=1; #seasonal factors to spread F over the stages (months) in the period (qtr)

**param** EDQ {PDS} ;

**param** KL {LPOOL} >= 0 **integer**; #monthly ownership costs of each pool

**param** KP {PPOOL} >= 0 **integer**;

**param** RS {SPOOL} >= 0 **integer**; #this one will be a revenue from sublease

#param Mmax >=0 integer default 50; #max length of a unit train

#param Mmin >= 0 <=Mmax integer default 20; #min length of a unit train

#param Nmax integer >=0 default 5; #max no of unit trains ;

#param Nmin integer >=0 default 2; #minimum no of unit trains

**var** La {LPOOL} >=0 **integer**;

**var** Pa {PPOOL} >=0 **integer**;

**var** Sa {SPOOL} >=0 **integer**;

**var** Lop ;

**var** Pop ;

**var** Sop ;

**param** ERecourseCost ; # recourse costs from stages (months)

**minimize** Qcost: **sum** {i **in** LPOOL} KL[i]\*La[i] + **sum** {i **in** PPOOL} KP[i]\*Pa[i] - **sum** {i **in** SPOOL} RS[i]\*Sa[i]

+ ERecourseCost ;

**subject** **to** lcars: **sum**{i **in** LPOOL} La[i] = Lop ;

# the pools add up to LL, PP, SS

**subject** **to** pcars: **sum** {i **in** PPOOL} Pa[i] = Pop ;

**subject** **to** scars: **sum** {i **in** SPOOL} Sa[i] = Sop ;

**subject** **to** slim {i **in** LPOOL}: Sa[i] <= La[i] + Pa[i];

**subject** **to** cars: **sum** {i **in** LPOOL} La[i] + **sum** {i **in** PPOOL} Pa[i] - **sum** {i **in** SPOOL} Sa[i] >= 20;

**subject** **to** totcars: **sum** {i **in** LPOOL} La[i] + **sum** {i **in** PPOOL} Pa[i] + **sum** {i **in** SPOOL} Sa[i] >= F / **card**(PDS);

### Data

**param** NLpool := 2;

**param** NPpool := 2;

**param** NSpool := 2;

**set** LPOOL := 1 2 ;

**set** PPOOL := 1 2 ;

**set** SPOOL := 1 2 ;

**param** NQ := 3 ;

**set** PDS := 1 2 3 4 5 6;

**param** KL := 1 300 2 240 ; #monthly ownership costs of each pool

**param** KP := 1 600 2 650 ;

**param** RS := 1 125 2 200 ; #this one will be a revenue from sublease

#param F ;

#param seas := 1 0.25 2 0.40 3 0.35;

## C Logic for Five Horizon Provisioning Periods

### Level (EOQ)

#script for running HorizonModel and all others 3

load amplgsl.dll; #load gnu statistical library

function gsl\_cdf\_poisson\_P; #use poisson distribution

**option** solver cplex; #select solver cplex

**option** solver\_msg 0; #suppress messages

**model** EthanolTransportStageModel.mod; #invoke Scheduling Model

**data** EthanolTransportStageData.dat; #Scheduling Model Data

**model** ETQ.mod; #invoke Provisioning Model

**data** ETQ.dat; #Provisioning Model Data

**problem** Recourse: XL,XP,O,Q,N,Y,RCost,

notrains, mintrains, maxtrains, trainmin, trainmax, useallcars, outsource, Lcars, Pcars

