Homework Collection 2

Contents

1. Homework 8: AMPL Exercise 18-5

A caterer has booked dinners for the next T days, and has as a result a requirement for a certain number of napkins each day. He has a certain initial stock of napkins, and can buy new ones each day at a certain price. In addition, used napkins can be laundered either at a slow service that takes 4 days, or at a faster but more expensive service that takes 2 days. The caterer's problem is to find the most economical combination of puchase and laundering that will meet the forthcoming demand.

Part A

Problem

The book provides the following decision variables:

```
var Buy[t]        # clean napkins bought for day t
var Carry[t]      # clean napkins still on hand at the end of day t.
var Wash2[t]      # used napkins sent to the fast laundry after day t
var Wash4[t]      # used napkins sent to the slow laundry after day t
var Trash[t]      # used napkins discarded after day t
```

There are 2 collections of constraints on these variables, described as such:

1. The number of clean napkins acquired through purchase, carryover and laundering on day t must equal the number sent the laundering, discarded or carried over *after* day t.
2. The number of used napkins laundered or discarded after day t must equal the number that were required for that day's catering.

Solution

For this problem I'll be building an AMPL model from scratch. The goal here was to build a set of parameters that would play nice with the given variables.

Parameters

```
Set DAYS := 1..T;

param T integer > 0;
param initial_stock integer >= 0                # napkins available for start of day
1
```

```
param demand{DAYS} integer >= 0;               # Number of napkins needed on day t
param napkin_price > 0                         # Buy price per single napkin
param wash2_price > 0                          # Price per napkin for fast laundry
param wash4_price > 0                          # price per napkin for slow laundry
```

Here we try to keep things simple. I make sure to set the days as an integer and let the prices for the napkins and laundering to be floats. Note here that I assume the price of trashing napkins is free.

Variables

```
var Buy{t in DAYS} integer >= 0      # clean napkins bought for day t
var Carry{t in 0..T} integer >= 0    # clean napkins still on hand at the end of day
t. 0 included for initial stock
var Wash2{t in DAYS} integer >= 0    # used napkins sent to the fast laundry after
day t
var Wash4{t in DAYS} integer >= 0    # used napkins sent to the slow laundry after
day t
var Trash{t in DAYS} integer >= 0    # used napkins discarded after day t
```

Here I make the decision to force only integer values for the napkins. We allow `Carry` to index on 0 so it can function as our initial inventory.

Objective Function

```
minimize Total_Cost:
        sum {t in DAYS} (napkinprice * Buy[t]
        + slowprice * Wash4[t]
        + fastprice * Wash2[t]);
```

Nothing too crazy here. We add up all the stuff that costs money and try to minimize our spend.

Constraints

```
# The number of clean napkins acquired through purchase, carryover and laundering on
day t
# must equal the number sent to laundering, discarded or carried over after day t.
# Basically, this check ensures that we're tracking all the untrashed napkins
correctly
subject to HandlingInventory {t in DAYS}:
        Buy[t]
        + Carry[t-1]
        + (if t-2 >= 1 then Wash2[t-2] else 0)
        + (if t-4 >= 1 then Wash4[t-4] else 0)
        = Demand[t] + Carry[t]

# The number of used napkins laundered or discarded after day t
# must equal the number that were required for that day’s catering.
subject to UsedNapkins {t in DAYS}:
        Wash2[t] + Wash4[t] + Trash[t] = demand[t];
```

The interesting part with the constraints is making sure that the scheduling for laundering works out appropriately. We want to avoid any indexing issues. Aside from that though I just follow the contraints to the letter given in the problem statement.

Part B

Problem

Formulate an alternative network linear programming model for this problem. Write it in AMPL using node and arc declarations.

Solution

This one took me a little while to figure out. According to the book, as a general statement, we use `arc` and `node` to take the place of `var` and `subject to` respectively. So there's a lot of reworking here. The objective function simplifies a lot here as well.

For starters, the parameters all stay the same.

For the nodes, I visualized it as having one per day, a node for trash and one for the store we buy the napkins for. I also added another node that functions as our day 0 carry.

```
node Day {t in DAYS}: net_in = demand[t];
node Trash;
node Stock: net_out = initial_stock;
node Store;
```

Our objective function is dead simple.

```
minimize Total_Cost;
```

This is all we need. The arcs will handle the math on this one. Lastly we got the bulk of the changes, the arcs. The arcs represent all the ways the napkins can flow. They can move from day to day, to the trash, and from the store. There are also the different ways they can flow through the days with the laundering. It's a weird set up to wrap your head around initially but it's very elegant.

```
arc InitialNapkins,
    from Stock, to Day[1], obj Total_Cost 0;
arc Buy {t in DAYS},
    from Store, to Day[t], obj Total_Cost napkin_price;
arc Carry {t in 1..T-1},
    from Day[t], to Day[t + 1], obj Total_Cost 0;
arc FastLaundry {t in 1..T-2},
    from Day[t], to Day[t+2], obj Total_Cost wash2_price;
arc SlowLaundry {t in 1..T-4},
    from Day[t], to Day[t+4], obj Total_Cost wash4_price;
arc TrashFlow {t in DAYS},
    from Day[t], to Trash, obj Total_Cost 0;
```

This is also where the objective function gets handled. All of these arcs have costs and the solver will handle them automatically with these statements.

Part C

Problem

The "caterer problem" was introduced in a 1954 paper by Walter Jacobs of the U.S. Air Force. Although it has been presented in countless books on linear and network programming, it does not seem to have ever been used by any caterer. In what application do you suppose it really originated?

Solution

There is an interesting article on this man titled **Eloge: Walter W. Jacobs, 1914-1982** by Joseph Blum et al. written in 1984. It actually brings up his history in the Air Force and the exact motivation behind this problem. To oversimplify, the problem involved the engines of aircraft.

1. Engines are procured for aircraft.
2. An engine flies a certain number of hours and then is shipped to an overhaul facility. The shipment can be done by slow surface transportation or fast airlift. The fast method is assumed to be more expensive.
3. At some time the aircraft are phased out, and an inventory of engines must be written off.

Robert L. Kirby

This is a very similar setup, it's kind of funny. It's pretty easy to see how this was modified to become napkins and laundry. This is unsurprising, many mathematical advances in optimization came about because of military applications.

Full source of the article: Annals of the History of Computing, Volume 6, Number 2, April 1984.

Part D

Problem

Since this is an artificial problem, you might as well make up your own data for it. Use your data to check that the formulations in (a) and (b) give the same optimal value.

Solution

2. Homework 9: AMPL Exercise 20-4