; #define Scheduling Model as problem for reuse

**problem** Quarter: La, Pa, Sa, Lop, Pop, Sop, Qcost,

lcars, pcars, scars, slim, cars, totcars

; #define Provisioning Model as problem for reuse

**option** relax\_integrality 1; #don’t need integer solution since problem is unimodulqr

**param** seas {PDS} ; #seasonal factors for Scheduling periods

**let** seas[1] := .20 ; **let** seas[2] := .30 ; **let** seas[3] := .10 ;

**let** seas[4] := .15 ; **let** seas[5] := .10 ; **let** seas[6] := .15 ;

**param** Qcostold; **let** Qcostold := Infinity ; #not needed

**param** Qcostmin; **let** Qcostmin := Infinity ; #not needed

**param** Idlecostold ; **let** Idlecostold := Infinity ; #not needed

**param** Outsourcecostold ; **let** Outsourcecostold := Infinity ; #not needed

**param** Linit; **param** Pinit; **param** Sinit; #initial number of cars at horizon 0

**let** Linit := 69 ; **let** Pinit := 0 ; **let** Sinit := 5 ; #initial values

**param** LPBound; **param** SBound; #bounds for search using Provisioning Model

# parameters for selecting proper output

**param** down; **param** up; **param** down2 ; **param** up2 ; **param** downfirst ;

# define trip periods

**let** PDS := 1 .. 6 **by** 1;

# define horizons for long term planning

# horizon 0 is consistent with the EOQ model for not so lumpy demand

**set** HORIZON := { 0 .. 5 } ; #horizon 0 makes a run with a forecast of the average of horizon periods from 1 up

**set** CARS := { "Leased", "Purchased", "Subleased\_L", "Subleased\_P" } ; #car types for storing results

**set** CHANGES := { "al", "ap", "rl", "rp", "sl", "sp", "cl", "dp" } ; #types of transactions, not needed presently

**param** nh := **card**(HORIZON) ; #calculate number of horizons

**param** Forecast {HORIZON} ; #forecast

**param** Majorcost {HORIZON} ; #cost of any change, one time

**param** Minorcost {HORIZON, CARS} ; #cost of a change of a particular type

**param** Holdcost { HORIZON, CARS } ; #cost to hold onto a car per quarter

# data for horizon model

**data** ;

#these are the 3-month forecasts input by user

#period 0 is the average forecast

**param** Forecast :=

0 295

1 400

2 300

3 250

4 200

5 325 ;

#major setup costs for each horizon

**param** Majorcost :=

0 1000

1 1000

2 1100

3 1100

4 1200

5 1250

;

#minor setup costs for each horizon and car type

**param** Minorcost : Leased Purchased Subleased\_L Subleased\_P :=

0 10700 10500 8000 6000

1 10700 10500 8000 6000

2 10700 10500 8000 6000

3 10700 10500 8000 6000

4 10700 10500 8000 6000

5 10700 10500 8000 6000

;

#holding costs for an idle car

**param** Holdcost: Leased Purchased Subleased\_L Subleased\_P :=

0 500 700 300 500

1 500 700 300 500

2 500 700 300 500

3 500 700 300 500

4 500 700 300 500

5 500 700 300 500

;

**end** data ;

# F&I parameters for the horizon model, not used presently, user input

param Acceptable\_Idle\_rate default 0.10 ; #10%

param Acceptable\_Outsource\_rate default 0.10 ; #10%

#passed in from Provisioning for given horizon

param Expected\_Outsource { HORIZON } ; #this to be passed in

param Expected\_Idle {HORIZON} ; #this to be passed in

param Expected\_Outsource\_Cost {HORIZON} ; #to be passed in

param Expected\_Idle\_Cost {HORIZON} ; #to be passed in

param Nocars { HORIZON, CARS } ; # this might be passed in by quarter

param Combined\_Cost {HORIZON} ; #passed in by period

param Expected\_Recourse\_Cost {HORIZON} ; #passed in by period

#########################run some horizon periods

# scheduling sets and parameters for input and output

set EDR;

param Pr {EDR} ;

param PPr {EDR} ;

param PPr\_mean ;

param PPr\_trunc\_const ;

param edr\_obj {EDR} ;

param edr\_dist {EDR} ;

param Expected\_Cost {PDS} ;

param Expected\_Q {PDS}; param Expected\_O {PDS};

param ECQ {PDS} ; param ECO {PDS} ;

# scheduling result variables

param Recourse\_Costs ;

param Netmastercost ;

param Combinedcost ;

#Minimum result parameters for storage

param MinCombinedcost default Infinity ;

param MinLPB default 0; param MinSB default 0;

param MinL default 0 ; param MinP default 0 ; param MinS default 0 ;

param MinEQ {PDS} default 0 ; param MinEO {PDS} default 0 ;

param MinECQ {PDS} default 0 ; param MinEOQ {PDS} default 0 ;

param MinExpectedCost {PDS} default 0 ;

param MinIdleCars default 0 ;

param MinOutsourceCars default 0 ;

param MinIdleCosts default 0 ;

param MinOutsourceCosts default 0 ;

#Initial result parameters for storage

param FirstCombinedcost {HORIZON} default Infinity ;

param FirstLPB {HORIZON} default 0; param FirstSB {HORIZON} default 0;

param FirstL {HORIZON} default 0 ; param FirstP {HORIZON} default 0 ; param FirstS {HORIZON} default 0 ;

param FirstEQ {PDS} default 0 ; param FirstEO {PDS} default 0 ;

param FirstECQ {PDS} default 0 ; param FirstEOQ {PDS} default 0 ;

param FirstExpectedCost {PDS} default 0 ;

param FirstIdleCars {HORIZON} default 0 ;

param FirstOutsourceCars {HORIZON} default 0 ;

param FirstIdleCosts {HORIZON} default 0 ;

param FirstOutsourceCosts {HORIZON} default 0 ;

param FirstRecourse\_Costs {HORIZON} default 0 ;

#parameters for idle/outsource costs and cars

param Idle\_Costs ; param Outsource\_Costs ;

param Idle\_Cars ; param Outsource\_Cars ;

#Parameters for Scheduling run storage

param edr\_O {EDR};

param edr\_Q {EDR};

param edr\_CO {EDR} ; param edr\_CQ {EDR} ;

param edr\_XP {EDR,1..5}; param edr\_XL {EDR,1..5};

#minimum and maximum demands for calculating expectations

param Emin := 10 ; param Emax := 150 ; param Estep := 1 ;

param searchcount default 0 ; #counts the no of steps in search

# outer loop over all time periods in the HORIZON

for {h in HORIZON} {

printf "Forecast for QTR: %d = %d\n", h, Forecast[h];

let F := Forecast[h] ;

let Fc := F ;

# set up the starting point;

# for EOQ it is to always start with the initial value from h = 1

if ( h < 1 ) then { #this is for 0

let L := Linit ;

let P := Pinit ;

let S := Sinit ;

let LPBound := Linit + Pinit ;

let SBound := Sinit ;

}

else { #lets us start over with the result from horizon 0

let L := Nocars[ 0, 'Leased' ] ;

let P := Nocars[ 0, 'Purchased' ] ;

let S := Nocars[ 0, 'Subleased\_L' ] ;

# let L := MinL ;

# let P := MinP ;

# let S := MinS ;

let LPBound := L + P ;

let SBound := S ;

}

#set internal bounds in the Recourse Scheduling model

let LPB := LPBound ;

let SB := SBound ;

# control var for break point when optimum for period has been located.

let down := 0 ; let up := 0 ;

let down2 := 0 ; let up2 := 0 ;

let downfirst := 0 ; #control for reporting specific results

# initialize variables to hold optimum for period

let MinCombinedcost := Infinity ;

let MinLPB := 0 ; let MinSB := 0 ;

let MinL := 0 ; let MinP := 0 ; let MinS := 0 ;

let MinIdleCars := 0 ;

let MinOutsourceCars := 0 ;

let MinIdleCosts := 0 ;

let MinOutsourceCosts := 0 ;

# initialize variables to hold the initial for the period

let FirstCombinedcost[ h ] := Infinity ;

let FirstLPB [ h ] := 0 ; let FirstSB[ h ] := 0 ;

let FirstL [ h ] := 0 ; let FirstP [ h ] := 0 ; let FirstS [ h ] := 0 ;

let FirstIdleCars [ h ] := 0 ;

let FirstOutsourceCars [ h ] := 0 ;

let FirstIdleCosts [ h ] := 0 ;

let FirstOutsourceCosts [ h ] := 0 ;

let FirstRecourse\_Costs [ h ] := 0 ;

# initialize variables to hold some costs for the period, used for testing for the min

let Combinedcost := Infinity ;

let Netmastercost := Infinity ;

# initialize variables to hold optimum values which are averages for each schedule trip

for {i in PDS} {

let MinEQ[i] := 0 ;

let MinEO[i] := 0 ;

let MinECQ[i] := 0 ;

let MinEOQ[i] := 0 ;

let MinExpectedCost[i] := 0 ;

}

# start algorithm for a specific horizon period

printf "BEGIN RUN for horizon %d with Forecast %d \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", h, Fc ;

# count the number of steps in the search, can be used to find a specific one

let searchcount := 0 ; # initialize it here

#major repeat of the subproblem followed by master

# the repeat loop searches for the optimal number of cars by adjusting the bounds

repeat {

let searchcount := searchcount + 1 ; #count the next step

printf "%20s \n", "Starting Scheduling Run" ;

# run Scheduling (Recourse) for all trip PDS (6) for all possible demands to get expected values

for {q in PDS} {

let EDQ[q] := F\*seas[q]; #this is the demand in the schedule trip

let EDR := Emin .. Emax by Estep ; #these are the possible demands to get the expectations

for {i in EDR} {

# set sample demand

let E\_demand := i;

# compute stochastic weights of each demand level, use appropriate gsl function

let PPr\_trunc\_const := gsl\_cdf\_poisson\_P(Emax,EDQ[q]) - gsl\_cdf\_poisson\_P(Emin-1,EDQ[q]) ;

let PPr[i] := ( gsl\_cdf\_poisson\_P(i,EDQ[q]) - gsl\_cdf\_poisson\_P(i-1,EDQ[q]) ) / PPr\_trunc\_const;

# scheduling; write result in scratchfile to speed up process.

solve Recourse > scratchfile.out;

# compute weighted values for sample demand

let edr\_obj[i] := RCost ;

let edr\_dist[i] := edr\_obj[i] \* PPr[i] ;

let edr\_O[i] := O\*PPr[i] ;

let edr\_Q[i] := Q\*PPr[i];

let edr\_CO[i] := CO\*edr\_O[i] ;

let edr\_CQ[i] := CQ\*edr\_Q[i] ;

# record the no of purchased and leased in each of the 5 categories needed for the master model

for {j in 1..5} {

let edr\_XP[i,j] := XP[j]; let edr\_XL[i,j] := XL[j];

}

}

# calculate some key expected values of cost for qth trip

#overall expected cost

let Expected\_Cost[q] := sum {i in EDR} edr\_dist[i];

# expected number outsourced (O) and idle (Q) cars

let Expected\_Q[q] := sum {i in EDR} edr\_Q[i] ;

let Expected\_O[q] := sum {i in EDR} edr\_O[i] ;

# expected cost of outsourced and idle cars

let ECQ[q] := sum {i in EDR} edr\_CQ[i] ;

let ECO[q] := sum {i in EDR} edr\_CO[i] ;

/\* print the results if we want to see it on the console.

printf "%5s %2d %20s %4d %4d %4d %9.2f %9.2f %9.2f %9.2f %10s %10.3f %3.2f\n",

"q=",q,"(L P S EQ EO ECQ ECO)=",L, P, S,

Expected\_Q[q],Expected\_O[q],ECQ[q],ECO[q],

"ERcost=",Expected\_Cost[q], seas[q] ;

\*/

} # end of loop over PDS q

# calculate key values to use later on in computing the best number for horizon

let Recourse\_Costs := sum {q in PDS} Expected\_Cost[q] ;

let Idle\_Cars := sum {q in PDS} Expected\_Q[q] ;

let Outsource\_Cars := sum {q in PDS} Expected\_O[q] ;

let Idle\_Costs := sum {q in PDS} ECQ[q] ;

let Outsource\_Costs := sum {q in PDS} ECO[q] ;

let Combinedcost := Netmastercost + Recourse\_Costs ;

# display some of these values in a nice line or two

# uncomment if you want to see these after each schedule model run

/\* printf "%15s: $%10.2f %15s: $%10.2f %15s: $%10.2f Min: $%10.2f\n", "Total Recourse cost",Recourse\_Costs, "LastMastercost", Netmastercost, "Combined cost", Combinedcost, MinCombinedcost ;

printf "LPS: %d %d %d IO: %d %d IOCost: $%8.2f $%8.2f\n", L,P,S,

Idle\_Cars, Outsource\_Cars, Idle\_Costs, Outsource\_Costs ;

\*/

/\* This logic is supposed to use logical variables down and downfirst to set the minimums found to date in the horizon.

inserting and down == 1 means record the min only when the bound is going down.

seems to work now.

Combinedcost <= MinCombinedcost when we are descending (most of the time), since the total costs tend to drop with less cars.

In this case the minimums re just at the smallest bound, so not useful to find optimum

\*/

# this is to capture the starting point costs to be able to

# get a difference later on to compare.

# only happens when first step of search is done

if ( searchcount == 2 ) then {

# we get here after we have run one step of the search both schedule and quarter.

# we capture the values we need for the future

let FirstCombinedcost [ h ] := Combinedcost ;

let FirstLPB [ h ] := LPB ; let FirstSB [ h ] := SB ;

let FirstL [ h ] := L ; let FirstP [ h ] := P ; let FirstS [ h ] := S ;

let FirstIdleCars [ h ] := Idle\_Cars ;

let FirstOutsourceCars [ h ] := Outsource\_Cars ;

let FirstIdleCosts [ h ] := Idle\_Costs ;

let FirstOutsourceCosts [ h ] := Outsource\_Costs ;

let FirstRecourse\_Costs [ h ] := Combinedcost - Netmastercost ;

# Now that it works the following is for diagnostic purposes and need not be printed

/\* printf "First: Combined Cost: $%10.2f LPS: %d %d %d Bounds: %d %d IOCars: %d %d IOCost: $%8.2f $%8.2f Flags: %d %d %d\n",

FirstCombinedcost, FirstL, FirstP, FirstS, FirstLPB, FirstSB,

FirstIdleCars, FirstOutsourceCars, FirstIdleCosts, FirstOutsourceCosts,

down, up, downfirst ;

\*/

}

if ( Combinedcost <= MinCombinedcost and down == 1 ) then {

# we dont get here till we have run one Quarter problem.

# that is because of the fact that it has to be less

let MinCombinedcost := Combinedcost ;

let MinLPB := LPB ; let MinSB := SB ;

let MinL := L ; let MinP := P ; let MinS := S ;

let MinIdleCars := Idle\_Cars ;

let MinOutsourceCars := Outsource\_Cars ;

let MinIdleCosts := Idle\_Costs ;

let MinOutsourceCosts := Outsource\_Costs ;

# Now that it works the following is for diagnostic purposes and need not be printed

/\* printf "Mins: Combined Cost: $%10.2f LPS: %d %d %d Bounds: %d %d IOCars: %d %d IOCost: $%8.2f $%8.2f Flags: %d %d %d\n",

MinCombinedcost, MinL, MinP, MinS, MinLPB, MinSB, MinIdleCars, MinOutsourceCars, MinIdleCosts, MinOutsourceCosts,

down, up, downfirst ;

\*/

}

# set inputs for Quarter master problem

let ERecourseCost := Recourse\_Costs ; # set the recourse cost for input, ERecourseCost is internal to Quarter problem

# set the bounds. the bounds will move up or down in the search, LPB, SB are internal to Quarter problem

# we also pass in the idle costs, outsource costs,

let LPB := LPBound ;

let SB := SBound ;

# summarize what is going in

printf "%30s %10.3f IC=%10.3f OC=%10.3f LPB=%10d SB=%10d\n",

"Starting Quarter (Master) Problem with recourse cost = ",

ERecourseCost, Idle\_Costs, Outsource\_Costs, LPB, SB ;

# quarter model computes the fixed costs using the previous recourse cost and the current bounds LPB, SB

solve Quarter; #solve Provisioning problem for this horizon

# calculate the portion that does not depend on the recourse costs, so we can calculate the post hoc combined cost.

let Netmastercost := Qcost-ERecourseCost ;

# report the outcomes

printf "%10s %10.2f %10s %10.2f %10s %5d %5d %5d Idle Costs: %10.3f Outsource Costs: %10.3f ", "Master cost=", Qcost,

"Netmastercost=",Netmastercost,"(Lop Pop Sop)=", Lop, Pop, Sop, Idle\_Costs, Outsource\_Costs;

# this section compares idle costs to outsource costs to decide whether to change bound up or down

# this is where the search direction gets set.

# also set whether the direction is up first or down first.

if Idle\_Costs > Outsource\_Costs then {

printf "%15s\n", "Idle > Out" ;

let LPBound := LPBound - 1 ;

let down := 1 ;

# if ( Netmastercost == Infinity and MinCombinedcost == Infinity ) then let downfirst := 1 ;

if ( down == 1 and up == 0 ) then let downfirst := 1 ; # means we went down first

if ( down == 1 and up == 1 ) then let down2 := 1 ; #happens when we reverse from an up cycle

if ( down2 == 1 ) then { #leave in for now, though diagnostic

printf "Switch back down2, $%10.2f %4d %4d %4d Flags: %4d %4d %4d\n", Netmastercost, Lop, Pop, Sop, down, up, downfirst ;

}

}

if Idle\_Costs <= Outsource\_Costs then {

printf "%15s\n", "Idle <= Out" ;

let LPBound := LPBound + 1 ;

let up := 1 ;

# if ( Netmastercost == Infinity and MinCombinedcost == Infinity ) then let downfirst := 0 ;

if ( down == 0 and up == 1 ) then let downfirst := 0 ; # means we went up first

if ( down == 1 and up == 1 ) then let up2 := 1 ; #happens when we reverse from a down cycle

if ( up2 == 1 ) then { #leave in for now, though diagnostic

printf "Switch back up2, $%10.2f %4d %4d %4d Flags: %4d %4d %4d\n", Netmastercost, Lop, Pop, Sop, down, up, downfirst ;

}

}

# this logic decides whether the search is over

if ( down2 == 1 and up2 == 1 and downfirst == 1 ) then {

printf "%15s %10.3f %10.3f %10.3f %10.3f %4d %4d %4d Flags: %4d %4d %4d\n", "Gonna break.",

Qcost, Qcostold, Idle\_Costs, Outsource\_Costs, Lop, Pop, Sop, down, up, downfirst ;

# not needed?

let Qcostold := Qcost ;

let L := Lop; let P := Pop; let S := Sop; #make sure to do it before the break for next step

# leave the repeat loop

break ;

}

if ( down2 == 1 and up2 == 1 and downfirst == 0 ) then {

printf "%15s %10.3f %10.3f %10.3f %10.3f %4d %4d %4d Flags: %4d %4d %4d\n", "Set mins to last before break.",

Qcost, Qcostold, Idle\_Costs, Outsource\_Costs, Lop, Pop, Sop, down, up, downfirst ;

# not needed?

let Qcostold := Qcost ;

# define inputs for rerun of scheduling recourse problem

let L := Lop; let P := Pop; let S := Sop; #make sure set for rerun before break, for next step

# define minimum bounds

let MinL := Lop ; let MinP := Pop ; let MinS := Sop ;

let MinLPB := LPB ; let MinSB := SB ;

# define the minimums for the calculation

let MinCombinedcost := Combinedcost ;

let MinIdleCars := Idle\_Cars ;

let MinOutsourceCars := Outsource\_Cars ;

let MinIdleCosts := Idle\_Costs ;

let MinOutsourceCosts := Outsource\_Costs ;

# leave the repeat loop

break ;

}

# define L, P, S for the next Scheduling pass to get the proper recourse costs

# this should happen if no break has occurred in the last 2 statements, ie.

# when down2 !=1 or up2 != 1

# since the breaks above handle all other cases of downfirst ;

let L := Lop; let P := Pop; let S := Sop; #?????

printf "Completed Schedule and Quarter for search step no: %d\n", searchcount ;

# let searchcount := searchcount + 1 ; # dont want it here, set at start of repeat

} # repeat end of the search for the minimum cost L, P, S

# rerun outside the repeat loop to get the 'final answer' with correct recourse cost

# finish off by rerunning the subproblem

# first print L, P, S for input to the schedule model

# these are diagnostic if it works

/\* printf "L Settings for rerunning subproblem 1: LPS: %d %d %d : Bounds: %d %d\n", L, P, S, LPB, SB ;

printf "Settings for rerunning subproblem 1 Min LPS: %d %d %d : Bounds: %d %d\n", MinL, MinP, MinS, MinLPB, MinSB ;

\*/

# logic to set the settings properly for the rerun, currently worked above for downfirst = 0

if ( downfirst == 1 ) then {

let L := MinL ;

let P := MinP ;

let S := MinS ;

let LPB := MinLPB ;

let SB := MinSB ;

}

# this may be diagnostic, but leave for now

printf "Adjusted settings for rerunning subproblem 1 (L,P,S): %d %d %d : Bounds: %d %d\n", L, P, S, LPB, SB ;

# rerun scheduling model 6 times and record the info as above

for {q in PDS} {

let EDQ[q] := F\*seas[q];

let PPr\_mean := EDQ[q] ;

let EDR := Emin .. Emax by Estep ;

for {i in EDR} {

let E\_demand := i;

solve Recourse > scratchfile.out;

let PPr\_trunc\_const := gsl\_cdf\_poisson\_P(Emax,PPr\_mean) - gsl\_cdf\_poisson\_P(Emin-1,PPr\_mean) ;

let PPr[i] := ( gsl\_cdf\_poisson\_P(i,PPr\_mean) - gsl\_cdf\_poisson\_P(i-1,PPr\_mean) ) / PPr\_trunc\_const;

let edr\_obj[i] := RCost ;

let edr\_dist[i] := edr\_obj[i] \* PPr[i] ;

let edr\_O[i] := O\*PPr[i] ;

let edr\_Q[i] := Q\*PPr[i];

let edr\_CO[i] := CO\*edr\_O[i] ;

let edr\_CQ[i] := CQ\*edr\_Q[i] ;

for {j in 1..5} {

let edr\_XP[i,j] := XP[j]; let edr\_XL[i,j] := XL[j];

}

}

let Expected\_Cost[q] := sum {i in EDR} edr\_dist[i] ;

let Expected\_Q[q] := sum {i in EDR} edr\_Q[i] ;

let Expected\_O[q] := sum {i in EDR} edr\_O[i] ;

let ECO[q] := CO\*Expected\_O[q] ;

let ECQ[q] := CQ\*Expected\_Q[q] ;

printf "%5s %2d %20s %4d %4d %4d %9.2f %9.2f %9.2f %9.2f %10s %10.3f\n",

"q=",q,"(L P S EQ EO ECQ ECO)=",L, P, S,

Expected\_Q[q],Expected\_O[q],ECQ[q],ECO[q],

"ERcost=",Expected\_Cost[q] ;

}

# display results of last scheduling model with accurate recourse costs

let Recourse\_Costs := sum {q in PDS} Expected\_Cost[q] ;

/\* this is diagnostic since it shows up in next printed line

printf "Rerun 1 Sum of expected recourse costs: %15s: $%10.2f\n", "Total Recourse cost",Recourse\_Costs ;

\*/

#report the summary answers

printf "\nFINAL RESULTS\n","" ;

printf "\nBeginning results for Horizon Period %d\n", h ;

# report the first output of search from the search start variables, for comparison later

printf "Combined Cost: $%10.2f LPS: %d %d %d Bounds: %d %d Recourse Cost: $%10.2f IOCars: %d %d IOCost: $%8.2f $%8.2f\n",

FirstCombinedcost[ h ], FirstL[ h ], FirstP[ h ], FirstS[ h ], FirstLPB[ h ], FirstSB[ h ],

FirstRecourse\_Costs[ h ], FirstIdleCars[ h ], FirstOutsourceCars[ h ], FirstIdleCosts[ h ], FirstOutsourceCosts[ h ] ;

printf "\nSummary results for Horizon Period %d\n", h ;

let Recourse\_Costs := sum {q in PDS} Expected\_Cost[q] ;

# report the best output from the min variables

printf "Combined Cost: $%10.2f LPS: %d %d %d Bounds: %d %d Recourse Cost: $%10.2f IOCars: %d %d IOCost: $%8.2f $%8.2f\n",

MinCombinedcost, MinL, MinP, MinS, MinLPB, MinSB,

Recourse\_Costs, MinIdleCars, MinOutsourceCars, MinIdleCosts, MinOutsourceCosts ;

/\* define the number of cars required for each horizon period and report out the data\*/

let Nocars[ h, 'Leased' ] := L ;

let Nocars[ h, 'Purchased' ] := P ;

let Nocars[ h, 'Subleased\_L' ] := S ;

let Nocars[ h, 'Subleased\_P' ] := 0 ;

/\* here are the costs generated by those configurations \*/

# needs possible revision

let Expected\_Outsource[ h ] := MinOutsourceCars ;

let Expected\_Idle[ h ] := MinIdleCars ;

let Expected\_Outsource\_Cost [ h ] := MinOutsourceCosts ;

let Expected\_Idle\_Cost [ h ] := MinIdleCosts ;

let Combined\_Cost [h] := MinCombinedcost ;

let Expected\_Recourse\_Cost [h] := Recourse\_Costs ;

printf "\nEND RUN for Period %d %s\n", h,"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" ;

} # end of horizon loop

# show some of input data for horizon optimization

printf "\nAcceptable Idle Rate: %4.2f\% Outsource Rate: %4.2f\%\n", 100\*Acceptable\_Idle\_rate, 100\*Acceptable\_Outsource\_rate ;

# report the summary of results we stored above in a nice format

printf "\nOPTIMAL RESULTS FOR HORIZONS\n","" ;

printf " h %8s %8s %8s %8s %12s %12s %8s %8s %8s %8s\n",

"LCars", "PCars", "SLCars", "SPCars","CombinedCost",

"ERecourseCost", "IdleCars","OutsCars", "IdleCost", "OutsCost" ;

for { h in HORIZON} {

printf "%3d %8d %8d %8d %8d %10.2f %10.2f %8d %8d %8.2f %8.2f\n", h,

Nocars[ h, 'Leased' ], Nocars[ h, 'Purchased' ], Nocars[ h, 'Subleased\_L' ], Nocars[ h, 'Subleased\_P' ],

Combined\_Cost [h], Expected\_Recourse\_Cost [h],

Expected\_Idle[h], Expected\_Outsource[h],

Expected\_Idle\_Cost[h], Expected\_Outsource\_Cost[h] ;

}

printf "INITIAL RESULTS FOR HORIZONS\n","" ;

printf " h %8s %8s %8s %8s %12s %12s %8s %8s %8s %8s %8s %8s\n",

"LCars", "PCars", "SCars", "", "CombinedCost",

"ERecourseCost", "IdleCars","OutsCars", "IdleCost", "OutsCost", "LPB", "SB" ;

for { h in HORIZON} {

printf "%3d %8d %8d %8d %8d %10.2f %10.2f %8d %8d %8.2f %8.2f\n", h,

FirstL[ h ], FirstP[ h ], FirstS[ h ], 0 ,

FirstCombinedcost[ h ], FirstRecourse\_Costs[ h ],

FirstIdleCars[ h ], FirstOutsourceCars[ h ],

FirstIdleCosts[ h ], FirstOutsourceCosts[ h ],

FirstLPB[ h ], FirstSB[ h ] ;

}

printf "END OF RESULTS FOR HORIZONS\n", "" ;

# some tries at using display to report results in tables

option display\_transpose -5 , display\_width 132 , display\_lcol 50 ;

display Combined\_Cost, Expected\_Recourse\_Cost, Expected\_Idle, Expected\_Outsource, Expected\_Idle\_Cost, Expected\_Outsource\_Cost ;

display { h in HORIZON } : { c in CARS} Nocars [h,c] ;